Efficient Data Center Architectures Using Non-Volatile Memory and Reliability Techniques

by

David Andrew Roberts

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2011

Doctoral Committee:

Professor Trevor N. Mudge, Chair
Professor David Blaauw
Professor Dennis Sylvester
Associate Professor Scott Mahlke

© David Andrew Roberts 2011
All Rights Reserved

ACKNOWLEDGEMENTS

I'd first like to thank my research advisor, Trevor Mudge, for his continued support and sharing of his expertise while being flexible enough to let me explore alternative research directions that I found interesting. My committee members David Blaauw, Scott Mahlke and Dennis Sylvester have also had a major influence on the ideas that shaped my thesis. Profs Todd Austin, Mark Brehob and Thomas Wenisch provided valuable advice and collaboration on several of my research papers.

The learning experiences enabled by my managers and colleagues during internships at HP Labs and ARM Ltd. were very rewarding. I'd especially like to thank Parthasarathy Ranganathan, Jichuan Chang and Mehul Shah at HP whom I collaborated with on the "Nanostores" project during the summer of 2009. Partha, Jichuan and I developed the evaluation methodology and system design for this energy-efficient data center architecture. During the initial brainstorming sessions, Mehul provided us with valuable insights on large-sale workloads and the types of system architectures currently used to solve these data-oriented problems. All three contributed significant effort in enhancing and editing our conference paper submissions, for which I am very grateful. Access to their computer resources during my time as a visiting researcher allowed me to continue to perform simulations and run-time measurements using a large number of networked servers.

At ARM I had the pleasure of working with David Bull, Krisztian Flautner, David Flynn and Dipesh Patel on several energy-related projects. Prior to starting my Ph.D

studies, they exposed me to CPU energy efficiency concerns and dynamic voltage scaling (DVS), controlled by their Intelligent Energy Manager (IEM) software. I worked on the first implementation of IEM based on Krisztian's thesis work, then went on to explore Razor timing-fault tolerance and fault-tolerant "Block Grouping" caches under their funding while at Michigan.

My fellow graduate students provided inspiration and entertaining discussions, and I'd like to thank Geoff Blake, Shidhartha Das, Ron Dreslinski, Dave Fick, Taeho Kgil, Seokwoo Lee, Yuan Lin, Dave Meisner, Sujay Phadke, Sangwon Seo, Korey Sewell, Denny Vandenberg, Ilya Wagner, Mark Woh and Bo Zhai, among others. I owe much of my success to my girlfriend Dan Chang and the rest of my family for their support and encouragement over the years. I'd also like to thank my friends for giving me the chance to step back from the focus of study and have some fun. Thanks to Bill and Kim Battle, Mike Morelli, Dan O'Karma, Gary Wallen and the rest of the Ann Arbor chapter of the Ride Motorcycle club. And also thanks to musicians Kevin Kapalla, Mumtaz Karatas, Roy Schmidt and Greg Scott for many enjoyable jam sessions, performances, guitar lessons etc. Elson Liu and Adam Zettel, you were great roommates.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

Efficient Data Center Architectures Using Non-Volatile Memory and Reliability Techniques

by
David Andrew Roberts

Chair: Trevor N. Mudge

The cost of running a data center is increasingly dominated by energy consumption, contributed by power provisioning, cooling and server components such as processors, memories and disk drives. Meanwhile, emerging classes of complex data center workloads place a heavier burden on processing and storage hardware, involving accesses to huge datasets for each operation. Fortunately, emerging technologies promise better performance and efficiency. Non-volatile (NV) memories for applications such as disk caches [7, 4, 2, 8] are proven ways to save energy, and in recent developments, byte-addressable persistent storage such as phase-change memory (PCM) or Memristors can serve as both main memory and permanent storage, reducing data transfers between layers of hierarchy. Further, 3D die-stacking provides a low-energy high-bandwidth means of connecting storage with computation

hardware. The challenge lies in how to optimally combine and balance system elements when data-center workload demands vary significantly. Once combined, new, inherent drawbacks such as limited memory write endurance need to be countered. Further, as processors often dominate system power consumption, they become a critical target for energy optimization. Unfortunately, current CPU architectures cannot fully exploit voltage scaling due to the need for safety margins [9] as well as having large caches that fail at higher voltages than the logic circuits [10].

In this thesis, we address these challenges via the following novel techniques;

- We propose a distributed, energy-efficient data center architecture [11, 12], replacing hard disk drives and DRAM main memory with non-volatile Memristors or PCM. The system is composed of a network of uniform building blocks called Nanostores that combine processors with a permanent data store. To reduce unnecessary data movement, DRAM and disk layers are eliminated, resulting in a flattened memory hierarchy.

- Because NV memories wear out with the number of data writes, we propose novel wear-leveling solutions. First we propose distributed data center wear-leveling [13] to address SSD-based and future Nanostore based storage, with a 3.9x improvement in lifetime. Second, we propose server-level reliability improvements for Flash memory based disk caches [4] that provide 20x improve-

ments in lifetime on average.

- We propose a novel on-chip cache fault tolerance scheme that allows more than a 30% improvement in energy efficiency [14, 15, 16].

CHAPTER I

Introduction

This thesis proposes that new architectures at the chip, server and data-center levels have the potential to save energy and increase performance by significant factors. Although optimization at any one level provides some of the benefits, the improvements multiply when combined at the system level. In the era of "cloud computing" where applications are provided as a globally accessible service that accesses persistent data, these are critical issues. The data centers which house the computing, storage and associated cooling infrastructure for these services are now being designed around minimizing cooling costs. For example, there have been studies on operating at different temperatures [17] and special building designs for improved cooling [18]. These cooling costs are directly related to the power consumption, so minimizing energy per unit of work (efficiency) can bring significant cost reductions. Meanwhile, the applications themselves have to process and combine multiple discrete data items in a single operation, for example, combining information about multiple friends on a Facebook web page, or correlating the raw text and images of multiple websites [19]. To handle this type of data, alternative storage such as DRAM (instead of hard disk) provides a higher performance, but expensive alternative.

Fortunately, these demands come at a time when technology is becoming available

that may overcome the limitations of conventional architectures. Flash-based solid-state disks have been around for several years and are gradually being adopted to augment or replace hard disks due to their performance and efficiency. Further out on the horizon are competitors to DRAM and SRAM including phase-change memory (PCM), Memristors (MRT) and MRAM (Magnetic RAM) which promise future scalability, higher capacities and lower energy than current storage. Most of these non-volatile memories have an important limitation - finite write endurance - that has been addressed through write reduction, wear-leveling and fault-tolerance schemes. However, because we want to take advantage of these devices in the data center, this thesis proposes mechanisms for system-wide wear-leveling rather than conventional leveling inside one SSD only. Finally, we identify that especially for current systems, and even for our new architecture, the CPU often consumes the largest fraction of system power. To address this critical component, we enable an existing energy-saving technique (voltage scaling) to work even more effectively, by reducing the voltage at which on-chip memories fail.

This section describes the changing data center landscape in detail, from system architecture to workload types. We go on to examine emerging technology trends and architectural device parameters, then specify the challenges which this thesis overcomes.

1.1 Data Center Trends

1.1.1 Energy-efficient Data Center Architectures

Energy consumed by data centers is a significant contributor of costs to companies and the environment [20]. For example, a survey estimates that over a third of operating costs in 2010 are on power and supporting infrastructure (power distribution and cooling) (see Figure 1.1). In an attempt to reduce these costs, recent research

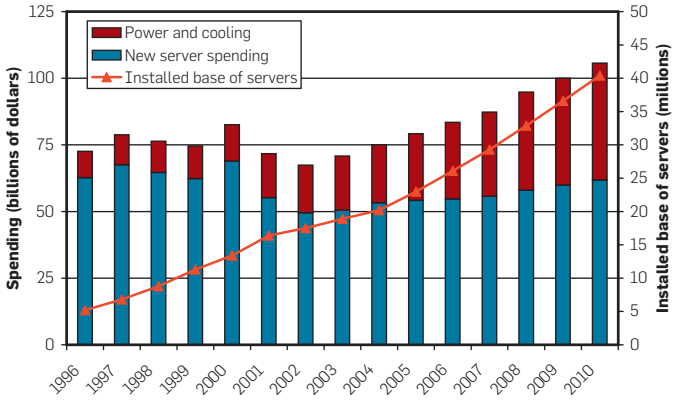


Figure 1.1: IDC survey indicates increasing cost of power and cooling [1, 2]

has focused on several different approaches to reduce energy footprint. These include aspects such as data center cooling schemes and efficient power supply provisioning. However, this thesis focuses on the server and networking hardware components. In this area, initial work has focused on choosing the right balance of system components to prevent energy waste. For example, simple processors remove circuit complexity and increase efficiency for workloads that perform large amounts if I/O and have low instruction-level parallelism (ILP). Also, the use of solid-state disks instead of hard disks yield further benefits in idle and active power. Components are generally provisioned to be fully utilized. For example, an idle hard disk still consumes a large fraction of full power when not being accessed, unless it is switched to a low-power state with long wake-up delay. A typical study of this type is JouleSort [21] which employs a low-power CPU with 13 laptop disk drives to sort a fixed amount of data using minimum energy. More radical system architecture changes take component integration even further and utilize low-power cores with emerging memory technologies. Low-power cores are viable for storage I/O intensive workloads because they do not need to perform complex processing on data (reducing circuit complexity), and can run at a low frequency and voltage. Flash based storage is better than hard

disk because is has almost no idle power (versus up to 80% of peak for hard disks) and lower access energy per bit. For example, FAWN and Gordon [22][23] approach the problem by combining low-power, simple processors with energy-efficient Flash memory. Gordon represents a distributed network of nodes consisting of a large amount of Flash memory for storage and a small amount of DRAM along with a power-efficient 1.9 GHz Intel Atom processor. They emphasize a 75% reduction in power consumption combined with 900 MB/s of storage bandwidth over conventional designs. FAWN is a similar system but uses 500 MHz cores and is designed specifically for a key-value data store. While these two designs derive their benefits from low-power cores and Flash-based storage, there is another performance-driven end of the spectrum. RamCloud [24] is another architecture trend whose energy efficiency has not been widely studied. A RamCloud uses distributed DRAM main memory to hold all persistent data instead of disk drives. It is argued that because of the very low latency and high bandwidth of DRAM, energy per operation can be orders of magnitude lower than hard disks based systems, and 5-10x better than Flash SSD systems. Our results also demonstrate that applications which are not bottlenecked on other system components such as the network do have better energy per operation than disk based systems. Our Nanostore design provides even greater benefits however, by combining non-volatility with low latency and high bandwidth similar to DRAM.

1.1.2 Workloads and Datasets

The amount of data being created is exploding, growing significantly faster than Moore's law. For example, the size of the largest data warehouse in the Winter Top Ten Survey has been increasing at a cumulative annual growth rate of 173% [25]. The amount of online data is estimated to have risen nearly 60-fold in the last seven

years [26]. Data from richer sensors, digitization of offline content, and new applications like twitter, search, etc., will only increase data growth rates. Indeed, it is estimated that only 5% of the worlds offline data has been made online so far [27]. This growth in data is leading to a corresponding growth in data-centric applications that operate on data in diverse ways (capture, classify, analyze, process, archive, etc). Compared to traditional enterprise workloads (e.g., online transaction processing, web services), emerging data-centric workloads change a lot of assumptions about system design. These workloads typically operate at larger scale (hundreds of thousands of servers) and on more diverse data (e.g., structured, unstructured, rich media) with I/O intensive, often random, data access patterns and limited locality. In addition, these workloads have been characterized by a lot of innovations in the software stack targeted at increased scalability and commodity hardware (e.g., Google MapReduce/BigTable). MapReduce [28] is a software framework that breaks workloads into "Map" and "Reduce" functions. During the map phases, data is separated into small units and each one is processed on a different worker node. The results are sent back to a master node where the Reduce function extracts the final answer from all of the sub-problem results. BigTable is the distributed database system used to handle data manipulation by MapReduce tasks. As for the applications themselves, emerging data center workloads have diverse requirements in multiple dimensions. For example, some impose a response time constraint e.g. Web searches, while others may be background tasks, such as data compression and de-duplication. Other dimensions include compute complexity, I/O intensity, data access locality and volume of data to be processed in a single operation. These application requirements significantly influence the best system architecture for performance or energy efficiency. Further, we would like to ensure that the data center architecture that we propose provides benefits for all types of workload. To ensure this, in Section 3.2.1

we propose a taxonomy of the relevant dimensions that a representative data center benchmark set should cover. We then use benchmark models that we identified to optimize our proposed efficient data center design in Section 3.3.3.

1.2 Technology Trends

1.2.1 Storage Technology and I/O interfaces

Historically, hard disk drives have been the major persistent storage device used in data centers. They are available in a wide range of capacities up to around 2 GB, and with multiple rotational speeds to trade off latency, bandwidth and cost. With a relatively high random seek time of several milliseconds, sequential accesses provide higher performance. A further drawback of hard disks is that even when idle, a large fraction of peak power is consumed by the spinning disk. Although disks can be "spun down" during idle periods, the long spin-up times reduce energy savings. More recently, NAND Flash-based solid-state disks (SSDs) have been made available, offering random access times of tens of microseconds, with much lower idle power. The highest-performing SSDs are composed of single-level (SLC) Flash which stores one bit per cell. Cheaper, denser Flash chips using multi-level cells (MLC) can be used at the price of roughly double the read latency and triple the write latency for 2-bit-per-cell MLC. Clearly, NAND Flash is most suitable for data where high IOPS (I/Os per second) are required. Due to the price premium of an SSD, hybrid systems are likely to be more common in the near-term.

Current consumer-grade hard disks and solid-state disks employ serial-ATA (SATA) interfaces, allowing throughputs of up to 3 Gbit/s or 6 Gbit/s for revision 2 or 3 of the standard. This narrow channel can act as a bottleneck for SSDs, preventing the peak theoretical throughput of the storage medium from being reached. Therefore, multiple drives on separate channels can be used together for higher bandwidths. For

more I/O intensive workloads that can benefit from greater bandwidths, PCI-express based I/O cards based on NAND Flash memory [29] provide bandwidths up to 1.5 GB/s. These bandwidths are still very low relative to the memory interface of a typical computer system, and present a bottleneck for workloads that spend most of their time moving data with little or no local manipulation.

The usage models pursued by industry and academia for NAND Flash integration can be categorized as follows [2].

- Extended system memory usage model: A NAND Flash memory module is connected to the current system memory interface or to a dedicated Flash memory interface.
- Storage accelerator usage model: A NAND Flash PCI express card is connected to the PCI express interface.
- Alternative storage device usage model: A Solid State Drive (SSD) replaces or augments the hard disk drive. It is connected to the disk interface. An example would be a SATA SSD.

	Primary powersavings	**Secondary powersavings**	**Hardware complexity**	**OS kernel modification**	**Application modification**	**Comments**
Extended system memory	DRAM	Disk	Minimal	Medium	None	Extend kernel memory manager to manage Flash devices
I/O accelerator	Disk	DRAM	Medium	Medium	Yes	Need to build I/O accelerator driver stack
Alternative storage device	Disk	DRAM	High	Minimal	None	Need to implement filesystem for Flash

Figure 1.2: Comparison of flash usage models in a server [2]

Each usage model presents a unique set of benefits and challenges. Figure 1.2 qualitatively captures them.

The “extended system memory” usage model presents Flash as a part of the system memory. It addresses the rising contribution of power consumed by DRAM in

addition to the electrical constraints limiting the integration of more system memory. For example, to increase storage capacity without having to reduce the operating frequency of the memory channel, [30] packs more DRAM onto each DIMM module. Using denser memory such as Flash or PCRAM may serve a similar purpose. However, this usage model requires modification to the operating system kernel. Specifically, the current implementation in the kernel memory manager that supports non-uniform memory architectures needs to be aware of the unique organization and behavior of Flash. Flash reliability management can be performed by the kernel memory manager with the assistance of the Flash controller.

The "storage accelerator" usage model presents Flash as a PCI express device that can be directly managed by the user application. This usage model allows the server application to manage Flash directly as a cache that stores frequently accessed code and data. It reduces the number of accesses to the hard disk drive thereby reducing overall disk power. Further, it may also be used as a way to implement the "extended system memory" usage model but with several drawbacks such as higher latency, lower throughput and added complexity in managing Flash. Flash management is distributed across the user application, device driver stack and the Flash PCI express card firmware. To truly leverage Flash as a "storage accelerator," the user application should be Flash aware. A device driver stack needs to be implemented to support the PCI express device. The device driver stack needs to implement device sharing mechanisms such that other con- current user applications and kernel components can make use of it simultaneously. In Fusion-IOs Solid State Storage [29]— they have also shown the "storage accelerator" usage model can expose the Flash PCI express device as an SSD by providing disk emulation features in the device driver stack.

The "alternative storage device" usage model presents Flash as an SSD that re-

places a hard disk drive. This usage model improves the latency and throughput to disk and reduces overall disk power consumption in a data center. With appropriate filesystems such as ZFS [31] it improves storage device scalability in a data center. Flash reliability management is performed by the Flash device controller in the SAN or NAS. However, this usage model also requires modification in the kernel, and a complex Flash device controller that is capable of performing intelligent Flash reliability management. A customized filesystem needs to be implemented to fully take advantage of the benefits of Flash. Further, this usage model ties itself to the non-Flash aware features that are found in a hard disk drive interface protocol such as SATA. Flash-specific commands have only just been implemented in the disk drive standards, such as the TRIM command which informs an SSD when certain blocks are no longer needed, assisting in garbage collection. On the other hand, the operating system in the "extended main memory" model has full visibility of memory page classification and activity statistics that can be used for more intelligent mapping of data to Flash. Servers clearly benefit from all three usage models that essentially integrate Flash as a faster hard disk or disk cache. All usage models help (1) reduce unnecessary standby power from hard disk drives and (2) improve overall throughput by reading and writing from disk cache instead of a hard disk drive.

This thesis proposes to employ the "extended system memory" storage model to reap the benefits of using the same physical medium for memory and persistent storage. When moving from Flash-based storage to an emerging resistive memory such as PCRAM, the storage cells already have similar properties to the system memory. Indeed, several proposals have been made for using PCRAM as a DRAM main memory replacement [32, 33, 34]. It is a logical extension of this model to combine main memory and data storage in the same device, minimizing inter-device data transfers and maximizing communication bandwidth.

1.2.2 Networking and Communication

Different communication paths are fundamental to data center systems, and we described the storage layer in the previous section. Inter-server data center networks can currently contribute up to 50% of cluster power [35]. Therefore it is important to consider the network as both a potential performance bottleneck and power consumer, as we do in this thesis. Modern Ethernet networking can provide 40-100 Gbit/s per link (40/100GBASE). In terms of topologies, many have been explored, including more traditional folded clos (fat-tree) [36, 37] as well as proposals for other designs such as flattened butterfly [35]. This design supports the same bisection bandwidth (peak bandwidth when network is split into two equal parts) with fewer links and router chips. Further, there have been proposals for dynamically adjusting the bandwidth to enable power scaling (power proportionality) in network interfaces (NICs) and switches. This is particularly useful for energy saving where bandwidth demand varies over a period of time.

Optics have also been studied for on-chip communication [38, 39], promising higher performance for connecting cores and memories. However, we do not consider on-chip optical networking for this thesis because practical implementations are likely to be beyond our target 2015 timeframe. However, we do exploit 3D die stacking technology [40, 41, 42, 43, 44]. Instead of using an external bus that is restricted by pin count, through-silicon vias (TSVs) are used to connect die together. By using a short, wide interconnect between processors and memory, higher bandwidths and lower bus energy can be achieved. In addition, it allows tight integration of devices with different process technologies, such as logic and DRAM processes.

1.2.3 Emerging memory technologies

Charge-based memories such as DRAM and Flash memory are reaching scaling limits [3] where manufacturing solutions beyond a 2012 time-frame are not known. This is due to the decreasing amounts of charge that can be stored in any bit cell. Fortunately, resistance-based memories such as Phase-Change memory (PCM) are in production and do not depend on charge storage to record data values [32, 45]. Instead, PCM employs a chalcogenide glass material that can change between crystalline and amorphous states when heated, correspondingly changing its resistance. The major drawback with PCM is a high programming current. Other resistive RAMs (RRAM) [46] may help to overcome the high write-current limitations of PCM. The Memristor [47], for example, is predicted to have much lower write energy and higher density due to a multi-layer crossbar architecture [43, 48].

Each technology has system-level drawbacks, such as high latency (Flash), high idle power (DRAM) or high write energy (PCM). We consider these architectural and power parameters during our data center modelling and optimization process. In this work we consider DRAM, PCM and Memristors as the main memory and primary data store, with their different latency, energy and density characteristics, to determine the best data center configuration for each one.

1.3 Challenges

1.3.1 Overcoming the inefficiencies of existing storage devices

System memory power (DRAM power) and disk power contribute as much as 40% to the overall power consumption in a data center [8]. Further, current trends suggest that this will continue to increase at a rapid rate as more DRAM and disk drives are integrated to improve throughput. Both technologies also have a signifi-

cant idle power component, further reducing efficiency. There are emerging memory devices in the technology pipeline that may address this concern. These devices typically display high density and consume low idle power. Flash, Phase Change RAM (PCRAM), Memristors and Magnetic RAM (MRAM) fall into this class. Flash is an attractive technology that is already deployed heavily in various computing platforms. Today, NAND Flash can be found in hand-held devices such as smart phones, digital cameras and MP3 players. This has been made possible because of its high density, low power properties and non-volatility. Its popularity has meant that it is the focus of aggressive process scaling and innovation. The rapid rate of improvement in density has become the primary driver to consider Flash for other usage models. There are several Flash usage models in the data center that are currently being examined by industry and academia to address rising power and cooling costs, among other things. Recently, PCRAM has received much attention because of the challenges Flash faces when we scale below the 22nm process technology node. Studies [3] have shown that PCRAM is expected to scale better than Flash post 22nm and emerge as an important and widely used memory device. It currently has DRAM-like read latencies and has been proposed as a DRAM replacement. Memristors [47] are a memory technology predicted to behave similarly to PCRAM with potential for even higher capacity than PCRAM due to their stacked crossbar structure. Although their basic parameters may be better than DRAM or Flash, the most effective ways of integrating them in a system still need to be explored.

1.3.2 Breaking the I/O bandwidth barrier

In Section 1.2.1 we mentioned that the fastest PCI-express based Flash storage cards can achieve around 1.5 GB/s bandwidth. However, modern off-chip DRAM interfaces can achieve over 10 GB/s of throughput, while on-chip 3D-stacked processor

designs such as [40] achieve over 32 GB/s of bandwidth. As an extreme example, a 3D-stacked DRAM-PCRAM checkpointing system was designed for 2.5 TB/s transfers [49]. By opening up the bandwidth per unit of capacity, it may be possible to reach new levels of performance and efficiency in the data center. This in itself raises issues of balance in provisioning processors and networking, as well as reaching other constraints such as power density.

1.3.3 Modelling and optimization at data-center scale

Detailed performance and power modelling is possible for individual servers, processing and storage devices. However, when modelling systems that are scaled up to Petabytes of storage and thousands of compute nodes, detailed full-system simulation requires unrealistic amounts of time and hardware. Simplified models may run faster at the cost of some accuracy. Therefore, choosing the right methodology to draw meaningful conclusions about data center designs remains a major challenge.

1.3.4 Overcoming non-volatile memory endurance limitations

With growing numbers of Flash based SSDs in use, as well as emerging technologies such as resistive memories, there are concerns over whether the memories will last long enough to be viable solutions. The majority of NV memories can only sustain a limited number of re-writes per bit before becoming unreliable (around 10^5 for NAND Flash and 10^8 for PCRAM, for example). Further, due to the very low latency of emerging NV memories, the write limit can be reached in a very short time and can be exploited by viruses, for example. These issues have been addressed to some extent using wear-levelling techniques [50, 51] whereby writes are spread across memory locations as evenly as possible within a chip or SSD. However, in the data center, there is significant imbalance of write-bandwidth between individual

SSDs. This makes effective wear-levelling difficult to perform in an ensemble, and no solutions currently exist to address this.

1.3.5 Breaking CPU voltage-scaling limits

Processors consume a large fraction of energy in the data center, and improved efficiencies in this component will have a significant overall impact. To minimize power, an appropriate power supply voltage is used at a given frequency, including a safety margin to accommodate power supply noise and process variations. However, processor architectures cannot fully exploit voltage scaling due to the need for safety margins [9] as well as having large caches that fail at higher voltages than the core logic [10]. On-chip SRAM caches operating at the same voltage as the rest of the core can limit voltage reduction at lower frequencies, restricting chip-wide energy savings. As voltage is reduced, increasing numbers of bits fail to operate, but gorrecting them with conventional error-correcting codes (ECC) can introduce intolerable latencies. There is clearly a need for low-latency fault-tolerance schemes to maintain a functional cache with small latency overhead.

1.4 Contributions

1.4.1 Re-architecting the storage hierarchy with Nanostores

This thesis proposes and evaluates a novel data center system architecture that enables higher energy efficiency and performance to be achieved compared to conventional disk-based or DRAM-based architectures. The key features of the design are a uniform storage building block consisting of processing cores, a network interface, 3D-stacked interconnect and a unified data store/main memory device. By allowing I/O to take advantage of the high bandwidth normally reserved for main memory alone, emerging I/O intensive workloads get significant performance and

energy-delay-product (EDP) increases over an aggressively optimized baseline. The unified storage design also removes the need for data copying between disk and RAM. Further energy efficiency is gained from the use of low-power cores and efficient storage technologies. The nodes or "Nanostores" are networked in a conventional fat-tree configuration and appear to software as a conventional cluster of networked servers.

1.4.2 Extending wear-leveling to distributed storage

We propose extending wear-leveling algorithms normally constrained to a single storage drive to the entire data center. This is required to support the energy efficiency improvements of distributed non-volatile storage (SSDs or Nanostores). By exploiting typical access patterns present in data center disk workloads, we are able to significantly increase the time to first failure (TTFF) of a data center storage ensemble. Further, we propose novel techniques to increase the performance and lifetime of MLC Flash memory devices installed in servers. These include dynamically switching between SLC and MLC modes for pages of Flash, and progressively increasing the number of correctable errors to minimize average access latency.

1.4.3 Overcoming the limits of CPU voltage scaling

Because CPU energy is a significant power consumer for several data center workloads (especially in non-optimized conventional designs), we focus on techniques to reduce CPU power. Our first contribution was a voltage control system for a delay-fault-tolerant processor that can remove voltage safety margins [52, 53, 54]. The second (which is the focus of this thesis) is a low-latency cache fault-tolerance scheme that allows whole-chip energy savings by enabling cache operation at reduced voltages [14, 15, 16].

1.5 Thesis Structure

This thesis is structured as follows. Chapter II presents prior work related to data center architecture, emerging technologies and memory reliability. Chapter III presents our proposed Nanostore data center architecture and evaluation results. Chapter IV goes on to examine how memory reliability can be addressed in distributed non-volatile storage for the Nanostore design or SSDs. It also presents lifetime enhancements made specifically for NAND Flash-based disk cache applications. Chapter V completes the energy-saving techniques with a scheme to lower CPU voltage and save energy in one of the most power-hungry data center components. We conclude in Chapter VI.

CHAPTER II

Background and Related Work

In this section we provide background information on related data center architectures and the emerging technologies used in our proposed designs. This includes projections for future non-volatile memory device parameters and process scaling.

2.1 Efficient Data Center Architectures

Several of the principles leveraged in our Nanostore design have been studied in prior work. The iRAM and PIM work [55, 56] examine integrating a processor with DRAM in the same process, and consider benefits with vector streaming programming models. While thematically similar, our implementation is significantly different in several ways, including the use of lower-power commodity processors with a diskless flat hierarchy based on non-volatile memory, and a distributed software model. Active Storage [57] incorporates compute closer to disk, but only in the form of more powerful disk controllers for offloading and streaming. The main processor is still a deep hierarchy away. Recently, the RAMCloud project [24] has proposed distributed systems where all data resides in DRAM. Their planned research primarily focuses on the software stack, around low-latency remote procedure calls (RPC), durability, data model, scaling, and consistency. While several of their

motivating arguments are similar to ours, we differ in our assumptions around all data residing in 3D-stacked non-volatile RAM and in our architectural explorations around balanced system designs. Our results validate their projected performance but show even better benefits from our approach. Some recent studies have examined using lower-power (thin or wimpy) cores for energy efficiency [22, 58, 59, 21] while also being aware of the impact on quality of service [60]. There has also been prior work on ultra-low-voltage core design [61]. Like these studies, we also explore the benefits from better balanced designs, but synergistically in combination with rethinking compute-data proximity and hierarchy. Other studies have examined special-purpose architectures optimized for specific data-centric workloads including use of GPUs [62], FPGAs [63], and even ASICs [64] and co-design of hardware and software for data-centric workloads. (e.g., MonetDB [65], MapReduce [28]). Such optimizations would be applicable in our solution as well. Regarding data center workload classification, Google [66] classify groups of tasks that have similar workload demands. Tasks are classified by time in seconds, CPU usage in cores, and memory usage in gigabytes. Their main observation was that tasks have a bi-modal distribution between short-running and long -running. Short-running tasks are most common, while a few long-running tasks dominate execution. In contrast, our taxonomy captures more dimensions such as deadlines, access patterns and read/write ratio that have a clear impact on hardware requirements.

2.2 Emerging Technologies

2.2.1 Memory technology scaling trends and limitations

This thesis explores several existing and emerging memory technologies to assess their benefits in an optimized data center design. They can be broadly divided into charge-based memories (e.g. SRAM, DRAM, NAND Flash) and resistance-based

memories (e.g. phase-change memory (PCM), Memristors (MRT) and Magnetic RAM (MRAM)). In this section we examine scaling trends, timing and energy parameters for these devices based on published data.

SRAM is commonly used for processor caches due to its low latency and ability to integrate with logic on-die. We assume that SRAM is used for all on-chip cache memories due to its low latency and high endurance. Prior work has explored caches built from alternative or hybrid technologies [67], but with such high access rates and small capacities it is difficult to guarantee sufficient lifetime for some of these designs. For simplicity, we use SRAM caches in our Nanostore designs (Section III), and address SRAM cache reliability issues in Section V.

DRAM can be used for caches [68], main memories and data storage [24]. However, the scalability of charge-based memories such as DRAM is limited by physical factors such as the tiny amount of charge that must be stored and measured. DRAMs have an area advantage over SRAM which makes them more cost-effective. However, they need to be refreshed periodically, which increases their idle power consumption.

Figure 2.1 presents the scaling trends from the ITRS roadmap 2009 [3] for the memory technologies under consideration. For DRAM beyond 40nm and NAND Flash after 2011 (28nm), the limiting factors are dielectric material and thickness. Therefore, resistance-based memories such as PCRAM, Memristors and MRAM have the potential to scale further. The main limitations of PCM are a high programming current and relatively long programming time. However, because the quantity of phase change material decreases with each technology generation, both of these issues become easier with scaling [69].

Technology	Density	BW	R Lat.	W Lat.	R Energy	W Energy	End.
Units	$\mu m^2/bit$	GB/s	ns	ns	pJ/bit	pJ/bit	Wr/bit
HDD [70]	0.00006	0.5	3,000,000	3,000,000	2,500	2,500	-
Flash SSD	0.00210	1.0	25,000	200,000	250	250	10^5
DRAM DIMM [5]	0.00380	26	55	55	12.5	12.5	-
PCM (25nm) [32, 3]	0.00250	-	48	150	2	19.2	10^7
Memristor (18nm) [71]	0.00320	-	100	100	2	2	10^7
MRAM (65nm 512K) [72, 3]	0.0423	-	2.3	11.0	1.68	9.76	-
STT-RAM (32nm reg. file) [73]	0.0102	-	0.12	3.23	0.01	0.31	-
SRAM (32nm 4M) [73]	0.1495	-	2.36	2.36	2.48	2.48	-
STT-RAM (32nm 4M) [73]	0.0102	-	1.96	7.76	1.56	1.86	-

Table 2.1: Storage technology parameters (single-cell density, bandwidth, latency, energy and write endurance). We focus on non-volatile memories as candidate permanent data stores. The DRAM DIMM is included for baseline system modelling. The first three rows are estimates for an entire storage device rather than individual bit cells alone, projected to the 2015 timeframe, based on historical scaling trends. We assume 10 Watts total power per HDD and 2 Watts per SSD. Off-chip DRAM access energy is derived from [5]. The lower section contains published parameters from various sources. PCM and Memristor numbers were scaled according to ITRS'09 projections for the year 2015.

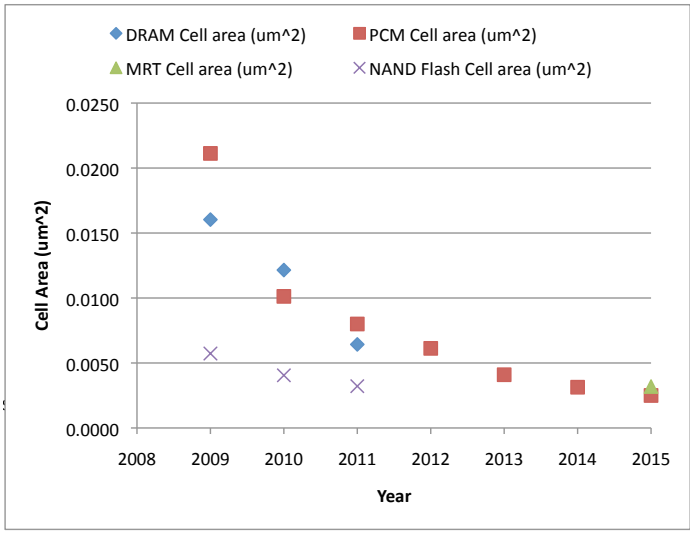


Figure 2.1: Predicted cell areas of different memory technologies [3] (DRAM and Flash from ORTC-2A, Phase-Change memory (PCM) from Focus C, PIDS5). Note that DRAM and NAND Flash do not have known manufacturable solutions for scaling beyond 2011 timeframe. For Memristors (MRT) we assume a feature size similar to NAND Flash projections, and cell area factor A=$10f^2$, based on direct discussions with HP. We do not show MRT predictions prior to the target year of our study (2015) due to the current experimental nature of the technology.

2.2.2 Non-Volatile Memories

Table 2.1 presents the characteristics of emerging non-volatile storage units and memory devices. The first three rows use historical scaling trends to predict aggressive performance and energy efficiencies for conventional storage media, scaled to the 2015 timeframe. These will serve as our baseline storage parameters, and are expected to over-estimate the actual efficiencies we will see in 2015. Similarly, we use 2015 projections for the PCM and Memristor parameters, which serve as our target technologies. It should be noted that the access energy/bit values for DRAM, PCM and Memristor have little impact on the overall results because their power contribution is dwarfed by other components such as processors and storage idle power (see Section 3.4.3). Approximate idle power for current-generation memory

Technology	Idle Power
Units	mW/GB
DRAM	100
PCM	1
NAND Flash	1

Table 2.2: Memory idle power [6].

chips is shown in Table 2.2. MRAM and STT-RAM are currently being explored as interesting candidates for future on-chip cache applications [72, 73], but are not as mature as PCM which has been commercialised and is being considered as a replacement main memory [32, 33, 44, 34]. For this reason, we omit these two devices from our system design exploration. Their parameters for MRAM and STT-RAM were included in the table for comparison, where they have much lower latency and similar access energy to the Memristor.

Previous studies have discussed different ways to use existing and emerging non-volatile memories e.g. Flash [23, 58, 74, 4], PCM [32, 33, 44, 34], and Memristors [48, 75, 47, 43, 76]. While Flash memories have been shown to be effective as storage, disk replacement, or disk cache [58, 29, 74, 4], their latency and endurance limitations make them inapplicable for our work. Recent work on PCM has examined its use both as Flash replacement and as memory replacement (including in 3D-stacked configurations [44]), but there have been no studies that have focused on simplification of the data hierarchy, in the context of data-centric workloads. Prior studies have prototyped and evaluated Memristors, but we are not aware of any architectural studies using Memristors. Several studies have proposed optimizations to improve endurance [32, 33, 44, 34] and others have identified potential improvements in the future [3].

2.2.3 3D die stacking

Recent architectural proposals have studied 3D stacking and demonstrated its viability and its benefits for improved bandwidth and memory redesign (e.g. [40, 44, 77]). Multiple die are bonded together either face-to-face or face-to-back, typically between a combination of logic and memory die. It is therefore possible to closely couple die built on different process technologies. Communication between die is achieved using through-silicon-vias (TSVs) which provide a very dense, low-capacitance communication bus. We propose to take advantage of the extra bandwidth afforded by 3D stacking as a means of connecting servers to their primary data storage and main memory in the datacenter. Bandwidths of 32 GB/s were demonstrated in [40] using a 1024-bit TSV bus, while [77] proposes a 4 KB bus to improve the single-thread performance of 3D-stacked processors by aggressively prefetching large amounts of data.

2.3 Memory Reliability

Memory reliability concerns vary depending on whether they are volatile or non-volatile. This thesis deals specifically with on-chip SRAM caches and on- or off-chip NV memories (Flash, PCRAM and Memristor). The SRAM failures of interest are non-persistent i.e. at a higher voltage, the SRAM cell may function correctly again. The NV failures are wearout-induced, when bits fail to reliably store the written value after a certain number of write cycles. This section covers related work for these two classes of memory.

2.3.1 Non-Volatile Memory Wear-Leveling in Data Centers

Wear-leveling and fault tolerance

The two main methods of dealing with wear-out are wear-leveling and fault tolerance. The first attempts to maximize the useful lifetime of the memory by using every storage location as much as possible before failures occur. The second attempts to work around failed bits after they have already stopped reliably reading or writing data. These two techniques have additive effects. Wear-leveling in NAND Flash SSDs is typically managed by the FTL. There are several FTL variants in existence that trade-off metadata storage requirements, performance and lifetime. The main functions of an FTL are address mapping (including allocation), garbage collection or cleaning (erasing blocks containing stale data and moving valid pages out of the erased blocks) and wear-leveling. The three basic address mapping schemes are page-level, block-level or hybrid [78]. For a page-level mapping, the FTL maintains a table that maps individual 512B disk blocks to physical pages on the SSD. This incurs a high overhead and requires a large mapping table. Block-level schemes use a coarser granularity (e.g. an entire erase block), reducing the mapping table size but requiring an entire block to be copied to a new block even if just one page is modified. Hybrid schemes maintain several log blocks which are page-mapped, targeted towards pages with temporal locality. This reduces mapping table overheads while reducing page copying overhead. Garbage collection is typically triggered by the free block capacity of an SSD falling below some threshold. When that happens, blocks are selected to be erased, where those containing the fewest valid pages incur the lowest cleaning time. Before erasure, any remaining valid data must be copied from the old block to a new block which is both time consuming and incurs additional writes. Wear-leveling is affected by the address allocation and garbage collection,

and wear-leveling algorithms are often an intrinsic part of the cleaning mechanism. Dynamic wear leveling only considers blocks that have been overwritten. However, some data is cold or static and may be ignored by this simple FTL, so static wear leveling sometimes swaps the active blocks with randomly chosen cold blocks [50]. This introduces migration overhead but balances wear more effectively. Despite the abundance of local wear leveling policies, there is no known prior work to address coordinated wear-leveling across servers.

Wear-leveling in distributed systems

Load balancing across servers has been widely studied for performance, power and cost reasons. As well as small-scale datacenters with a number of independent networked servers running different applications, infrastructures such as MapReduce [28] take advantage of distributed data and massive parallelism while also performing load balancing. With these emerging data-dominated workloads, managing wear-leveling on a distributed scale has been neglected. Although most storage drives are modular and replaceable, without proper monitoring and control, SSDs will fail at unpredictable physical locations and times. Replacement and data migration to new drives is therefore a time consuming process, although replication (at additional cost) can allow resilience to wear-out failure. We argue that with monitoring and control through distributed wear-leveling algorithms, the replacement, data restoration and replication costs can be reduced. The ideal distributed wear-leveling system should provide accurate estimates of remaining lifetime, as well providing a degree of control over failure times. The solution proposed in this paper is a step towards that goal, achieved by altering the write bandwidths to individual SSDs.

There have been a large number of papers on both SSD design [79] and wear-out mitigation for both NAND Flash such as [78, 50] and PCRAM devices [51, 34, 33, 32].

Several studies have assessed the benefits of hard disk replacement or augmentation with SSD file caches such as in [74, 4, 80, 81]. We go beyond the scope of these prior studies to consider wear imbalance across storage devices. Some works have considered wear-out on the scale of RAID arrays (Differential RAID) [82]. By distributing parity bits unevenly across the drives, they produce an age disparity so that drive failures are staggered. The Griffin system [83] uses hard disks as log-based write buffers for SSDs. For desktop workloads, writes to Flash are reduced by 49% on average. Such a design could be applied in front of our activity monitoring nodes for further lifetime benefits. Other work considers large-scale deployment of Flash memories to improve energy efficiency and performance such as Gordon [23] and FAWN [22]. Unlike our work, endurance was not considered in detail. The Write off-loading [84], SEA [85] and PEARL [86] systems consider data placement on disk arrays, for performance and power saving. In particular, PEARL redistributes data between SSDs and a hard disk that each one is associated with, reacting to changing access patterns. Periodically, blocks of data (zones) are classified as write-excessive, read-exclusive and read-write. If the frequency of writes exceeds a threshold, that data zone is placed on hard disk. As a result, wear-out is limited. Unlike this work, we examine a much larger system and longer traces with the primary focus on wear-leveling across servers, not within them. Our system model is different, with an emphasis on distributed systems that rely on network bandwidth for communication. We also examine the interaction of the system-level policy with the low-level wear leveling on each SSD. Our paper provides significant, non-trivial contributions beyond this prior work. First, the prior work proposes vertical movement of data between a single hard disk and a single paired SSD. Instead, we use horizontal movement across multiple SSDs, with no limitation on where we can move data. Second, they use a single threshold to determine if writes to Flash exceed a lifetime requirement. This will not

result in effective inter-SSD wear-leveling because, for any read-mostly disk that is below the write threshold, they cannot move blocks horizontally between hard disks to fully balance writes.

2.3.2 Cache Memory Fault Tolerance

Minimizing CPU operating voltage is important in improving the energy efficiency of data centers, where processors can contribute a large fraction of total power consumption. However, on-chip cache memories can restrict the minimum voltage at which a CPU chip can operate [10]. This is due to process variations introducing imbalance between transistor devices, leading to read, write, access time or retention failures [15, 16]. When the voltage is too low, increasing numbers of cells exhibit failures. Failed cells may require cache lines to be disabled, increasing the miss rate and decreasing performance and energy efficiency. One solution is to scale up the size of the SRAM cells, or use alternative (larger) SRAM cell designs. These solution require additional area, however. Another solution is to employ error detection and/or correction. Conventionally, simple error-correction codes (ECC) such as parity can only handle a few faulty bits, but aggressive voltage scaling introduces hundreds of failures. More powerful codes require a long time to decode, drastically reducing the performance of the critical cache component. Therefore, alternative low-latency fault-tolerance that can handle large numbers of errors at low voltage is required. The following publications are related to this problem space.

Pour and Hill [87] derive an analytical model of the performance loss of a set-associative cache given a set of defective blocks. They employ an extra valid bit per cache block to identify whether or not it is defective. Their key findings for caches of size up to 32 kB are that miss ratio increase is negligible unless a set is completely disabled by faults. In [88] they present a model to estimate memory-failure prob-

ability using combined row and column redundancy. The Power4 architecture [89] employs parity on L1 caches and Hamming codes on L2. In addition, L1 and L2 have spare bits, while L3 has redundant cache lines. If correctable error thresholds are exceeded, a cache line delete function allows up to 2 deletions per L3 cache. For defects detected at power-on BIST that cannot be handled, the L3 cache is disabled. The Nanobox [90] applies redundancy and other ECC codes to logic functions built using lookup tables. A technique for memory self-repair at high defect densities is presented in [91]. It relies upon prior knowledge of the polarity of the error (i.e. faults are always stuck at 0 or 1). In our cache application, the scheme will not work because the value read from faulty bits is unpredictable and can change with operating point (e.g. voltage or temperature). Agarwal et al. [92] noted that the number of defective cells and their location changes depending on operating voltage. In addition, they proposed a cache block re-mapping technique for direct-mapped caches. The technique relies on a defective block mapping table determined prior to execution using BIST. They consider the use of block re-mapping in conjunction with ECC and row redundancy.

The following related work was published after our first publication of a dynamically resizeable cache employing "block grouping" fault-tolerance [15][1]. To the best of our knowledge, our work was the first to propose segmenting cache lines, identifying faulty segments with a fault map composed of more reliable bit cells, then combining cache line segments to produce working cache lines. Further, our scheme reaps the majority of voltage scaling benefits relative to an unprotected cache. Referring to Figure 9 in [93], at a yield of 1 in 1000 caches being defective, the most basic variant of block grouping (word-fix) enables a voltage drop of over 0.35V, whereas the improved designs offer less than 0.1V of additional savings.

[1] Initially submitted to ISCA'06

Makhzan [94] presents a fault-tolerance scheme that saves energy on an entire chip by reducing the miss rate caused by lowering the voltage. They obtain 40% energy savings with performance under error conditions that is close to defect-free operation. As with our work, they maintain a defect map (in a high voltage region) with one bit per word in the cache, to identify where the faulty bits are. They also cache the recently accessed parts of the defect map, similar to our "selector bit caching." However, they use an external cache called the IDC that is used to store data that would otherwise be placed in defective cache locations. Wilkerson [10, 95] presents two fault-tolerance schemes with an energy per instruction (EPI) reduction of 53%. The schemes are word-disable and bit-fix, where word-disable is an implementation of block grouping [96] with adjacent cache lines paired together ("adjacent pairing"). This is the least complex variant to implement, but improved fault tolerance can be achieved through same-set or whole-cache pairing (see Section V). The cache is resized during a low-voltage mode to 50% or 75% of full capacity for each scheme, whereas ours is dynamically resized to allow fault-free lines to operate normally. The bit-fix scheme dedicates one way of the cache to storing patches which can replace faulty locations in the main cache. It is similar to the $Log(B)$ scheme which we proposed in [15], except that multiple patches can be applied per block, and the patches reside within a cache way. Instead of using a higher voltage domain or larger 6-transistor SRAM cell to store the defect map, their work proposes the use of an alternative SRAM cell design with more transistors. Koh [97, 98] presented the Buddy Cache and Salvage cache. The Buddy cache is almost identical to block grouping with "same-set" pairing, where one faulty cache line is paired with another in the set (its buddy), assuming that fault positions do not overlap (they are compatible [15]), to produce one non-faulty cache line. They require an additional table (the Buddy Map) to identify which compatible blocks can be combined (see

our "grouping table" [15]). Relative to the adjacent pairing used for word-disable [10], they see a less dramatic reduction in cache size as voltage is lowered. The Salvage Cache is a similar technique that uses a single non-functional block to repair several others, while disabling that block. In addition to the fault map, they add a victim map entry per cache line (instead of the buddy map) which stores the index of a block in the same set used to repair its faulty segments. Sasan [93] proposes the Resizable Data-Composer cache (RDC). The RDC considers each bank of the cache as a sequence, with one way of the next bank being used to replace defective words in the other ways. This improves performance as both banks can be accessed simultaneously. Ansari [99, 100] proposes a first scheme where compatible lines are grouped together, and one line is specified as the "sacrificial" line whose working segments are used to replace faulty segments in the other lines. The second design, called ZerehCache, partitions the lines into same-sized logical groups, each with one spare line. Logic circuits can shuffle the location of non-faulty line segments to make the lines compatible with each other. They formulate the problem of configuring the groups as graph colouring, and solve it with an approximate algorithm. Chishti [101] proposed multi-bit segmented ECC (MS-ECC), allowing dynamic cache resizing without relying on a fault map, and providing coverage of soft or transient errors. The scheme uses an Orthogonal Latin Square code on segmented cache lines to provide the low latencies needed for caches, at the expense of more check bits than some other codes. Abella [96] proposed a scheme for a set-associative cache whereby faulty segments of cache lines are disabled and recorded in a fault map. On a cache hit to a faulty segment, the line is evicted as if it were a miss. The faulty line is then promoted to most-recently-used, preventing it from being filled next time there is a miss to that line. Because the data in the faulty segment is likely to be accessed again in the near future, it will probably be stored in a different cache line, where

the corresponding segment is non-faulty. As a further means of ensuring consistent performance of applications, a scheme is proposed where all cache addresses are periodically re-mapped, averaging out any slowdown a workload has due to its data being mapped to faulty segments. Ipek's Dynamically Replicated Memory (DRM) [102] applied the concept of block grouping to non-volatile memories such as PCM. The memory is protected with a parity bit for every byte. As errors emerge in the bits due to wear-out, they are detected by the parity bits, and the faulty page is grouped with another compatible faulty page, under operating system control. They use the same greedy algorithm to select block pairs as in block grouping, where the first available block that is compatible with the current block is selected for pairing. Error-correcting pointers (ECP) [103] is another fault tolerance design for memories that wear out. It achieves better fault-tolerance than DRM [102] by using several repair pointers and associated spare bits per block, similar to our Log(B) scheme [15] and bit-fix [10].

CHAPTER III

Nanostores: Co-located Compute-Storage NVRAM-based Architecture for Data-Centric Workloads

The increasing gap between the speed of the processor and the time to access the data in the disk has historically been offset with deeper and larger memory hierarchies with multiple levels of SRAM, DRAM, and more recently, Flash layers for caching. However, recent trends that point to a potential slowdown of DRAM growth and the emergence of alternate resistive non-volatile memory technologies and properties of emerging data-centric workloads offer the opportunity to rethink future solutions. Specifically, we examine an approach that leverages both the memory-like and disk-like attributes of emerging non-volatile memory technologies. We propose a new architectural building block called Nanostores that co-locates computation with a single-level data store in a flat hierarchy, and enables large-scale distributed systems for future data-centric workloads. We present a new evaluation methodology to reason about these new architectures, including benchmarks designed to systematically study emerging data-centric workloads. Our evaluation results demonstrate significant potential for performance benefits from our approach (often orders of magnitude) with better energy efficiency.[1]

[1]This project was initiated at HP Labs and was a collaboration with Parthasarathy Ranganathan, Jichuan Chang, Mehul Shah and Trevor Mudge.

3.1 Introduction

Section 1.1 presented the increasing data center energy costs associated with power provisioning, cooling and server components such as disks and processors. Further, due to the explosion in data volumes involved in each application, data center infrastructure must scale-out to accommodate enough storage capacity and compute capability. Although many workloads are I/O dominated and may take advantage of devices such as simple processor cores to save energy, other workloads are performing increasingly complex processing on each data item [60]. Given these trends in data scale, workload diversity and the need for energy efficiency, it is clear that data center architecture should be revisited. Concurrently, significant changes are also expected in the memory industry (see Section 2.2.1). Recently, new non-volatile RAM (NVRAM) memory technologies have been demonstrated that significantly improve latency and energy efficiency compared to Flash and Hard Disk. Some of these NV memories, such as Phase-Change Memory (PCM) and Memristors have been demonstrated to have the potential to replace DRAM with competitive performance and better energy efficiency and technology scaling. At the same time, several studies have postulated the potential end of DRAM scaling [3, 32, 33, 34] over the next decade, further increasing the likelihood of DRAM being replaced by these NVRAM memories in future systems.

The confluence of these trends (future large-scale distributed data-centric workloads with I/O intensive behavior, the corresponding innovations in the software stack, the end of scaling for DRAM, and their potential replacement with NVRAM memories) offers an opportunity to rethink traditional system architecture and memory hierarchy design for future workloads.

The rest of the chapter is organized as follows. Section 3.2 motivates and presents

the design of nanostores, a new building block for data-centric systems. A nanostore includes 3D-stacked non-volatile memory with a layer of compute cores and a network interface, and can operate as a system node in a larger distributed system running a data-parallel environment like MapReduce. Section 3.3 presents a new evaluation methodology to reason about these new architectures. Our approach includes a hybrid evaluation model that incorporates high-level application models akin to database query plans or MapReduce simulations in combination with detailed micro-architectural simulations and a data-centric workload taxonomy and a benchmark suite designed to systematically exercise this taxonomy. Section 3.4 presents our evaluation results demonstrating significant benefits even when compared to aggressive future extrapolations of current best systems one to three orders of magnitude performance improvements at 2X to 10X improved energy efficiency. We break down the benefits across the key design aspects, discuss workload-specific trends and key assumptions in leveraging these benefits, and present sensitivity results to technology trends and limits. Section 3.5 concludes the chapter.

3.2 Architecture: Nanostores

3.2.1 Motivation

Data-centric workloads

An important trend in the emergence of data-centric workloads has been the emergence of complex analysis at immense scale (coupled closely with the growth of large-scale internet web services). Traditional data-centric workloads like web serving and online transaction processing are being superseded by workloads like real-time multimedia streaming and conversion, history-based recommendation systems, searches of text, images and even videos, and deep analysis of unstructured data (e.g., Google Squared). From a system architecture point of view, a common characteristic of these

workloads is that they are generally implemented on highly distributed systems and adopt approaches that scale by partitioning data across individual nodes. Their large scale is reflected both in the total amount of data involved in a single task and the number of distributed compute nodes required to process the data. Additionally, these workloads are I/O intensive often with random access patterns to small-sized objects over large data sets. Many of these applications are also operating on larger fractions of data in memory. A recent study reports that, for non-image data, the total amount of DRAM used in Facebook is approximately 75% of the total data size [24]. While this trend partly reflects the little or no locality due to complex linkages between data for the Facebook workload, similar trends can be seen for memcached servers (DRAM distributed data cache) and TPC-H (database benchmark) winners over the past decade. Similarly, search algorithms (e.g., from Google) have evolved to store their search indices entirely in DRAM. These trends motivate us to rethink the balance between memory and disk-based storage in traditional designs. Second, recent data-centric workloads have also been characterized by a lot of commercially deployed innovations in the software stack (e.g., Google BigTable and MapReduce, Amazon Dynamo, Yahoo PNUTS, Microsoft Dryad, Facebook Memcached, LinkedIn Voldemort). Indeed, a recent talk mentions that the software stack behind the very successful Google search engine was re-architected significantly four times in the last seven years, to achieve better performance at increased scale [20]. The growing importance of this class of workloads, their focus on large-scale distributed systems with ever increasing use of memory, and their openness to software-level innovations together offer an opportunity for a corresponding clean-slate architecture design targeted at these workloads.

Technology trends

Concurrently, recent technology trends are providing opportunities for better system designs. Recent microprocessors have favored multicore designs, emphasizing multiple simpler cores for greater throughput. This is well matched with the large-scale distributed parallelism discussed earlier in data-centric workloads. Operating cores at near-threshold voltage has been shown to significantly improve energy efficiency [61]. Similarly, recent advances in networking, particularly around optics, show a strong growth in bandwidth for communication between different compute elements at various levels of the system design. The most important technology changes pertinent to data-centric computing, however, relate to the advances and adoption of non-volatile memory. Flash memories have been increasingly adopted in popular consumer systems (e.g. smartphones) and are starting to gain adoption in the enterprise market (e.g, FusionIO [29]). Emerging non-volatile memories have been demonstrated with superior properties to Flash, most notably, Phase-Change Memory (PCM) and, more recently, Memristors. Table 2.1 summarizes key attributes (density, bandwidth, latency, energy and endurance) of potential storage alternatives in the next decade. These trends suggest that future non-volatile memories can be viable DRAM replacements, achieving competitive speeds at much lower power consumption, and with non-volatility properties similar to disk but without the power overhead. Additionally, several recent studies have identified a slowing of DRAM growth (Section 2.2.1) due to scaling challenges for charge-based memories. The adoption of NVRAM memories as DRAM replacement can potentially be accelerated due to such limitations in scaling DRAM. Two traditional limitations of NVRAM technologies have been around density and endurance, but recent trends suggest that these limitations can be addressed. Increased density can be achieved

within a single-die through multi-level designs, and potentially multiple layers per die [43]. At a single chip level, 3D die stacking using through-silicon vias (TSVs) for inter-die communication can further increase density. Such 3D stacking also has the additional advantage of closely integrating the processor and memory for higher bandwidth and lower power (due to short length low capacitance wires). In terms of endurance, compared to flash, PCM and Memristor offer significantly better functionality (10^7-10^8 writes per cell compared to the 10^5 writes per cell for Flash). Optimizations at the technology, circuit, and systems levels [51, 32, 34, 33] have been shown to further address endurance issues, and more improvements are likely as the technologies mature and gain widespread adoption. (We discuss the lifetime of our proposed designs further in Chapter 4.2.) These trends suggest that technologies like PCM and Memristors, especially when viewed in the context of advances like 3D die stacking, multicores, and improved networking, can induce more fundamental architectural change for data-intensive computing than traditional approaches that use them as solid-state disks or as another intermediate level in the memory hierarchy.

3.2.2 Proposed architecture

High-level design

Combining the opportunity for a clean-slate architectural redesign for data-centric workloads with the potential to use emerging NVRAM memories in more disruptive ways, we propose a new system architecture and memory system design that co-locates power-efficient compute cores with non-volatile storage, eliminating many intervening levels of the memory hierarchy. All data is stored in a single NVRAM data-store layer that replaces traditional disk and DRAM layers (disk is relegated to archival backup). Figure 3.1 illustrates the proposed solution consisting of a distributed system comprised of a large number of small homogeneous building blocks.

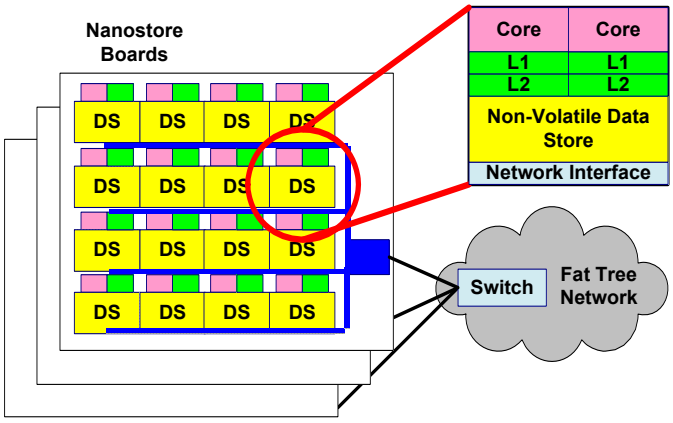


Figure 3.1: Nanostore system architecture

Our individual building block is a single chip and we refer to it as nanostore. A nanostore consists of multiple 3D-stacked layers of dense silicon non-volatile memories (e.g., PCM or Memristors) with a top layer of power-efficient compute cores. TSVs are used to provide wide, low-energy data paths between the processors and data stores. Power and thermal issues are important concerns with 3D-stacking. This limits the amount of compute that can be included in a nanostore. In this work, we make 3D-packaging assumptions similar to the PicoServer project [40] and focus on low-power simpler cores for the nanostore. Our assumption of simpler cores also means that we can correspondingly provision the bandwidth to save power. Each nanostore can act as a full-fledged system with a network interface. The individual nanostores are networked via an on-board connector to form a large-scale distributed system or cluster akin to the existing solutions for data-centric workloads discussed in Section 3.3.2. In terms of physical organization, multiple nanostore chips are organized into small daughter boards that, in turn, plug into traditional blade server boards.

System design choices

While the above description summarizes the high-level organization, there is a wide range of possible implementations. There are a number of design choices in terms of the provisioning, organization, and balance of the compute, storage, and network per nanostore as well as the sharing model across the individual nodes and the topology of the network (including potential differences between the on-chip, on-board, and cross-cluster networks). The design choices are constrained by technology- and circuits-level parameters such as the size of the die and the yield, or the number of 3D-stacked or intra-die layers, as well as packaging constraints such as the power/thermal budget per chip or board.

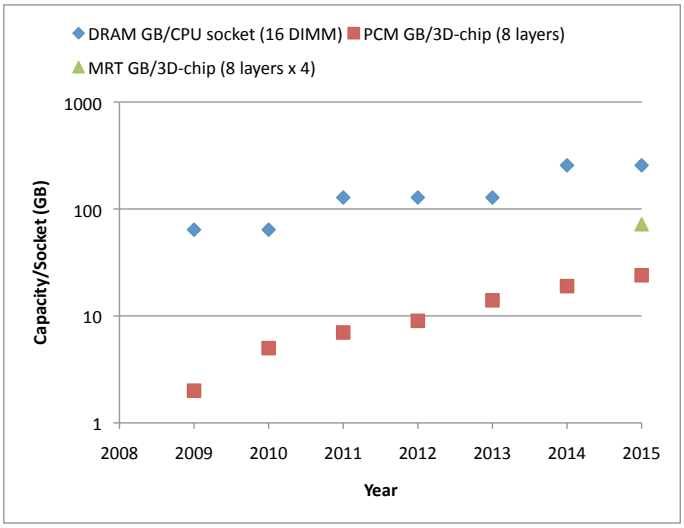


Figure 3.2: Per-node capacities derived from ITRS roadmap 2009 [3]. Note that we use design points at the 2015 timeframe in our models. DRAM capacity represents 16 off-chip DIMMs consisting of 16 DRAM chips each. The 3D-stacked PCM and MRT consist of 8 stacked die, while the MRT also has 4 internal layers on each die.

Figure 3.2 presents the projected device capacities used for Nanostore and baseline evaluations. The baselines, similar to conventional designs, have a die area of $250mm^2$ and a maximum power (TDP) of 80 Watts. We assume a nanostore

die size of $100mm^2$, similar to the cost-effective design point for memories [3]. For a Memristor-based design circa 2015 [47], assuming 8 layers of 3D stacking and 4 intra-die layers [43], the on-chip datastore capacity is approximately 75 GB per socket. An area-equivalent 8-layer PCM-based design would have 25GB per socket, as shown in the figure. These capacities were derived by taking the cell size and array area efficiency as predicted by ITRS, to determine capacity per layer. The Memristor projections employed a cell area factor of $10F^2$ and the same feature size as NAND Flash memory. For in-memory DRAM based baseline designs, we assume that the per-socket DRAM capacity is 128 GB, unless stated otherwise.

For the number of supported cores, we estimated the die area requirement of a single core by performing automatic place-and-route (APR) of a simple 32-bit embedded processor using 45nm technology, with an area of 45,000 μm^2. Scaled to 22nm [2], this is approximately

$$45,000\mu m^2 / \left(\frac{45^2}{22^2}\right) / 10^6(\mu m^2 \text{ per } mm^2) \approx 0.012mm^2 \text{scaled to 22nm} \tag{3.1}$$

For the caches, scaling a 0.145 μm^2 bitcell from 32nm technology [104] and allowing 25% overhead for peripheral circuits, the area of a 1 MB per-core cache is

$$0.145\mu m^2 / \left(\frac{32^2}{22^2}\right) = 0.0685\mu m^2 \text{scaled to 22nm} \tag{3.2}$$

$$\begin{aligned} 0.0685(\mu m^2) \quad &* \quad (8 * 1024 * 1024)(\text{bits per cache}) \\ &* \quad 1.25(\text{overheads})/10^6(\mu m^2 \text{ per } mm^2) \approx 0.76mm^2 \end{aligned}$$

[2] Dave Fick is acknowledged for assistance and validation of these area calculations

Therefore, with a total area of $0.77mm^2$ per core, each $100mm^2$ die can accommodate up to $100/0.77 = 129$ cores.

The read and write latencies as well as access energy are given in Table 2.1. The objective is to include data store capacity, latency, and energy in the overall model to understand the sensitivity of our results to alternate NVRAM instantiations in the future. The cores in the compute layer are based on low-voltage power-efficient microarchitectures with simple SRAM cache hierarchies. Different organizations are possible for the compute layer in the number of cores (1 to 128), clock frequency (100MHz to 2GHz), issue width and pipeline depth (2-way simple to 4-way deep), and L2 cache size (512KB or 1MB per core) and we study this design space. To ensure realistic designs, we limit the power density at the socket (32 Watt/cm2). Later, we vary this power constraint to see the effect on the best designs chosen by our optimizer. For our projected timeframe, we expect 3D stacking to provide significant bandwidth (we model up to 32GB/s as per the PicoServer design [40]) between the processor and stacked memory, and 80Gbps (2x 40Gbps NICs) networking bandwidth per socket (in a traditional architecture) but we vary these as design space parameters too. Since PCM requires 150ns to write a bit, a certain number of physical banks are required to achieve this (worst-case) sustained write throughput. Assuming a 1024-bit TSV bus (as per Picoserver) we would need to interleave accesses over 41 banks to reach 32 GB/s. This is achievable (reference [77] used 64 3D-stacked DRAM banks), and read accesses would require fewer banks due to their lower access latency. We assume a large-scale distributed shared-nothing system abstraction. This is well-matched with current data-centric workloads. Each nanostore can be viewed as a complete independent system executing the software stack needed to implement a data parallel execution environment like MapReduce.

Discussion

The two most important aspects of nanostores are (1) the co-location of power-efficient computing with a single-level data store, and (2) the support for large-scale distributed design. Together, these enable several benefits. The single-level data store enables improved performance due to faster data access (in latency and bandwidth). Energy efficiency is also improved from the flattening of the memory hierarchy (fewer data transfers) and the increased energy efficiency of NVRAM over disk and DRAM. The large-scale distributed design allows for higher performance from increased parallelism and higher overall data/network bandwidth. This design also improves energy efficiency by partitioning the system into smaller elements that can leverage more power-efficient components (e.g., simpler cores). At the same time, there are potential disadvantages. Given the smaller capacities of per-socket storage, the number of individual elements in the system increases dramatically. This can potentially increase the stress on the networking subsystem around bandwidth contention (particularly for all-to-all communication), topological complexity and port count, and power. Software scalability can also be an issue. While large-scale deployments of data-centric workloads have been demonstrated, latency requirements (e.g., 500ms response time for a search request) will still have to be carefully considered in the sizing of the system. Finally, chip-level thermal constraints can limit the compute per nanostore; this could lead to compute bottlenecks (e.g., for sophisticated data processing like collaborative filtering). The rest of this chapter analyses the performance and power benefits relative to aggressively scaled and optimized versions of conventional architectures, in the 2015 timeframe. We identify the varied balance points between storage capacity, compute power, memory and network bandwidths for each system type. As well as providing detailed power and perfor-

mance breakdowns and a novel hybrid modelling methodology, we look at sensitivity to important thermal and network constraints.

3.3 Evaluation Methodology

3.3.1 Challenges

Evaluating our proposed architecture and understanding the design space poses several challenges. Our focus on the combination of multiple future technologies for emerging workloads poses several challenges in the choice of benchmarks, technology parameters and baseline systems appropriate to this longer timeframe.

Furthermore, given the speculative nature of such an exercise, we need to relate these assumptions to a systematic understanding of the trends as well as understand the variance of the results to alternate constraints (affecting both the specific parameters and the combinations in which they are used). We also back up our results via small-scale experiments, using a cluster of networked machines with up to 64 processors to validate our performance models.

To evaluate our proposed design and its tradeoffs, we need to model large-scale clusters running distributed workloads operating on large volumes of data. We also need to examine tradeoffs at the full system level including computing, networking, memory, and storage layers. Conventional architecture simulators not only lack the ability to cope with this level of system scale, but also the modeling means for storage and network subsystems at a distributed systems level. There is also a combinatorial explosion in the design space from various assumptions at the fine-grained and coarse-grained architectural levels as well as the choice of technology and workload parameters. An appropriate evaluation methodology is required to systematically reason about this large design space. To address these challenges, we next discuss a new set of benchmarks that provides systematic coverage of data-centric workloads

and develop a new evaluation framework that uses hybrid performance models to reason about future new architectures.

3.3.2 Proposed benchmarks

Response Time	Real-time	Real-time or interactive responses required
	Background	Response time is not critical for user needs
Access Pattern	Random	Unpredictable access to regions of data store
	Sequential	Sequential access of data chunks
	Permutation	Data is re-distributed across the system
Working Set	All	The entire dataset is accessed
	Partial	Only a subset of data is accessed
Data Type	Structured	Metadata/schema/type are used for data records
	Unstructured	No explicit data structure, e.g., text/binary files
	Rich media	Audio/video and image data with inherent structures and specific processing algorithms
Read vs. Write	Read heavy	Data reads are significant for processing
	Write heavy	Data writes are significant for processing
Processing Complexity	High	Complex processing of data is required per data item. Examples: video trans-coding, classification, prediction
	Low	Dominated by data access with low compute ratio. Examples: sort, upload, download, filtering, and aggregation.

Figure 3.3: A data-centric workload taxonomy

The space of data-centric workloads is vast, fast-evolving, and characterized by rich diversity across multiple dimensions. To study a subset of workloads that provide sufficient coverage and representativeness, we systematically create a taxonomy of data-centric workloads to characterize the key dimensions of diversity and pick a subset of workloads that exercises all these dimensions [12]. Figure 3.3 illustrates the taxonomy of data-centric workloads that we developed based on examination of a wide class of emerging applications. The key dimensions are as follows. Response time may be real-time with a time constraint or background tasks with no deadline. The access pattern is either dominated by random, sequential or permutation accesses, where data are read then re-written elsewhere. The working set could involve

	Access: Random	Access: Sequential	Access: Permute	Time: Real-time	Time: Background	Compute: High	Compute: Low	Write-Heavy	Read-Heavy	Working Set: All	Working Set: Partial	Data: Structured	Data: Unstructured	Data: Rich media
Sort			X		X		X	X		X		X		
Search (web)	X			X		X			X	X		X		
Search (image)	X			X		X			X	X			X	X
Search Indexer	X				X	X			X	X		X	X	X
Recommender	X				X	X			X	X		X		
De-duplication		X			X		X		X	X		X	X	
Transaction	X	X	X	X			X	X			X	X		
Decision Support	X	X	X		X	X			X		X	X		
Video Sharing	X	X		X	X	X			X		X	X	X	X
Data Mining			X		X		X		X	X		X		

Figure 3.4: Workload mapping

all stored data, such as distributed sorting, or a smaller subset. The data type may be structured (database-like), unstructured (e.g. raw text) or rich media (requiring complex processing to encode or decode). Access type can be read or write dominated. Finally, processing complexity (low, medium or high) is a qualitative measure of the number of transformations required per unit of data being processed. Figure 3.3 further explains the attributes of each dimension. Figure 3.4 shows an example of mapping some popularly-referenced workloads to the taxonomy and picking a small subset with full coverage (shaded rows). Our chosen workloads provide representative coverage of different dimensions of data-centric workloads, capture emerging trends towards data analysis and media processing, and support publicly available implementations that we can use for simulation. We describe these workloads next.

- Sort: The sort benchmark models a two-pass distributed sort of a 1 petabyte dataset. The workload is both read- and write-heavy, and stresses the balance between compute/storage/networking subsystems. The algorithm we study is massively data-parallel and has two phases: (i) a shuffle phase where each server first reads keys from its local storage and sends them over the network to their

individual target servers while simultaneously receiving keys from other servers. As the memory fills with incoming keys, the server sorts the buffered keys and writes the results to its local storage. (ii) a local merge phase after the shuffle, where each nanostore reads partially sorted keys from many local files, performs a merge sort then writes the final sorted results to local storage. We use the nsort benchmark to model the local sort phases of this workload.

- Checksum in data deduplication: This benchmark cksum implements checksum calculation in data deduplication, resulting in mostly read-only, sequential access with low processing complexity. We model a massively parallel implementation of cksum over a 1 Petabyte dataset. Each server scans its local files and generates block or file signatures using the SHA-1 hash function. We use sha1sum for our simulations.

- Video transcoding: This benchmark models popular video encoding/transcoding web services that exploit cloud infrastructure for batch processing. The algorithm reads the video input files, transcodes, and then stores the output in a new format locally. For our simulations, we use the ffmpeg benchmark over a 1 petabyte dataset. Our sample video is in FLV format at 320x240 resolution and is transcoded into JPEG snapshots. The average size of the video is 10MB.

- Recommender: This benchmark recom represents parallel machine learning algorithms with high processing complexity and regular communication patterns. Our workload models the Netflix video recommendation benchmark [105] over a 5 Terabyte dataset using parallel matrix factorization. The algorithm iteratively refines two matrices so their product can best summarize the ratings matrix. The large matrices are distributed across the servers main memory. Each iteration has four phases, two of them are compute-heavy matrix operations while

in the other interleaving phases. The servers update their local copies of input matrices using personalized broadcast messages. The implementation requires large main memory to host the matrices and compute/communication balance. For our simulations, we use a MATLAB compiled winner of the Netflix challenge from 2008.

- Search: The search benchmark models text search across a 128 Terabyte data set, using an in-memory index to achieve sub-second response times. The workload is read-only and stresses random access patterns. Similar to popularly-used in-memory index-based text searches (e.g., Google), the entire index is partitioned and stored in-memory in a large-scale distributed cluster. Each server searches its local index in the map phase, and sends the top-matching document list to the front-end server in the reduce phase. In addition to search query throughput, this benchmark models a quality of service (QoS) requirement that average query latency should be less than 500ms. For our simulations, we use the lucene benchmark for the map phase.

3.3.3 Proposed evaluation methodology

Two-level simulation with design optimizer

The evaluation methodology that we use is summarized in Figure 3.5. Specifically, our approach has three main components: (1) a high-level distributed system model, (2) a lower-level microarchitecture-based model, and (3) a design space optimizer. Our high-level distributed system simulation captures the applications system-level behavior and allows exploration of compute-network-datastore balance. Our implementation is inspired by approaches currently used for MapReduce/Hadoop simulation (e.g., [106]) and query optimizers in databases (e.g., [107]). Similar to these approaches, our high-level model uses an application-level execution template that

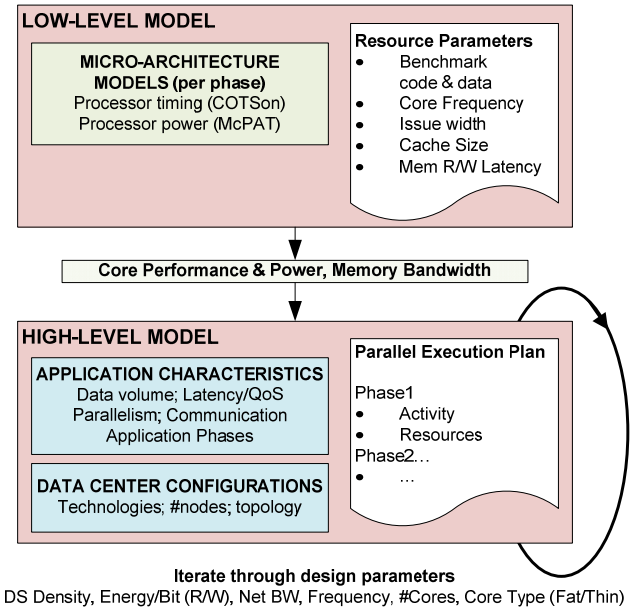


Figure 3.5: Two-level simulation with design optimizer

breaks down execution time into key compute, communicate, and I/O subsystem phases coupled with models for performance and power for each subsystem. We use high-level models for the network and I/O subsystems, but use the input from a detailed microarchitectural simulator to model the compute subsystem (nanostore). This lower-level microarchitecture-based model captures the applications instruction-level behavior and allows exploration of architecture choices like issue width and cache size. The CPU data processing throughput and memory bandwidth results from this model feed into the higher-level simulation. A final design space optimizer iterates through all of the compute, storage, and network parameters while filtering out invalid configurations (that exceed thermal limits, for example). Sorting the remaining valid configurations by the objective function (maximum performance, energy efficiency or energy-delay product (EDP)) gives the optimal balanced design. Figures 3.6 to 3.13 present the execution templates and the high-level models for each of the benchmarks. Apart from the in-memory workloads (Recommender and

Search) which already assume unified data storage and main memory in DRAM, there are two execution plans for each workload. The first represents the steps involved for a conventional system with separate data store and main memory. Loads and saves of persistent data to/from main memory are necessary because the entire data set is too large to fit in memory. Although these same transfers could be used when the data store (DS) and main memory are physically combined, the file loads/-stores are redundant. They are eliminated from the "unified" execution plans, which represent software optimized for Nanostore-like designs.

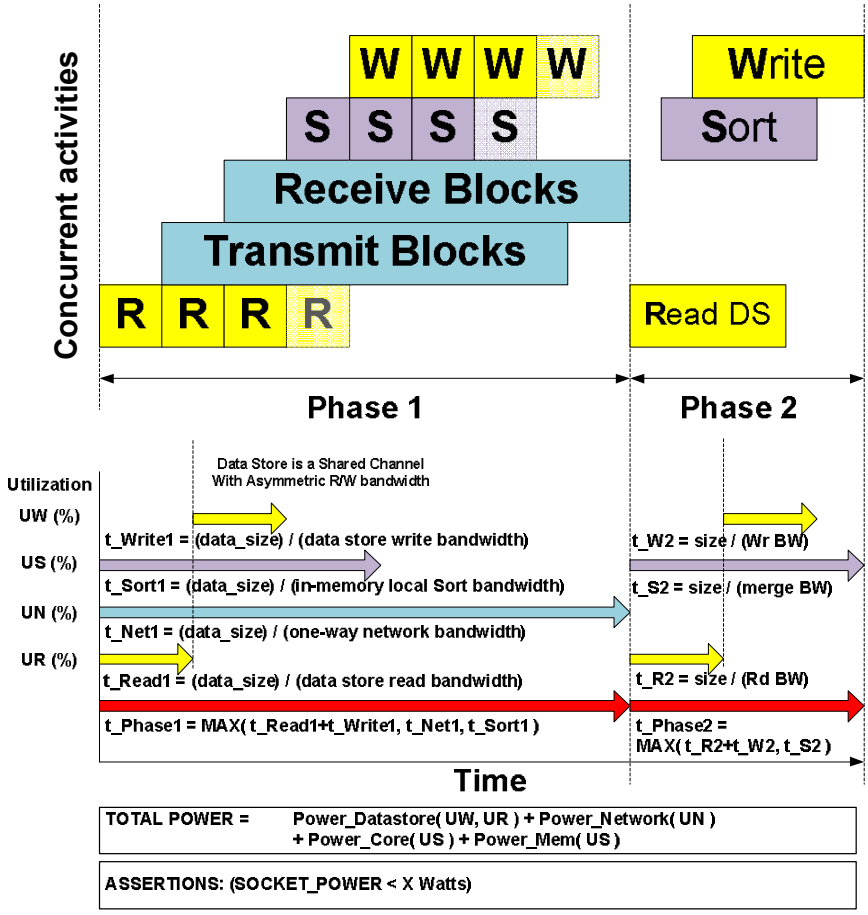


Figure 3.6: Sort: Separate Data Store and Main Memory

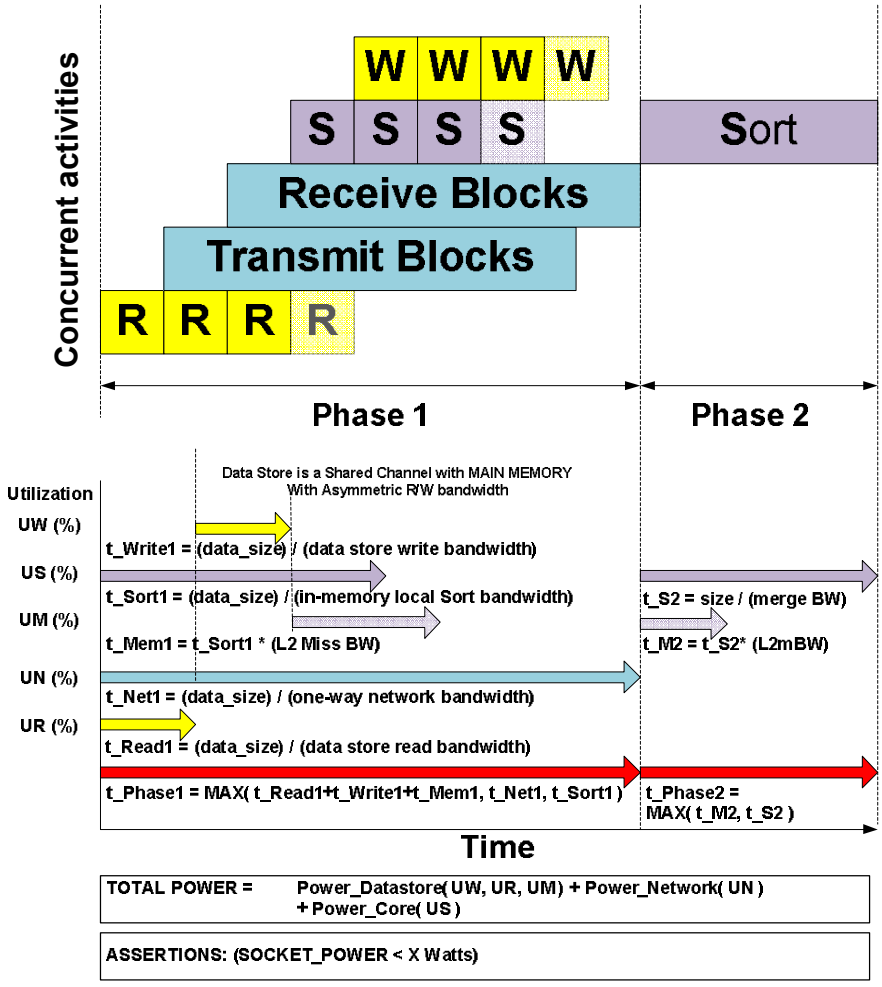


Figure 3.7: Sort: Unified Data Store and Main Memory

Here we describe the execution plans and how they model the high-level behaviour of the data center workloads (see Section 3.3.2). The *sort* benchmark (Figures 3.6 and 3.7) is shown with time on the x-axis, illustrating the two basic phases (network shuffle and local merge sort). There are two separate execution plans, one for the non-unified baseline memory/storage system (Figure 3.6), and another for the combined single-level memory/data store (Figure 3.7). During the first phase, blocks of unsorted data are read from the data store, bucketed into ranges of keys (one for each destination node), then transmitted to those destinations. Concurrently, incoming

blocks in the range allocated to the current node are received and placed in the data store. The received blocks are then fully sorted, as they are received. This makes good use of any idle CPU time during the communication process. During the second phase, there is only local computation and no network transfers occur. The sorted blocks are read from the data store, merge-sorted on each node, then written back to the data store, resulting in totally ordered data across all of the data center nodes. There are some small differences in the unified execution plan. First, the memory bandwidth generated during data sorting is added to the file transfer bandwidths caused by transferring blocks on the network. This is because the 3D-stacked memory and data store channel is now shared. Secondly, during the local merge phase, reading and writing data to/from main memory are not required, as the cores read data directly from the Nanostore, and the final in-memory sorted data represents the sorted file. For combined main memory and data stores, it is envisaged that the system software will provide a mechanism to nominate regions of memory as permanent data files for later reference by other processes.

Underneath each execution plan block diagram are the high-level equations used to calculate execution time. For Figure 3.7, the overall memory bandwidth utilization is determined by three potential bottlenecks. The first is the data store (yellow), whose channel is shared with main memory in the unified case. To find the utilization (and thus minimum execution time when bandwidth-constrained) we add together the minimum read, write and memory transfer times given the full network bandwidth ($t\_Read1 + t\_Write1 + t\_Mem1$). Second, we have limited block sorting throughput depending on the core performance (obtained during micro-architecture simulation in the low-level model). Finally, the minimum network transfer time is calculated as the amount of outgoing sorted data divided by the outgoing bandwidth (we receive an equivalent amount from the other nodes, in parallel). The maximum of these

three times provides the minimum execution time for phase 1;

$$t\_Phase1 = MAX((t\_Read1 + t\_Write1 + t\_Mem1), t\_Net1, t\_Sort1) \tag{3.3}$$

During the second phase, we only need to deal with the in-place merge-sort of the data blocks. Therefore there are no network traffic or file transfers, only the compute throughput and network bandwidth limit. Therefore the execution time of the second phase can be approximated as;

$$t\_Phase2 = MAX(t\_Sort2, t\_Mem2) \tag{3.4}$$

This first-order model for sharing the memory channel assumes that if more memory bandwidth is required for computation than is available, performance decreases proportionally to the amount that demand exceeds bandwidth supply. In the case of phase 1, this can be interpreted as the sorting task being paused until a data block transfer has finished, then resuming. Although this ignores micro-architectural effects such as cache pollution due to file access, careful use of DMA transfers to/from the network interface and non-cacheable memory regions can help to mitigate these effects.

Similar reasoning is applied to the other benchmarks regarding the separate and unified execution plans. Note that the *search* and *recommender* baselines store their data in DRAM, therefore there is only a unified execution plan. All of the benchmarks are subject to a per-socket thermal constraint, which is nominally 32 Watts per $100mm^2$, but is varied for sensitivity analysis. Search has an additional constraint in that each search must complete within a response time of 0.6 seconds. System configurations which violate any of the constraints are removed from the list of

candidates by the optimizer, before finding the remaining valid design with the best performance metric.

The utilization factors of each resource (active time / total time) are fed into the power models to scale the active power consumption depending on utilization.

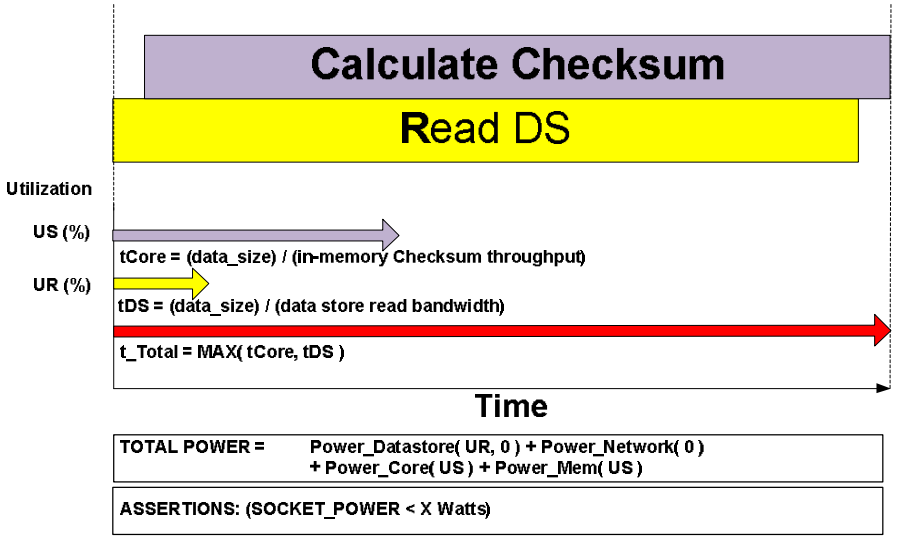


Figure 3.8: cksum: Separate Data Store and Main Memory

Our evaluation methodology is unique in addressing the challenges discussed earlier in evaluating a new architectural model, with forward-looking technology and workload assumptions, on large-scale distributed systems. It also addresses all the compute, I/O, and network components of the system, and provides a powerful way to systematically explore the rich design space at practical simulation times. However, a few caveats need to be noted. Our high-level application execution models assume that computation and network communication can overlap and are purely bandwidth based (i.e. no queuing models are used). These assumptions are acceptable for the distinct phases and coarse-grained communication behavior of the representative benchmarks we consider in this study, but care needs to be exercised in extrapolating the model to other workload classes. Data is also assumed to be

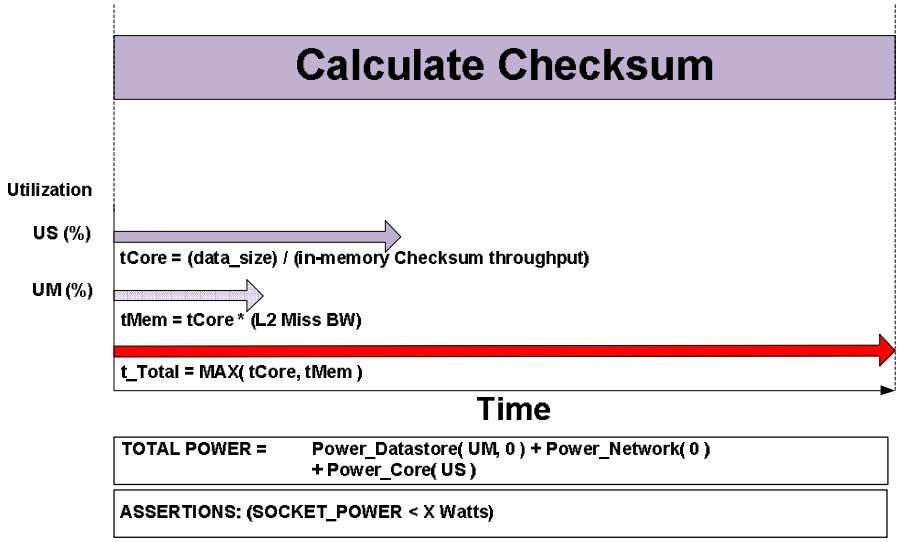


Figure 3.9: cksum: Unified Data Store and Main Memory

distributed uniformly, and load-balancing effects are ignored. For the purposes of our study, these assumptions allow an adequate comparison of hardware architectures under reasonable software conditions. However, issues around optimizing the operating systems and middleware for load-balancing and scheduling need to be addressed in future work. Finally, in the absence of actual prototypes (which is difficult given that many of the technologies we study are still lab samples), it is hard to validate the model for futuristic design configurations. However, for simple near-term design alternatives, we have tried to make sure that our results match prior published trends. For example, we studied the performance of a simple MPI-based sort implementation on small cluster sizes (4 to 64 cores) and found the results tracked the model well (see Section 3.4.7).

Performance and power models

The data store and network subsystem performance models calculate the execution time for storage access and communication activities based on the provisioned

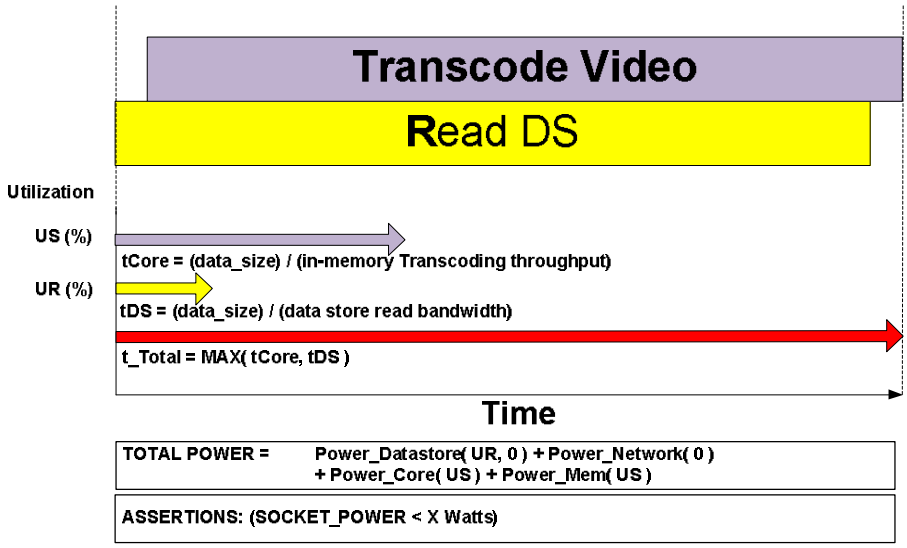


Figure 3.10: video: Separate Data Store and Main Memory

bandwidths of the subsystems and the amount of data transferred to and from the subsystem for the workload execution. For the data store bandwidth, we model the combined bandwidth needs of both file and memory accesses in our proposed designs. For the compute subsystem, we use the publicly-available benchmarks discussed earlier for each of our workloads and proceed in two steps. We execute the benchmark on an existing Xeon-based server with the baseline processor and DRAM. The server is configured with minimal storage and network overhead (e.g., in-memory search and processing cached data files). This experiment allows us to measure a "compute data processing throughput" baseline. For different specific processor configurations, we run the benchmark through a detailed publicly-available microarchitectural simulator (COTSon [108]) and use the simulated IPC values to normalize the data throughput rate. For our results, we focus mainly on average power consumption. (We also study peak power consumption, but use it primarily to verify compliance with cooling constraints.) We use the execution time models to compute the utilization for the processors, memory, data store and network ports. Active power is assumed

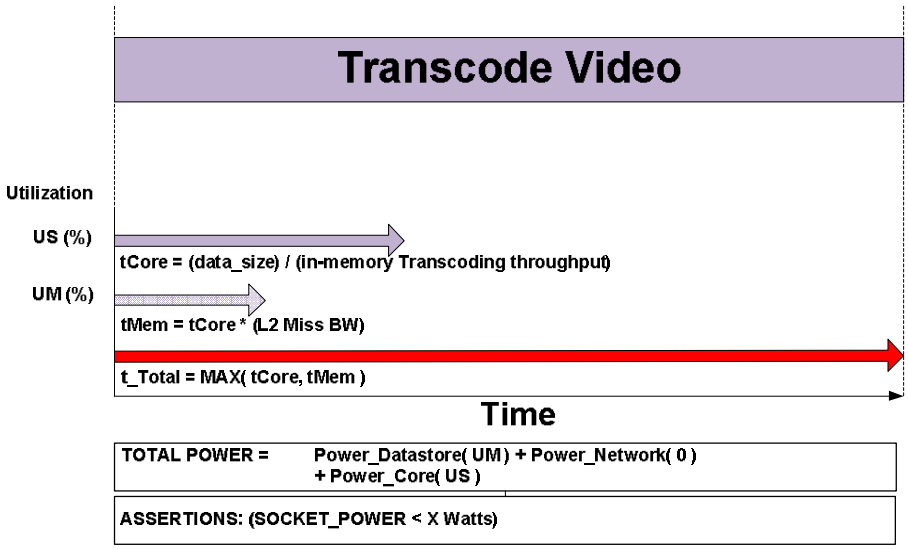


Figure 3.11: video: Unified Data Store and Main Memory

to scale linearly up to peak power as utilization increases. Several components also have a non-zero idle power (such as DRAM refresh or CPU leakage power). During phases where the CPU cores are active, we use the memory bandwidths (read and write) simulated in COTSon to calculate the memory utilization.

Network power: Since network packets traverse multiple switches in the datacenter, we scale our NIC-level power model with a network layer multiplier (the effective levels in a fat tree switch network) to calculate the total network power. We calculate the number of network layers required based on a rearrangeably non-blocking [36] fat-tree network [37]. Rearrangeably means that given any pair of nodes that wish to communicate, it is possible to reconfigure the already-established connections to satisfy the new communication. The most basic network consists of two physical layers of switches, with nanostores as the leaves acting as senders and receivers. Using the equations in [37] we derived Table 3.1 which relates the number of network nodes to the number of network layers required. By having a non-blocking network, we assume in the models that any pair of communicating nodes can make use of the

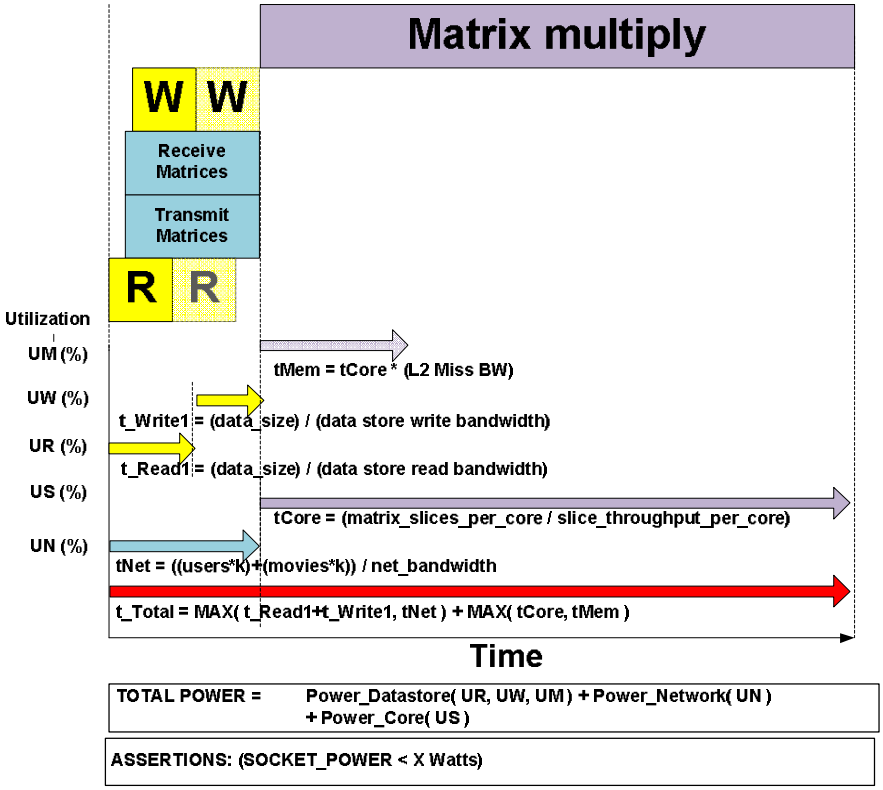


Figure 3.12: Recommender

full bandwidth available from their network interface. In the high-level model, we impose a limit on the per-node network bandwidth between the baseline (conventional) design and the Nanostore design. This has the effect of limiting the scale-out of the number of network nodes, a side-effect of using smaller data stores.

CPU power: The CPU peak power is determined as a function of issue width, frequency and cache size using the McPAT modelling tool [109]. CPU configurations

ports/switch	64	64	64	64
height (including leaves)	2	3	4	5
maximum nodes	64	2048	65536	2097152
switches	1	96	5120	229376

Table 3.1: Network layer scaling

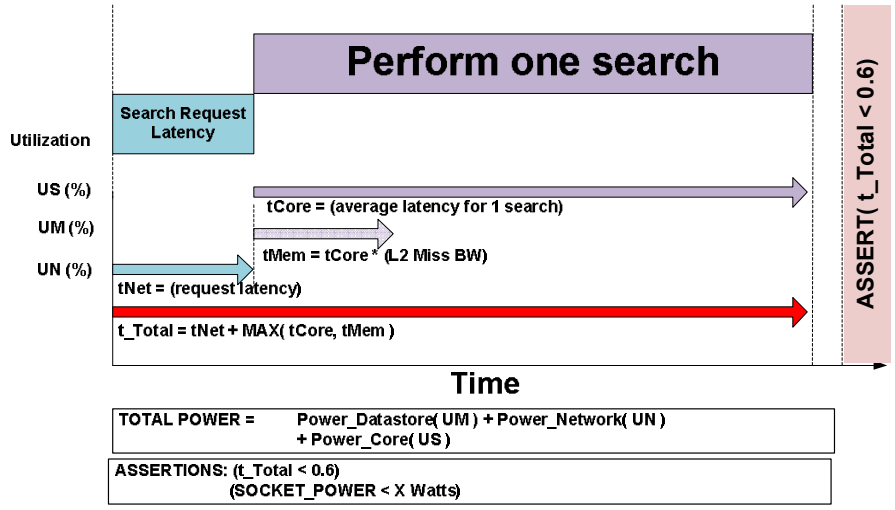


Figure 3.13: Search

at each frequency also factor in the power benefits from voltage and frequency scaling using the models specified in [109]. CPU idle power is scaled based on the number of cores and caches on each parameterized nanostore chip.

3.3.4 Choice of parameters and baselines

As discussed earlier, we examined ITRS roadmaps and prior publications and talked to industry sources to determine forward-looking parameters for our studies, but in most cases also study sensitivity to a range of values for each parameter. Also, we have generally tried to choose parameters that favored more conservative projections for the benefits of our design.

Figure 3.14 summarizes our system parameter assumptions. The different compute configurations (varying number of thin/fat cores with different cache sizes), nanostore memory configurations (memristor/PCM), and data/network bandwidth assumptions are listed. For DRAM, we ignore any potential end-of-life scaling limitations at our projected time frame and extrapolate historical scaling trends in capacity and bandwidth. We assume configurations of 16GB per DRAM DIMM module and

Processor	Baseline	Nanostore
Core count	32	1-128
Frequency (GHz)	2	0.1-2.0
Issue width	4	2, 4
Per-core L1 cache	64K+64K	64K+64K
Per-core L2 cache	1M	512K, 1M
Peak Power/Core (W)	1.83	(model)
Idle Power/Core (W)	0.04	(model)

Network	Baseline	Nanostore
Peak BW/Port (Gbit/s)	40	40
Peak Power/Port (W)	10	10
Idle Power/Port (W)	2	2

Main Memory	Baseline	Nanostore
Peak BW/Unit (GB/s)	25.6	32
Capacity/Unit (GB)	16	25, 75
Peak Power/Unit (W)	4.6	5.0, 0.6
Idle Power/Unit (W)	2	0.0, 0.0

Hard Disk/SSD	HDD	SSD
Peak BW/Drive (GB/s)	0.5	4.5
Capacity/Drive (TB)	6	1.2
Peak Power/Drive (W)	10	10
Idle Power/Drive (W)	8	1

Figure 3.14: System parameters

25.6 GB/s bandwidth for the DRAM channel, with each DIMM consuming 2.6W of active power and 2W at idle [110]. For persistent storage, we assume HDDs at 6TB capacity, 500 MB/s bandwidth, and total power consumption varying between 8W to 10W from idle to peak. We also study SSDs at lower capacity per drive (1.2TB), but higher bandwidth (ranging up to 4.5GB/s) with inherently better energy efficiency (see Table 2.1). For the networking subsystem, we assume peak and idle power for a 40 Gbps Ethernet NIC of 10W and 2W respectively, closely matching the assumptions in [35]. The changes to the network topology corresponding to the increase in the number of leaf nodes with the relatively small capacity networked nanostores is modelled as an increase in the number of layers of switches, as described.

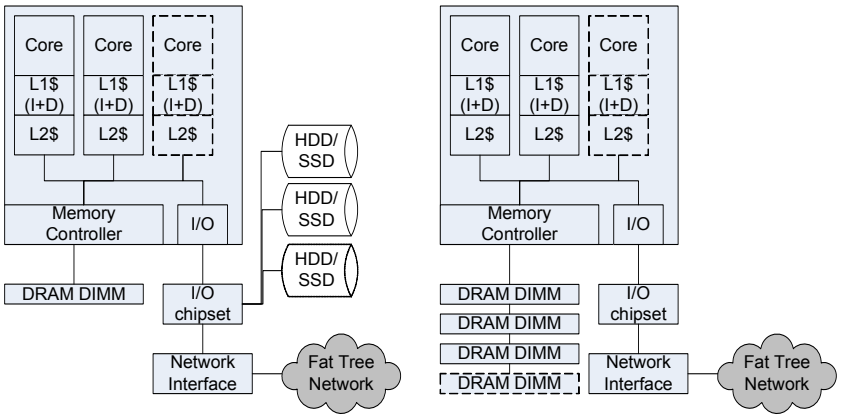


Figure 3.15: Baseline architectures

Figure 3.15 summarizes the baseline system architectures we study for each workload. To provide a fair comparison to the baseline, we recognize that different sweet spot design configurations have evolved for different workloads (in terms of emphasis on compute, storage, and networking, and organization), and correspondingly choose a baseline known to be best-suited for each workload. Specifically, sort, cksum and video benchmarks keep their data on disks and are each allocated a single DRAM DIMM. Search and Netflix are in-memory workloads so do not have a hard disk component, but have multiple DIMMs. To ensure the most appropriate balanced baseline, other than the choice of an enterprise-class processor, other design parameters in the baseline are determined through the iterative design space search of our simulator.

3.4 Evaluation Results

3.4.1 Baseline benefits

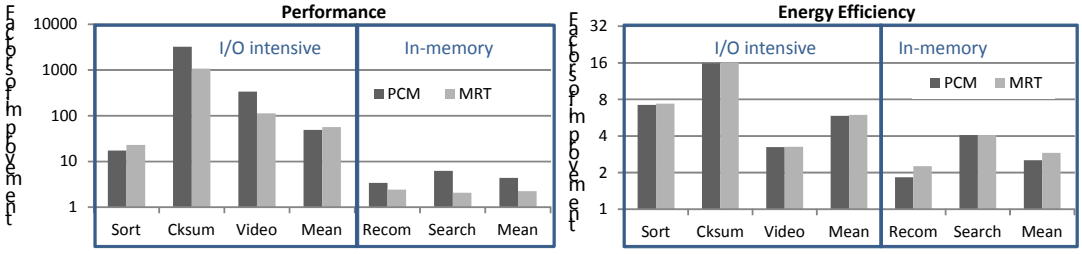


Figure 3.16: Performance and energy efficiency improvements relative to 2015 baselines (EDP-optimized).

Figure 3.16 presents the improvements in performance and energy efficiency from our nanostore designs relative to the baselines discussed above. Results for both the PCM- and Memristor-based nanostore designs are presented for the five benchmarks. For these results, consistent with the data-centric focus of this work, we assume that the baseline and nanostore systems both operate on the same dataset size and keep

the amount of permanent persistent storage the same. The data size and device capacity together determine the number of sockets in the nanostore designs; all other parameters are based on Section 3.3.4. Also, as discussed earlier, each point represents the results of a design space search by the optimizer across a range of configuration parameters, for both the baseline and nanostore designs, but with caps on the thermal density and aggregate network bandwidth as discussed in Sections 3.2.2 and 3.3.3. This ensures that the individual designs are locally balanced for their objective function while meeting the design constraints. We focus on energy-delay-product as the primary objective function (since we want to optimize both energy efficiency and performance) but discuss other objective functions briefly in Section 3.4.3.

The results in Figure 3.16 show that for all our benchmarks, the nanostore solutions achieve higher performance at better energy efficiency. For the three I/O intensive benchmarks sort, cksum, video the nanostore designs achieve one to three orders of magnitude higher performance improvement with 3X-16X improved energy efficiency. For the in-memory benchmarks with DRAM baselines recom, search nanostores achieve 2X-6X better performance with 2X-4X improved energy efficiency. Comparing the two different NVRAM technologies we consider, the PCM-based nanostores generally outperform the memristor-based designs, but at reduced energy efficiencies. However, it should be noted that our constant dataset size constraint presents the memristor-based design at a disadvantage. While, as the results indicate, a pre-packaged 1 petabyte PCM-based design would have more performance (and cores) than a pre-packaged 1 petabyte memristor-based design, the PCM-based design also has more individual nanostore sockets (and correspondingly more volume). If an alternate comparison considered performance for the same number of sockets (or equivalently the same amount of silicon), the Memristor-based design

would have higher performance than PCM (by a factor corresponding to the Node multiplier discussed later).

3.4.2 Analysis of performance benefits

Bench	Scheme	1/EDP	Perf	EE	Configuration	ScaleX (Node/OPS/DS/Net)			
Sort	Base	1.0	1.0	1.0	Core=28/Freq=2.0/Issue=4/L2=1024k/DS=6000GB/nDS=2/DS_BW=0.5GBs/Net_BW=0.500GBs	1	1	1	1
	SSD	47.2	15.3	3.1	Core=128/Freq=2.0/Issue=2/L2=1024k/DS=1280GB/nDS=2/DS_BW=4.5GBs/Net_BW=1.250GBs	5	11	42	12
	DRAM	20.9	20.5	1.0	Core=128/Freq=0.5/Issue=2/L2=512k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.125GBs	47	27	1198	12
	PCM	124.8	17.3	7.2	Core=22/Freq=0.1/Issue=2/L2=512k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.013GBs	479	9	15329	12
	MRT	170.8	23.1	7.4	Core=88/Freq=0.1/Issue=2/L2=512k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.050GBs	160	13	5110	16
Cksum	Base	1.0	1.0	1.0	Core=40/Freq=2.0/Issue=4/L2=1024k/DS=6000GB/nDS=14/DS_BW=0.5GBs/Net_BW=0.001GBs	1	1	1	1
	SSD	316	42	7.5	Core=104/Freq=0.5/Issue=4/L2=512k/DS=1280GB/nDS=2/DS_BW=4.5GBs/Net_BW=0.001GBs	33	21	42	1
	DRAM	3695	583	6.3	Core=128/Freq=1.0/Issue=2/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	326	263	1198	1
	PCM	51353	3235	15.9	Core=128/Freq=0.5/Issue=2/L2=512k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	3333	1344	15329	1
	MRT	17432	1078	16.2	Core=128/Freq=0.5/Issue=2/L2=512k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	1111	448	5110	1
Video	Base	1.0	1.0	1.0	Core=40/Freq=2.0/Issue=4/L2=1024k/DS=6000GB/nDS=2/DS_BW=0.5GBs/Net_BW=0.001GBs	1	1	1	1
	SSD	15.3	7.8	2.0	Core=88/Freq=2.0/Issue=2/L2=1024k/DS=1280GB/nDS=2/DS_BW=4.5GBs/Net_BW=0.001GBs	5	5	42	1
	DRAM	113.1	78.1	1.4	Core=88/Freq=2.0/Issue=2/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	47	52	1198	1
	PCM	1093	337	3.2	Core=128/Freq=0.5/Issue=2/L2=1024k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	479	192	15329	1
	MRT	368	112	3.3	Core=128/Freq=0.5/Issue=2/L2=1024k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	160	64	5110	1
Recom	Base	1.0	1.0	1.0	Core=56/Freq=2.0/Issue=4/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.500GBs	1	1	1	1
	DRAM	3.0	1.7	1.7	Core=112/Freq=2.0/Issue=2/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=1.250GBs	1.0	1.0	1.0	2.5
	PCM	6.2	3.4	1.8	Core=128/Freq=2.0/Issue=2/L2=1024k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.500GBs	10.2	11.7	12.8	10.2
	MRT	5.5	2.4	2.3	Core=120/Freq=0.5/Issue=4/L2=1024k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=1.250GBs	3.4	1.8	4.3	8.5
Search	Base	1.0	1.0	1.0	Core=80/Freq=2.0/Issue=4/L2=1024k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	1	1	1	1
	DRAM	2.4	1.4	1.7	Core=128/Freq=2.0/Issue=2/L2=512k/DS=16GB/nDS=16/DS_BW=1.6GBs/Net_BW=0.001GBs	1.0	0.8	1.0	1.0
	PCM	25.4	6.2	4.1	Core=128/Freq=0.5/Issue=2/L2=512k/DS=25GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	10.2	2.0	12.8	10.2
	MRT	8.5	2.1	4.1	Core=128/Freq=0.5/Issue=2/L2=512k/DS=75GB/nDS=1/DS_BW=32.0GBs/Net_BW=0.001GBs	3.4	0.7	4.3	3.4

Figure 3.17: Configurations and scale multipliers of the baseline and SSD/DRAM/nanostore designs. ScaleX refers to the system-level scale factors relative to the baselines, as follows: Node (networked node count) OPS (peak compute throughput (Operations/second)) DS (Data store bandwidth) Net (Network bandwidth)

Table 3.17 presents additional data to provide further insight into these results (we focus on the non-shaded rows in this section). Columns 3, 4, and 5 provide the factor of improvement in the energy delay product, performance, and energy efficiency respectively. Column 6 summarizes the attributes of the best configuration chosen by the optimizer, and the last four columns present statistics on the multipliers of improvement in various system attributes. Specifically, Node, OPS, DS, and Net refer to the factor of increase in the number of processor sockets, and the total provisioned raw compute bandwidth, datastore bandwidth, and network bandwidth respectively. The results in Table 3.17 show that the greatest improvement in resources occurs for

the data store bandwidth, resulting from the combination of both the higher per-nanostore 3D-stacked bandwidth and lower per-device capacity. For example, with more than 5000 times higher bandwidth, the three I/O-intensive benchmarks no longer have any data store access bottleneck. With their small data store capacities, nanostores also allow significantly higher compute core and network bandwidths to match the increased data store bandwidth, regaining the balance across system resources to improve performance. By breaking the bandwidth wall in the conventional architecture, processor power density and network aggregate bandwidth now become the new, important system design constraints. (This also illustrates the reason why we capped these variables for a fair comparison with the baseline; Section 3.4.6 considers relaxing these constraints further.) Focusing on performance improvement, the biggest benefits stem from the increased parallelism of the nanostore solution that allows greater amounts of compute and network to be provisioned for smaller slices of data capacity. Using cksum as an illustrative example, the best PCM-based nanostore design (as determined by the design space search) uses 128 simple cores running at 500MHz in conjunction with the 25GB datastore. The design uses the lowest network bandwidth because this benchmark does not generate network traffic. The raw compute bandwidth increases by a factor of more than 1000 to match the 15000X increase in data store bandwidth, and these two resource improvement together provides three orders of magnitude better performance. (The OPS multiplier is lower than the Node multiplier due to processor power density constraint, that bounds the performance improvement).

Although cksum is an extreme example because of its low processing complexity, video illustrates the same performance benefits from higher data store bandwidth and matching, co-located, compute capability. To address the higher compute requirement over cksum, the optimal baseline node for video now has more cores and less

storage capacity. This leads to a lower Node multiplier and a lower OPS multiplier due to the power density constraint, explaining the lower performance improvement relative to cksum.

Sort is a benchmark with balanced compute, data access and network bandwidth requirements. In this case, the network bandwidth becomes the new bottleneck once the datastore bandwidth bottleneck is addressed, limiting the Net multiplier and subsequently, the performance improvement. For PCM-based nanostores, only 22 cores are required to match the limited network bandwidth; the Memristor-based nanostore has higher per-device capacity and correspondingly higher per-node network bandwidth for a given aggregate network bandwidth constraint, explaining its higher performance improvement.

For the two in-memory benchmarks, a similar analysis can be applied. The relatively smaller performance improvements compared to the IO-intensive benchmarks stem from a smaller DS multiplier over the high-bandwidth DRAM interface, and subsequently lower OPS and Net multipliers due to the power density and network bandwidth caps.

Our detailed analysis also identified (surprisingly) that the significant bandwidth improvements enabled by the 3D-stacked architecture were not being fully leveraged. One reason, as discussed above, is that the constraints on power density and network bandwidth can affect how well the bandwidth is used. Indeed, our additional experiments show significantly higher performance improvements when these constraints are relaxed (see Figure 3.21(c) in Section 3.4.6). Furthermore, our performance model and the COTSon-generated per-core memory bandwidth numbers used as input to the model are both conservative about the effect of improved memory bandwidth on performance, likely contributing further to these conservative results. Finally, the nanostore designs memory-like datastore latency has huge performance potential

for workloads that are random-access dominant and latency sensitive. However, the benchmarks we study are throughput-oriented and our performance model is mainly bandwidth based; therefore our results do not demonstrate this potential benefit.

3.4.3 Analysis of energy efficiency benefits

Besides significant performance gains, nanostores also achieve 2X to 16X improvement in energy efficiency. Below we discuss the three main contributors to nanostores better performance-per-watt. First, the NVRAM-based data stores are significantly more efficient than the hard drives or DRAM modules used in the baselines due to their better proportionality (no idle power) and lower access energy. For I/O heavy benchmarks, the hard drive access energy can be orders of magnitude higher than nanostore; while for in-memory workloads, the large DRAM capacity adds large idle power. The relative contribution of device power for *sort* for each hardware configuration is shown in Figure 3.18. In the unoptimized baselines (2 GHz 4-way issue cores) with either HDD or SSD (bHDD1 and bSSDx), CPU active power consumption is dominant. For the HDD baseline, almost 50% of total power is spent on the hard disk. Optimizing the processor and network bandwidth parameters reduces the impact of the CPU (configuration oHDD1), leaving the hard disk as the dominant power consumer. Similarly, optimizing the SSD-based designs cuts down on CPU power as more efficient cores are selected. For the RAM-based data stores (baseline bRAM4 and optimized oRAM4), memory idle power is the dominant component. Storing the entire 1 Petabyte dataset entirely in DRAM incurs an idle power penalty in the same way as the HDD-based system, but active power is much reduced. Finally, the PCM and Memristor-based optimized nanostore designs all have a similar power breakdown. The most notable difference is the slightly higher memory active power of PCM over Memristor. Core active power now makes up a third of the total

power, followed by a significant amount of chip leakage power. This is due to the use of low-frequency cores (active power less dominant) and necessary scale-up in the amount of processor dies in the system because of the much lower nanostore capacity than the HDD, SSD or off-chip DRAM configurations. The network power is the next highest contributor due to the larger network size (more nodes) and increased efficiency of other components which no longer dominate the power profile.

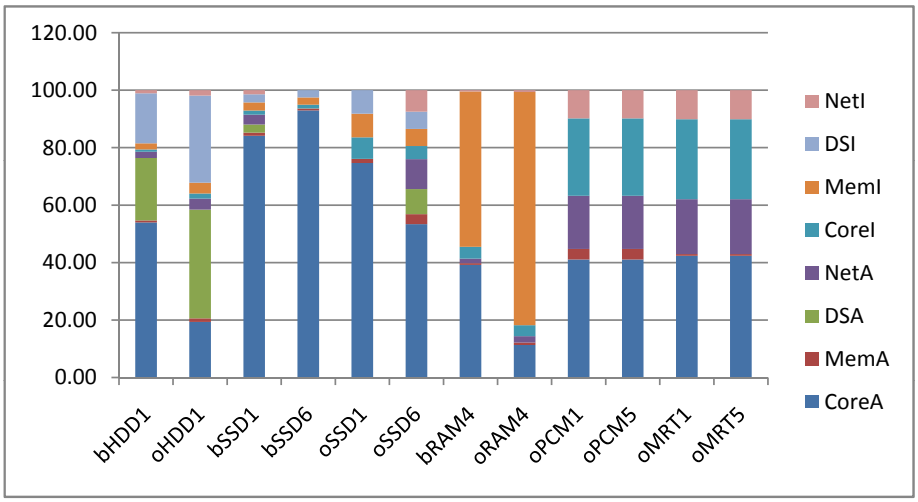


Figure 3.18: Power breakdown (% of total) for sort hardware configurations (energy-optimized). Net, DS, Mem and Core refer to network, data store, main memory and core power, respectively. A suffix of I refers to Idle (background) power, and A refers to Active power.

Second, compute co-location with lower per-nanostore capacity leads to the use of low-power, more energy-efficient processor cores to attain the same total throughput as the baseline. As shown in Table 3.17, nanostores often choose lower frequency, simpler cores, that are much more energy-efficient due to lower frequency and voltage.

Finally, having a single-level data store also provides the opportunity to avoid data movement between the logically separate segments of memory and persistent storage. Reducing the number of copy operations can improve performance due to reduced traffic and less energy for the same task, both leading to better energy

efficiency. These effects are hard to isolate with the integrated model we consider, but we performed separate experiments to study the elimination of redundant file load and save operations. Our results show significant traffic reduction in cksum and sort. This translates to about 10% efficiency improvement; the relatively low improvement stems from the efficient data stores in our designs. The benefits are more pronounced when the data access bandwidth is limited, as we cut down the amount of data movement. For example, cksum gets more than 30% better EDP from collapsing the main memory and storage layers, when per-nanostore bandwidth is 6.4GB/s. The memristor-based design achieves higher energy efficiency by virtue of having more energy-efficient data accesses than PCM, but as discussed earlier, has lower performance compared to the PCM-based nanostore because its higher capacity leads to a smaller Node multiplier.

3.4.4 Applicability of nanostore techniques in other system architectures

As discussed so far, the nanostore design achieves its benefits from a combination of several inter-related factors including high bandwidth per gigabyte, matching compute/network bandwidths, and co-location. Of these factors, the nanostore designs improvements to the system-wide data access bandwidth are fairly unique in comparison with traditional system architectures. First, cost-sensitive hard drives usually have a floor price (e.g., $30 for a mobile 2.5-inch drive) to amortize non-media costs. This effectively determines the hard drives minimum capacity, and sets it to a level much larger than for a single nanostore socket. Combined with the inherent low bandwidth of disks, it is not easy to apply nanostores compute/storage co-location principle that needs small storage chunks. Second, SSDs using current NAND flash technologies also have limited pins per package, having a higher, yet still limited, bound on bandwidth per device. Finally, DRAM and PCIe based SSDs can have

much higher bandwidth by exploiting device-level parallelism; however, compared to nanostores, they are still limited by the relatively narrow channels between the compute and the datastore.

To better illustrate the benefits from the nanostore design and the applicability of individual techniques to traditional architectures, Table 3.17 presents some additional data (shaded rows) listing the best SSD and DRAM-based systems. As before these points represent the outcome of the design space search optimized for the balanced design with the best energy delay product.

The results show that while the new DRAM and SSD-based designs show benefits, the nanostore designs still achieve higher performance at better energy efficiencies. The SSD designs share some of the energy efficiency advantages of lower idle power for the data store but suffer from lower bandwidth per GB; consequently, they have lower performance but higher energy-efficiency relative to the DRAM designs. Note that the best SSD-based designs also choose more efficient processor cores, as suggested by prior work using low-power processors with NAND flash [22, 23], but their scaling-down of the processors is much less aggressive than that of the nanostore designs. Interestingly, DRAM-based solutions often pick more powerful cores even if they are allowed to use more efficient, low-power cores. This is because the optimizer chooses to use faster, more powerful cores and hence more processor-power dominating solutions to offset the energy non-proportionality caused by DRAM idle power. In other words, more efficient and higher performance data stores can motivate the selection of more energy efficient processor cores, leading to additive efficiency benefits.

Result	Performance			EE		
Objective	EDP	EE	Perf	EDP	EE	Perf
Sort	17.3	11.3	23.9	7.2	8.5	0.9
Cksum	3234.8	674.6	3706.6	15.9	22.1	12.0
Video	337.4	72.9	362.2	3.2	4.8	2.2
Recom	3.4	0.9	3.5	1.8	2.6	1.2
Search	6.2	2.8	8.1	4.1	5.0	2.7

Figure 3.19: Impact of the optimizers objective function

3.4.5 Other objective functions

Figure 3.19 summarizes the results for the PCM-based nanostore when the optimizer uses other objective functions, under the same thermal and network constraints. For each benchmark, we normalize the performance and efficiency numbers over the same configuration (EDP-optimized baseline). The optimizer clearly chooses different configurations to reach different objectives. EDP is a good objective because EDP-optimized solutions usually have close-to-optimal performance and EE results across the entire table.

3.4.6 Impact of relaxed power density and network constraints

Result	Performance									EE								
Watt/cm2	32			50			100			32			50			100		
Net_BW	x1	x4	x16	x1	x4	x16	x1	x4	x16	x1	x4	x16	x1	x4	x16	x1	x4	x16
Sort	1	4	22	1	4	22	1	4	22	1.0	1.0	0.8	1.0	1.0	0.8	1.0	1.0	0.8
Cksum	1	1	1	2	2	2	2	2	2	1.0	1.0	1.0	0.8	0.8	0.8	0.8	0.8	0.8
Video	1	1	1	2	2	2	3	3	3	1.0	1.0	1.0	0.7	0.7	0.7	0.6	0.6	0.6
Recom	1	2	2	1	3	3	1	3	6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9
Search	1	1	1	2	2	2	2	2	2	1.0	1.0	1.0	0.6	0.6	0.6	0.6	0.6	0.6

Figure 3.20: Impact of thermal/network constraints

Figure 3.20 visualizes the effect of relaxing the socket power density (32, 50, and 100 W/cm^2) and network constraints (X1, X4, X16 aggregate network bandwidth)

bHDD1	Hard Disk (6.0 GB @ 0.5 GB/s), CPU (2.0 GHz, 4-way issue)
oHDD1	Hard Disk (6.0 GB @ 0.5 GB/s), CPU (optimized)
bSSD1	SSD (1.2 GB @ 0.8 GB/s), CPU (2.0 GHz, 4-way issue)
bSSD6	SSD (1.2 GB @ 4.5 GB/s), CPU (2.0 GHz, 4-way issue)
oSSD1	SSD (1.2 GB @ 0.8 GB/s), CPU (optimized)
oSSD6	SSD (1.2 GB @ 4.5 GB/s), CPU (optimized)
bRAM4	off-chip DRAM (256 GB @ 25.6 GB/s), CPU (2.0 GHz, 4-way issue)
oRAM4	off-chip DRAM (256 GB @ 25.6 GB/s), CPU (optimized)
oPCM1	3d-stacked PCM (25 GB @ 6.4 GB/s), CPU (optimized)
oPCM5	3d-stacked PCM (25 GB @ 32 GB/s), CPU (optimized)
oMRT1	3d-stacked Memristor (75 GB @ 6.4 GB/s), CPU (optimized)
oMRT5	3d-stacked Memristor (75 GB @ 32 GB/s), CPU (optimized)

Table 3.2: Hardware configurations. Optimized CPUs have their frequency, issue width and number of cores per socket adjusted for optimal system performance (throughput, energy or EDP)

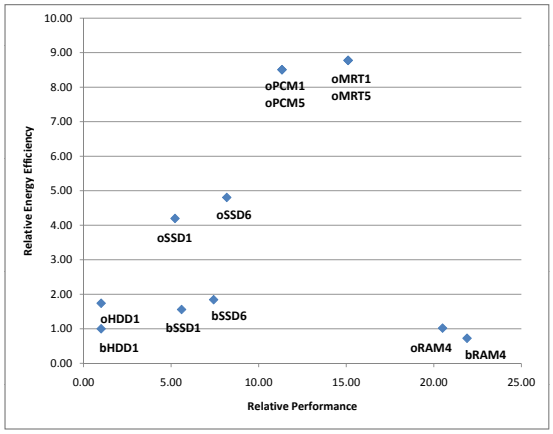
for our benchmarks. All results are normalized to the PCM nanostore design from Section 3.4.1 (X1 network bandwidth and 32W/cm^2). Darker shades illustrate improved benefits. Allowing higher power density has a positive performance effect for all workloads, matching our analysis in Section 3.4.2. Raising the network bandwidth cap only affects the two network-heavy benchmarks (sort and recom), especially sort where the network is the first bottleneck for performance scaling. Power density is the first bottleneck for recom, which has to trade core count with higher network bandwidth within the power envelope to get better performance.

Figure 3.21 compares energy-efficiency optimized designs and helps to illustrate the differing effects of the thermal and network constraints, as well as the effects of data store bandwidth provisioning. The hardware settings for each data point are listed in Table 3.2. In Figure 3.21(a), results are normalized to the hard-disk based baseline. The optimizer is able to use more efficient core parameters for oHDD1,

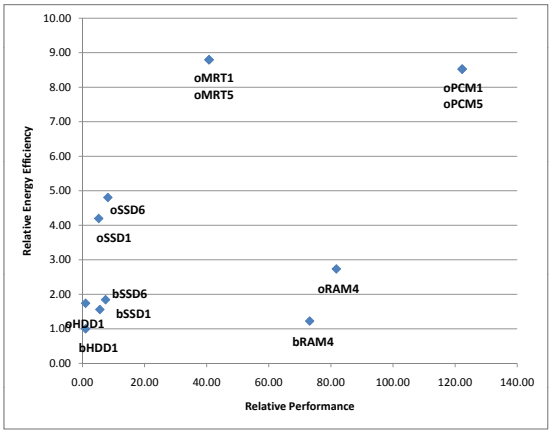
slightly improving energy efficiency, as with the optimized SSD designs. The SSDs have higher per-drive bandwidth (SSD1 has 1.0 GB/s and SSD2 has 4.5 GB/s peak bandwidth), and can correspondingly increase performance by removing the storage bandwidth bottleneck. Energy efficiency also increases with performance, as the core and network idle energy consumption is reduced during a shorter run. PCM and Memristor designs are not bottlenecked on the data store bandwidth, hence oPCM1 and oPCM5, as well as oMRT1 and oMRT5 have the same results. However, the larger socket count of PCM than Memristor to contain all the data, as well as its less efficient memory accesses slightly reduce overall efficiency. Finally, for the DRAM-based configurations, it is hard to improve on the baseline due to the high power of the DRAM-only data store. However, the high bandwidth and low latency of DRAM relative to SSD affords a large performance boost (much better EDP than the baseline).

Relaxing the network bandwidth constraint has a significant effect on the PCM, Memristor and DRAM designs (Figure 3.21(b)). First, all three designs can reach a higher performance point because they were bottlenecked on the network bandwidth. Second, by running faster, the DRAM-based design reaches mich higher energy efficiency by mitigating the large idle power. Third, the performance of the PCM-based design (with a third of the density and 3x as many chips as the Memristor design) increases significantly (along with an increase in 100 MHz cores per chip from 8 to 88 to saturate the bandwidth).

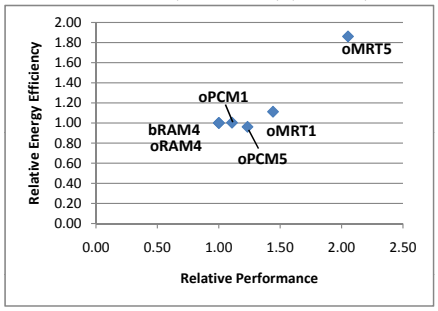
Finally, Figure 3.21(c) shows the recommender benchmark under a tight network bandwidth constraint (10 Gbps/node) but a relaxed thermal constraint (100W/-socket). By allowing higher socket power, the Memristor-based design is able to exploit the increased data store bandwidth (32 GB/s vs 6.4 GB/s) to reach a higher performance and efficiency point.



(a) Sort (energy-optimized, T=32W, N=10Gbps)



(b) Sort (energy-optimized, T=32W, N=80Gbps)



(c) Recommender (energy-optimized, T=100W, N=10Gbps)

Figure 3.21: Energy-efficiency and performance relative to baseline designs under different Thermal and Network bandwidth limits.

3.4.7 Model Validation

We performed experiments using a real implementation of our sort benchmark to validate the models used in Nanostore evaluation. Although validating every workload model is beyond the scope of this work, the sort benchmark has one of the most complex communication phases (with simultaneous computation) and establishes the general validity of our approach.

Phase 1: Shuffle

Because sort consists of two non-overlapping time phases, we measure each one separately. The first phase is by far the most complex and requires simultaneous

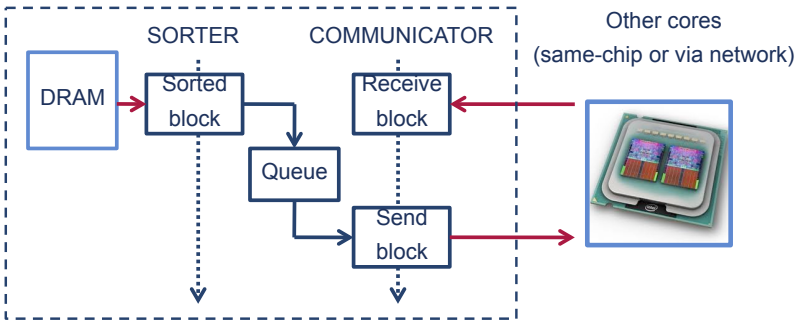


Figure 3.22: Block diagram of MPI-based shuffle phase software used in model validation

computation (sorting data buffers) and communication (sending and receiving data buffers to/from other nodes). By overlapping computation and communication we make more effective use of the idle compute power available while network packets are in transit. The communication code was implemented using MPI 1.3, a well-established message-passing communications library. This implementation of in-memory sorting is practically identical to that of the execution plan in Figure 3.7. The code was run on up to 16 networked servers (representing Nanostores), each with a 1Gbit/s full-duplex Ethernet interface. This corresponds to theoretical upstream and downstream bandwidths of 128 MB/s for a total of 256 MB/s. Each server has 4-cores in total, consisting of two dual-core AMD Opteron 2216 HE processors running at 2.4 GHz. Therefore, we can evaluate sorting performance for different numbers of cores and networked nodes. Further, by throttling the available bandwidth on each link using the wondershaper utility [111], we can simulate the effects of having different (reduced) bandwidths available to the Nanostores.

We assume that during data storage transactions prior to the start of sorting, the data records have been separated into files of N contiguous data ranges (one for each node in the system). Hence the data in each block already falls within the range of keys appropriate for any one destination node. The software creates one

process per CPU core, and creates two threads per process as illustrated in Figure 3.22. The first thread sorts 4 KB blocks of 32-bit random keys in memory buffers and adds them the the "ready queue". Meanwhile, the communication thread listens for incoming blocks, while monitoring the ready queue for sorted outgoing blocks. Whenever blocks become ready, they are sent by the communication thread. When the sends complete, their send buffers are freed so that they can be re-used to send more blocks of sorted data. In our experiments, each core is allocated 60 concurrent send buffers, and one concurrent receive buffer for every node in the system. That way, communication parallelism should be maximized. The performance results for phase 1 are as follows. Figure 3.23(a) shows the per-node external network bandwidth. Blocks are not sorted before they are transmitted, so these tests represent the performance of just the communication code. As the number of nodes increases from 2 to 16, there is a decrease in achieved bandwidth of approximately 25%. With 2 nodes, a peak bandwidth of 200 MB/s is achieved, close to the theoretical maximum of 256 MB/s. Increasing the number of cores from 1 to 2 per socket slightly improves throughput, but the workload is clearly network-bound as expected.

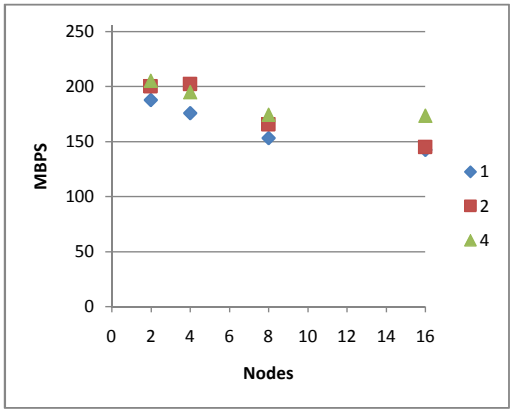
For a 2-node configuration with one core per node, when buffers are sorted before sending, the sorting throughput is approximately 80 MB/s. With a single core in use per node, it is clear that when sorting is enabled, the system should become compute-bound instead of network bound. To validate this, Figure 3.23(b) shows the same experiment, this time with buffer sorting before sending. There is a significant drop in bandwidth from the network-only experiment. As the number of nodes in the system increases from 2 to 8, the network bandwidth also increases, and flattens off at around 8 nodes. It should be noted that for small numbers of nodes (N), a fraction $\frac{1}{N}$ of data blocks remain on the same chip (for example, with 2 nodes, half of the blocks remain on the same node). Because many of the blocks do not traverse

the network, it makes sense that external network bandwidth is lower in these cases because we are compute-bound. For a Nanostore-like system, our conclusions should be drawn at higher node counts. As the number of cores is increased, there is a large step in throughput from 1 to 2 cores, and a much smaller step from 2 to 4 cores per node. Intuitively, this is because the compute-bound sorting bandwidth with 2 cores (2 cores $* 80MB/s = 160MB/s$) is relatively close to the outbound bandwidth limit of 128 MB/s, while beyond 2 cores, we become even more network bound. We validate this in Figure 3.23(c) where we throttle the network bandwidth in each direction to 50 MB/s. For all core counts, the bandwidth collapses to that of the single-core configuration while network-bound.

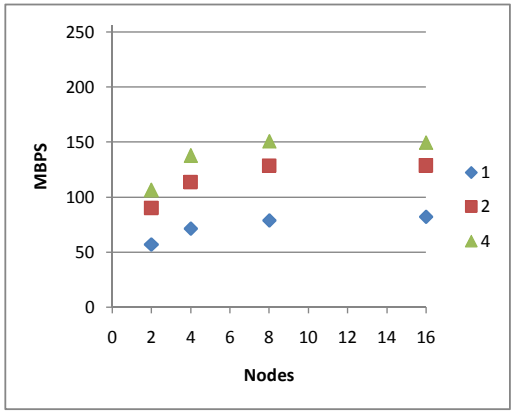
We validate the Nanostore model in Figure 3.24, which shows the measured and predicted aggregate sorting throughput of the system. This plot does not limit the network bandwidth, and there is almost a linear increase in sorting throughput with number of nodes. It can be seen how the step up to 4 cores causes throughput to saturate on the network bandwidth. The plotted lines show performance as predicted by the Nanostore model. We use a per-node network bandwidth limit of 150 MB/s (estimated at 16 nodes from 3.23(a)), which is approximately 75% of the peak throughput we saw. With the 80 MB/s/core compute bandwidth, the per-node bandwidth is calculated as

$$N_{nodes}.MIN(N_{cores}.80, 150) \text{ (nodes.(MB/s/node))} \tag{3.5}$$

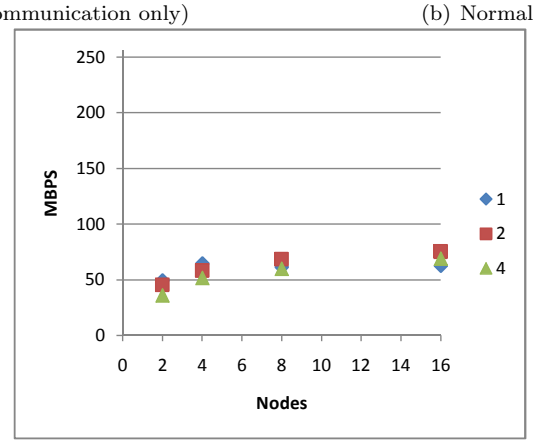
There is a larger divergence between the real and modelled throughput for the two-core configurations. By using a sorting throughput of 80 MB/s, the model predicts that network bandwidth will be saturated using two cores. In reality, there appear to be additional overheads from increasing the core count that reduce the sorting



(a) Maximum (communication only)



(b) Normal (with sorting)



(c) Network throttled (with sorting)

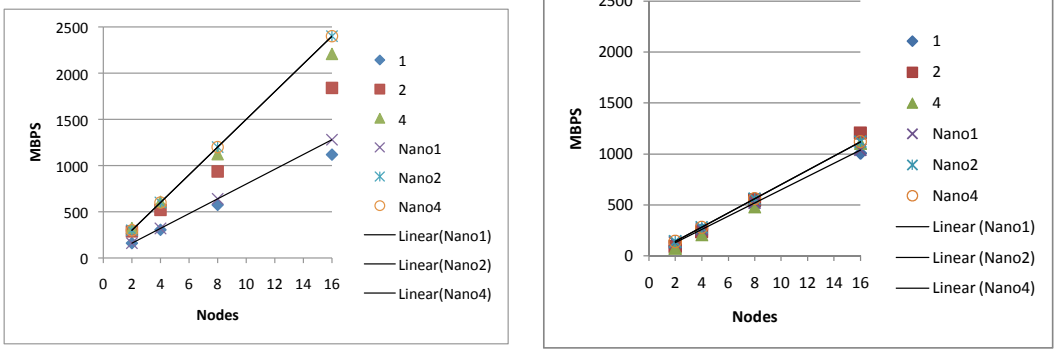
Figure 3.23: Per-node network bandwidths. The three points in the legend represent the number of cores per networked node.

throughput. These could stem from operating system or library code, or contention in the memory system. When the modelled sorting throughput is changed to 60 MB/s, the model fits the measurements more precisely (Figure 3.25).

We perform the same experiment after throttling the bandwidth (Figure 3.24(b)). The Nanostore model uses the same equation but with a reduced aggregate network bandwidth of 75 MB/s (peak bandwidth throttled to $50+50 = 100MB/s$ and scaled again by the 75% factor representing loss of efficiency at 16 cores). Again, the model closely tracks the way the 2- and 4-core configurations become network-bound.

These experiments serve to illustrate the general applicability of our bandwidth-

based performance model to real code.



(a) Normal (with sorting)

(b) Network throttled (with sorting)

Figure 3.24: Aggregate data sorting bandwidths. The first three legend items represent measured results for different numbers of cores. The points labelled NanoX represent the theoretical Nanostore performance model for each core count (modelled assuming 80 MB/s local sorting throughput), and their best-fit lines are also shown.

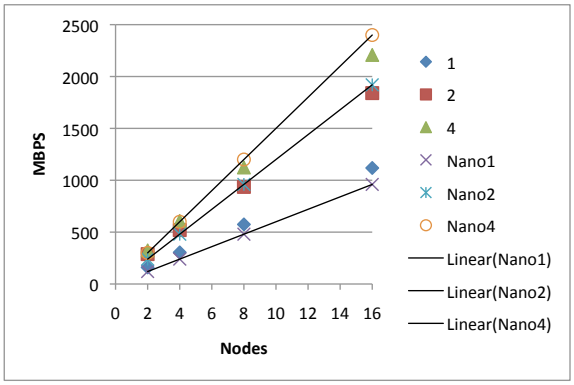


Figure 3.25: Aggregate data sorting bandwidths (Nanostore models assume 60 MB/s local sorting throughput).

Phase 2: Merge sort

For the second phase (local merge-sort) we ran the nsort [112] workload on the same server to estimate data throughput. It input consisted of different numbers of 10 MB files containing random 32-bit keys. The sort (single-threaded) is run multiple times to ensure that the files have been cached in RAM before being merged.

The results are shown in Figure 3.26. As the number of files being merged increases, there is a slight performance degradation before leveling off at a rate of 13 MB/s. To check that the simulated and real memory bandwidths were similar, we used the oprofile utility [113] to sample the CPU hardware performance counters. By recording the number of cache writebacks and misses and assuming a 64-byte cache line size, per-core DRAM write and read bandwidths were 58.1 and 70.6 MB/s respectively. Comparing this with the simulated results (using a 2.0 GHz 4-way issue core with 1MB L2 cache), we obtained 62 and 76 MB/s bandwidths. These are both within 10% of the measured results.

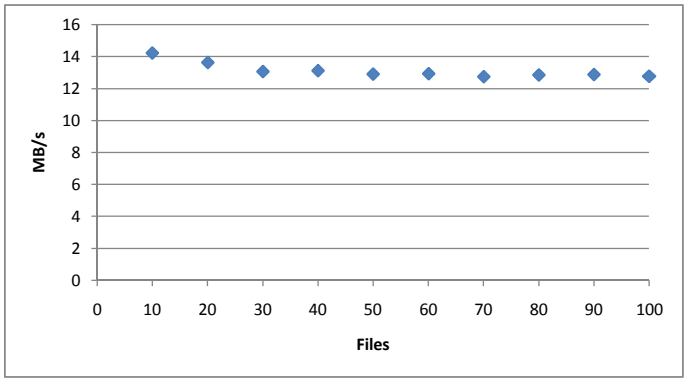


Figure 3.26: Measured nsort merge throughput

3.4.8 Discussion

Endurance is an important issue to consider. For the peak memory bandwidth we consider, in theory, storage wear out can occur in 2 years for PCM or 11 years for Memristor based on nanostore capacity and endurance. However, in practice, not all applications sustain rates at that level and the average across the application is much lower, leading to much longer lifetimes across the array. Wear-leveling schemes must still be used to spread writes across the entire memory to prevent early failure of hot data blocks. In Section 4.2 we estimate the lifetime of a Nanostore system using a

recent wear-leveling scheme. In Section 4.3 we also go further and look at how wear-leveling can be performed between nodes as well as within them. Another important issue is around scaling of workloads. The performance improvements from nanostores stem from the larger distributed scale of the workloads, with scaling factors ranging from 100 to 500. Even with the workloads we consider that are targeted at large-scale distributed implementations, such scaling is likely to pose challenges. Our idealistic assumptions around scaling are not meant to gloss over the challenges of scaling, but rather to provide an upper bound on the potential benefits. However, it is worth noting that over the decade from 1998 to 2009, Google's infrastructure is reported to have scaled performance (queries processed/day) by 1000X while scaling the infrastructure by 1000X [26]. In this work, we focus primarily on architectural and technology implications for best future designs, but cost is another issue that also needs to be considered. Current flash memories are about an order of magnitude higher cost on a $/byte basis compared to disk. The NVRAM memories we consider have the potential to lower these costs by more aggressive stacking and simpler fabrication processes. The improved energy efficiency of our design can also further lower total costs of ownership. Based on these observations, we expect the nanostore design to be competitive in costs compared to traditional designs, but this needs to be validated with further study.

3.5 Conclusions

With data volumes and data-centric applications on the rise, there is an emerging market for new system designs targeted at these workloads. At the same time, emerging technologies like 3D-stacked non-volatile memories are likely to disrupt traditional assumptions around storage latency and bandwidth. We argue that the best (and most intuitive) way to leverage the confluence of these application and technol-

ogy trends is a radical approach that co-locates processors with non-volatile storage eliminating intervening levels of the storage hierarchy. Our primary contributions are in developing the design of such an architecture and evaluating its potential benefits and implications. Specifically, we present nanostores, a new building block for data-centric system design. A nanostore is a single-chip computer that includes 3D-stacked layers of dense silicon non-volatile memory with a layer of compute cores and a network interface. A large number of individual nanostores communicate over a simple interconnect and run a data-parallel execution environment like MapReduce to support large-scale distributed data-centric workloads. The key aspects of our approach are large-scale distributed parallelism and balanced energy-efficient compute in close proximity to the data. Together, these allow nanostores to potentially achieve significantly higher performance at lower energy. Using an evaluation model and a benchmark suite that we newly designed for this study, we demonstrate orders of magnitude improvements in performance at significantly better energy efficiency for key classes of data-centric workloads. We also point out key challenges that need to be addressed to leverage this potential including scalable software design and improved network subsystem design. While our results are promising, we believe we have only scratched the surface of what is possible. We are currently examining the rich architectural space enabled by nanostores, including heterogeneous designs and integrated optics. There are also interesting opportunities for software optimizations including new interfaces and management of persistent data stores. Looking further out, the large scale and low latency of our designs will likely enable new previously impossible applications for more sophisticated insight generation across larger diverse multiple data sources; the corresponding hardware-software codesign provides rich opportunities for future research.

CHAPTER IV

Non-Volatile Memory Reliability for Data Center and Server Applications

4.1 Wear-out mitigation techniques

The possible approaches to wear-out mitigation are well summarized by [102], as follows;

- Wear-leveling algorithms
- Mechanisms to reduce the number of writes to non-volatile memory when possible
- Mechanisms to gracefully handle cells that fail permanently while programs are running

In an effort to address the lifetime limitations of non-volatile data center storage, we propose improved methods of applying some of the above points. In Section 4.2 we first estimate the lifetime of the Memristor and PCRAM Nanostores proposed in Section III, and propose a novel whole-system distributed wear-leveling scheme in Section 4.3 [13]. In Section 4.4 we propose improvements to increase the lifetime of NAND Flash based disk caches by gracefully handling run-time bit cell failures [4].

4.2 Estimating storage device wear-out in the Nanostore architecture

In this section, we estimate device lifetimes based on the workload write bandwidths observed in Chapter III. Wear-out in non-volatile memory technologies (Chapter 2.3) can limit the useful lifetime of our system. Assuming that a certain number of writes can be sustained per memory location before failure, and a wear-leveling algorithm that results in first block failure within some fraction of optimal (uniform writes), it is possible to estimate time before the first block failure. We assumed Start-Gap wear leveling with an efficiency of 90% of optimal wear-leveling, which was shown to be realistic for OLTP and database workloads [51]. Combining this with the memory write bandwidth results for the 32x 500 MHz 2-way issue core configuration, we estimate a per-socket lifetime of at least 7 years for PCRAM using a 512 KB L2 cache or at least 18 years using a 1 MB L2 cache (Figure 4.1). The larger cache reduces write bandwidth for these benchmarks. This estimation indicates that Nanostore systems will have acceptable device lifetimes.

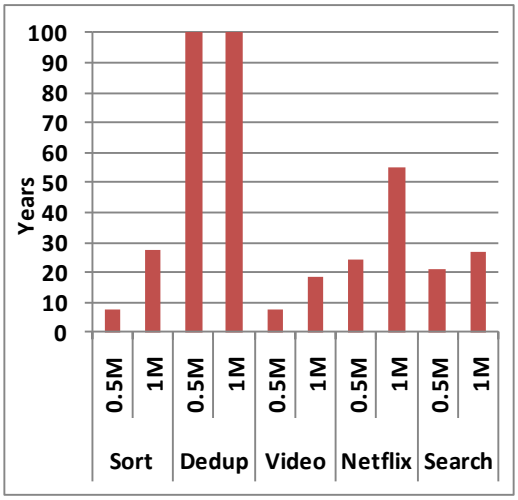


Figure 4.1: Estimated lifetime in years for PCRAM based system. Cache sizes in MB are shown below each column

4.3 Distributed Wear Leveling

Non-volatile (NV) memories such as flash and phase-change have recently emerged as promising memory/storage technologies for future computing systems. Wear leveling is widely used for these devices to improve lifetime by evenly spreading writes across memory cells that only have limited endurance cycles. Existing wear-leveling techniques are implemented at the individual device or server level, although the use of NV memories in the datacenter often involves an ensemble of devices or servers. To simplify system management tasks, it is desirable to improve the ensembles lifetime by increasing its time to first failure (TTFF). In this section we propose distributed wear-leveling (DWL) [13] techniques applied at runtime to increase TTFF in a datacenter, reaping additional benefits on top of existing local wear-leveling policies. We predict 3.9x longer TTFF for a collection of NAND Flash SSDs over a baseline using local wear-leveling only. In addition, our predictions point towards a low performance impact. To the best of our knowledge, this thesis is the first publication to propose coordinating memory wear-out across independent, distributed storage drives.

4.3.1 Introduction

Solid state disks are finding applications in servers due to their higher supported I/O operations per second (IOPS) relative to mechanical hard disks (HDDs). They can be attached to servers via conventional disk interfaces such as SATA or via higher bandwidth interfaces such as PCI-express [29]. As well as their performance benefits, they use considerably less idle and active energy. Despite hard disk drives being able to spin down into a low power state, subsequent accesses take some time to start up again. These aspects present SSDs as an attractive alternative or addition to HDDs.

Because all current non-volatile memories have limited write endurance, several approaches have been taken that effectively increase the lifetime of an SSD as a whole, namely wear-leveling and fault tolerance techniques. At a high level, wear-leveling seeks to spread writes evenly across every cell of the SSDs. In doing so, information must be maintained to map the datas logical addresses to their physical locations, and is typically handled by software or firmware (for example, in a Flash Translation Layer or FTL). The multiple-page erase-block structure of NAND Flash is particularly demanding in terms of extra information (metadata) needed to track the wear-out state and address mappings. Additional writes are often needed to relocate valid data (write amplification) during operation, adding to the overall wear-out. Depending on the device technology and physical structure of the device, wear-leveling policies of different complexities can be used. There are three key motivations behind this work, which are (1) reducing management costs for an ensemble of SSDs, (2) addressing future lifetime challenges for emerging multi-level cell (MLC) memories and (3) handling non-uniformity in workloads. Regarding management costs (point 1), because SSDs are still not as cost-effective as HDDs, they can be used as a caching layer above HDDs for the most heavily accessed data [74, 80]. Shared NAND Flash based caches have also been considered to cater for multiple hard disk drives in a networked system [81]. When used as a cache, the overall system performance can be greatly increased as well as potentially reducing energy consumption for the underlying HDDs. However, an aspect of large-scale non-volatile storage integration that has been overlooked is ensemble-level management of SSDs and wear-out balance between multiple drives. Black Box data centers have been proposed as a more modular and efficient design than ad-hoc data centers built from the ground up. Google patented the data center in a shipping container, [114], Suns Modular Datacenter [115] and HPs POD [116] combine compute, storage and networking into a portable

unit. In this trend towards modularity and reducing management costs, reducing the frequency of unpredictable hard disk failures becomes more important and is the focus of our paper. Regarding future lifetime challenges (point 2), emerging data center workloads involve deep analysis of increasingly larger quantities of data with complex relationships. For example, Google Squared [19] infers important aspects of a topic from massive volumes of data, going beyond the conventional web search applications of the past. This is putting greater pressure on the storage subsystem. Modern SSDs are specified to last for several years under typical usage using their built-in wear-leveling policies. However, the move towards MLC and lower costs are reducing endurance. As an example, a heavily loaded 256 GB SSD at 100 MB/s write bandwidth and 10^5 write endurance per bit gives up to 8.1 years of lifetime. However, predictions for higher density 4-bit MLC available in 2012 may only have an endurance of 10^4 [3] reducing that lifetime to 0.81 years (for the same capacity). This shows a fundamental limitation of local wear-leveling in a single SSD, so we consider the possibilities for distributed wear-leveling across non-volatile memories in larger systems (point 3). Non-uniformity between SSD write bandwidths arises from heterogeneous workloads, skew in the distribution of data, and at a higher level, heterogeneous SSD technologies (e.g. SLC/MLC/Flash/PCRAM) and device process variations. The contributions of this work are as follows;

- Analysis of data center storage characteristics, revealing inherent write bandwidth imbalance between drives and predictability between data block re-writes, enabling simple distributed wear-leveling.
- A novel system architecture for monitoring and communicating wear-out statistics in distributed systems. This includes block access monitoring behind all volatile caches, as well as specialized Monitoring Nodes that are given the task

of managing SSD lifetime across servers.

- Novel on-line algorithms to perform distributed wear-leveling. They increase system-wide time to first failure (TTFF) with low write bandwidth overhead and low performance impact.

4.3.2 Background and Motivation

Non-volatile memories and Solid-State Disks

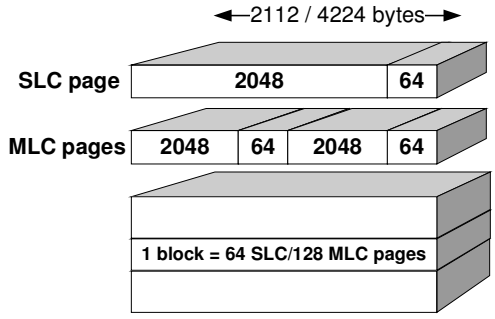


Figure 4.2: NAND Flash plane

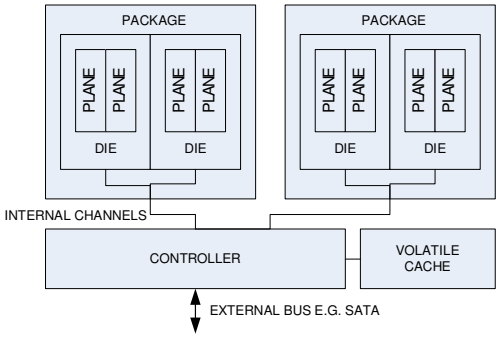


Figure 4.3: NAND Flash SSD

Several memory technologies have been employed in SSDs. DRAM based SSDs [117] have the advantage of very high performance and practically infinite write endurance, but suffer from relatively high power consumption. Further, they are volatile and lose their contents in the case of power loss, so backup power supplies

have to be available. This limits the utility of DRAM as a permanent data store. Considering charge-based non-volatile memories, Flash memory is a well-established technology for SSDs and comes in two main variants. NOR Flash has similar read latency characteristics as DRAM, but very high erase latency. For this reason, application as a hard disk replacement is not feasible if the sustained write bandwidths are to be matched. NAND Flash is the current technology of choice for SSDs. NAND Flash read latency is higher than that of NOR Flash, but erase latency is much lower. Further, they offer superior device density, even more so when using multi-level cells that store more than one bit per physical location (MLC). Write endurance per bit is currently limited to between 10^4 and 10^5, however, limiting the life span. More recently, resistance-based devices such as phase-change memory (PCRAM) are being produced and potentially improve on some of the limitations of Flash memory. Although expensive and still in its infancy, PCRAM has potential to scale to higher densities than Flash, with no erase requirement and higher write endurance. Our techniques apply equally to these emerging technologies, and can even enhance wear-leveling policies proposed for them as we will demonstrate. The non-volatile memory that we consider is NAND Flash. We assume that the chips are physically packaged into SSDs that provide a communication interface and management firmware. SSDs take advantage of multiple parallel chips to provide a high level of concurrency, supporting high bandwidth and IOPS. Agrawal et al. [79] provide a good introduction on internal NAND Flash chip and SSD structure. Flash chips are organized into pages of typically 2KB-8KB. The pages are organized into blocks that are erased as a single unit, before being written to. Page writes within a block must proceed sequentially. Figure 4.2 shows pages composed of single-level cells (SLC) or 2-bit multi-level cells (MLC) arranged into erase blocks. Every page has 64 bytes of spare area that are often used for storing metadata or ECC bits. At a higher level (Figure

4.3), each Flash package contains a number of die that share an I/O bus. Commands can operate concurrently between the die, and commands can also operate concurrently on a number of internal planes of erase blocks. Externally, the Flash packages share a number of buses to an SSD controller that performs management functions and interfaces with the host server.

Wear-leveling

Please refer to Section 2.3.1 for background information on wear-leveling.

4.3.3 Experimental Methodology

Hardware Configuration

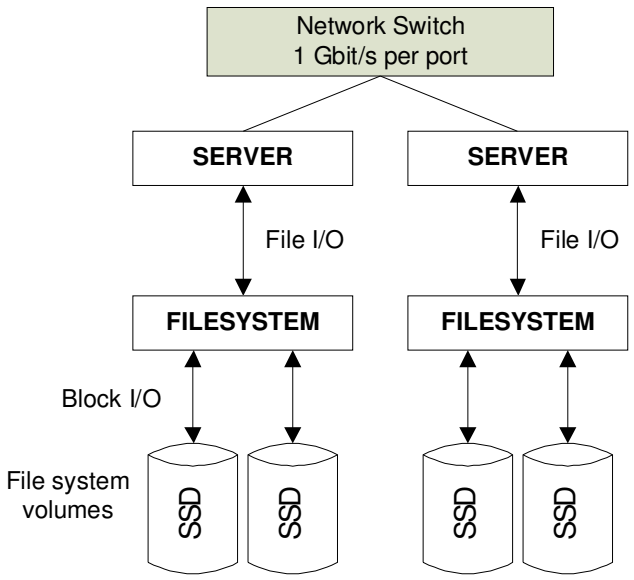


Figure 4.4: High-Level System Configuration, This figure shows the physical organization of the baseline system. All 13 servers are connected via a Gigabit Ethernet switch, supporting 1 Gbit/s (128 MB/s) of bandwidth for each one. Every volume is replaced with a 1 TB SSD drive.

Figure 4.4 shows an abstraction of the data center infrastructure. Servers with different purposes access a number of physical volumes via the filesystem. It is

Key	Description	Volumes
Usr	User home dirs	3
Proj	Project dirs	5
Prn	Print server	2
Hm	Hardware monitor	2
Rsrch	Research projects	3
Prxy	Web proxy	2
Src1	Source control	3
Src2	Source control	3
Stg	Web staging	2
Ts	Terminal server	1
Web	Web/SQL server	4
Mds	Media server	2
Wdev	Test web server	4

Table 4.1: List of traces

assumed that the filesystem is distributed and any server can access any physical volume for reading and writing (via local or network interfaces). Existing high-level filesystems such as the Network File System (NFS) [118] could handle cross-server accesses. On each server, a virtualized file system tailored for non-volatile memory such as the Direct File System DFS [119] would provide high performance and a large virtual address space that has a backwards-compatible software interface.

Workloads

To explore the potential for distributed wear leveling, we used traces from the Microsoft Research Cambridge data center. They consist of 1 week of block-level read/write access data for 13 servers each serving a different function, with a total of 36 storage volumes. Summary information about the traces are provided in Table 4.1, reproduced from [81].

Previous system architecture papers [84, 81] made use of these traces and analyzed

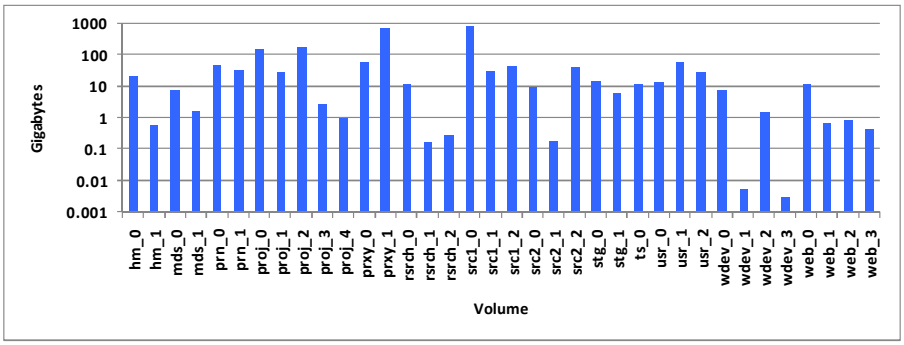
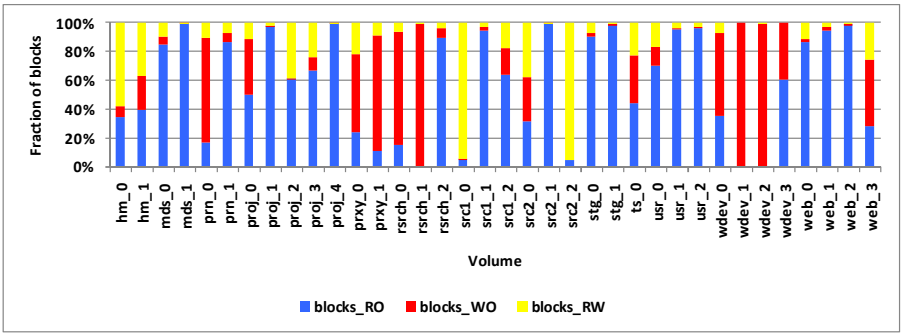


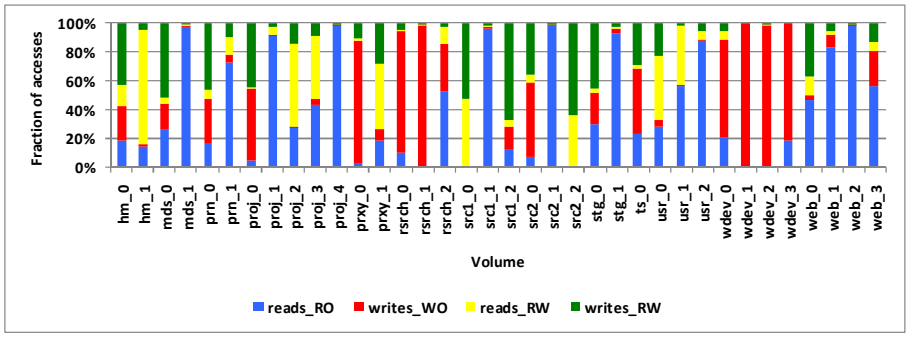
Figure 4.5: Total data written (GB) per volume over the 7-day tracing period

them for different purposes. In particular, they found that the workload is very bursty and the server producing the top 1% of accesses changed significantly over time. This motivates an adaptive mechanism for handing wear-leveling. We believe that these workloads are representative of data center workloads and sufficient to illustrate the principles of distributed wear leveling. Because we are most concerned with writes rather than reads, we examined the total data written to each volume over the entire 7-day period (Figure 4.5). The figure confirms our assertion that distributed workloads can have very large write imbalance between the installed storage devices. We identify servers prxy_1 and src1_0 as the dominant source of writes. These two require closer examination since the distribution of their blocks across volumes will have the greatest overall impact on wear-out.

To process the traces and proposed algorithms using a reasonable amount of system RAM, we opted to record accesses for sequential regions of 128 LBA blocks (64 KB in size) to restrict memory usage during analysis. The breakdown of block and access characteristics are shown in Figure 4.6. Figure 4.6(a) shows that most data blocks either read-only or write-only (also true for the prxy_1 volume), a situation also observed in [86]. Because we want to satisfy as many read accesses as possible from the local device for performance, this is potentially useful because few of the



(a) Classification of blocks into Read-Only, Write-Only and Read-Write



(b) Classification of all accesses to each type of block; reads to read-only blocks, writes to write-only blocks, reads to read-write blocks and writes to read-write blocks

Figure 4.6: Block and access classification. Data are for 7-day traces.

written blocks are re-read and can be off-loaded to other SSDs. However, for the heavily written src1_0 volume, most blocks are read-write. This causes a potential problem if we want to distribute blocks across servers for wear-leveling, because there may be many remote read accesses. Any block re-distribution algorithm we devise should try to maximize the number of read accesses satisfied by the local SSD by keeping the blocks with the highest read/write ratio local. Figure 4.6(b) indicates that although there are few read-write blocks in prxy_1, the majority of accesses are made to that small subset of read-write blocks.

Because the wear-leveling scheme needs to adapt to changing access patterns, we next examine the time behavior of the volumes that generate the most writes. 75% of all writes are generated by only three disks, so we study those in more detail.

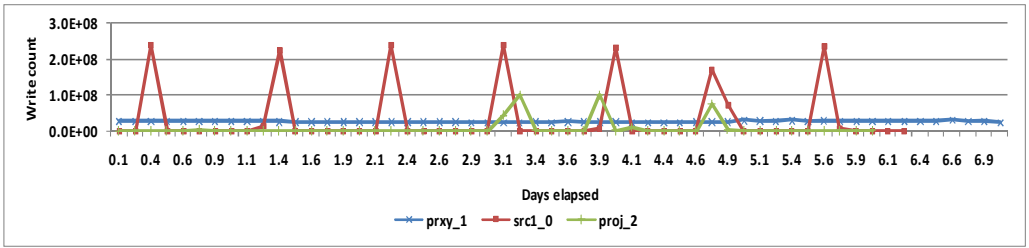
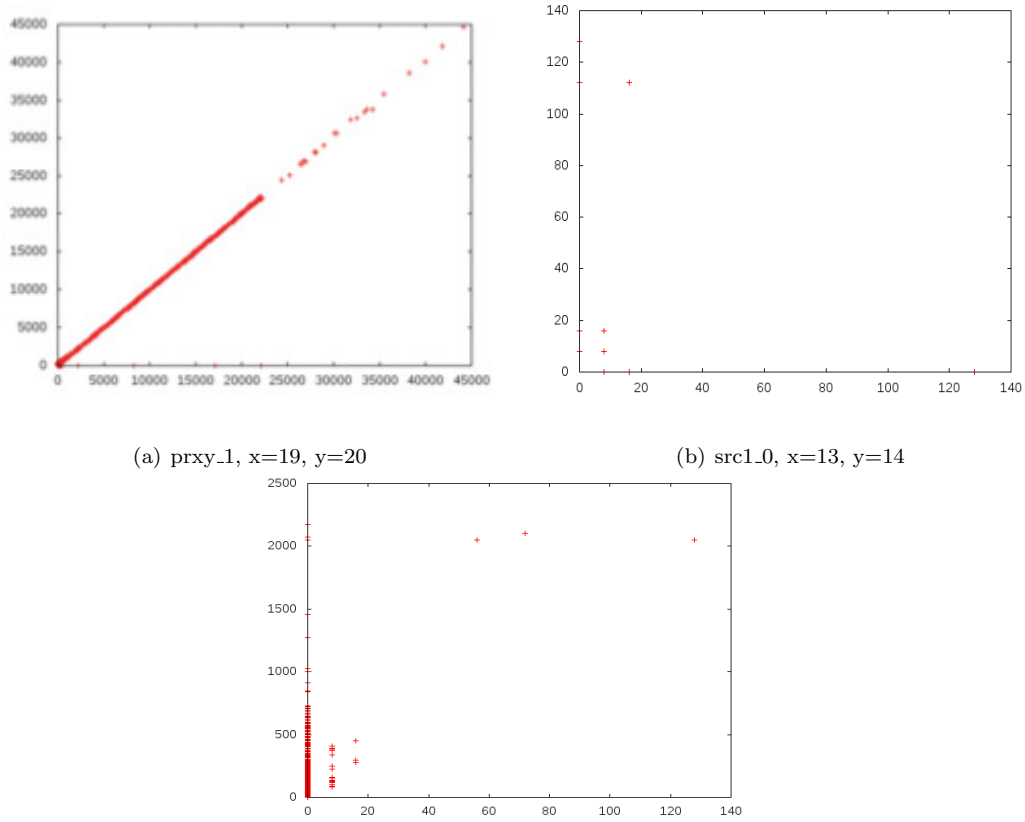


Figure 4.7: Time behavior of writes. Each data point is recorded for a 3-hour interval.

Ideally, we want to predict the number of writes to each block for the next time interval, so we can balance those writes across all SSDs by moving them there before the writes arrive. The time behaviors of the write-heavy volumes are significantly different (Figure 4.7). We also include proj_2 as the third highest contributor of write accesses. While prxy_1 appears to have a consistent rate of writing, src1_0 is periodic, with 24 hours between bursts. This is a feature that our block placement algorithm should take into account when predicting the total amount of data that is going to be written in a time interval. That way, we can estimate how many writes ought to go to the local drive (an equal fraction of the total, for each SSD).

To assess the predictability of accesses in the next period from the current one, we compare the write counts of each block in the current and earlier intervals (Figure 4.8). In this plot, the x-axis represents the write counts in the earlier interval and the y-axis represents write counts in the current interval. Each point represents a unique block address. A diagonal line in the plot would indicate perfect correlation with the same access count per block at each time. Points on the x-axis indicate blocks not written at all in the current interval, and points on the y-axis indicate new blocks being written.

For the correlation plots, there was an outlier block at 64K-address 48128. To better examine the trend for the majority of data points, this block was removed.



(a) prxy_1, x=19, y=20

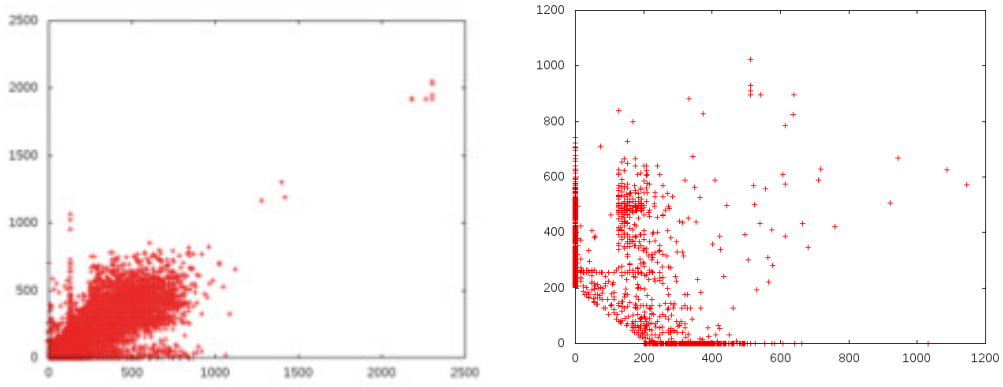
(b) src1_0, x=13, y=14

(c) src1_0, x=16, y=17

Figure 4.8: Correlation between time intervals. Outlier block 48128 has been removed. Each point represents a 64KB block accessed in either (or both) of successive 3-hour intervals. The x-axis represents write count to the block during the first, and y-axis the second interval.

We suspect that it is drive metadata, and could be moved to a battery-backed RAM to reduce its wear-out effects. Figure 4.8(a) indicates a good predictability from the prior interval for prxy_1. We confirmed this behavior is consistent for all successive time intervals. For 30 out of the 36 traces, we consistently see this type of correlation.

For src_2, we examined the idle and active periods in Figures 4.8(b) and 4.8(c). There is little correlation during the idle period and a burst of addresses being written at the start of the busy period that werent written last period. In Figure 4.9(a) we show the correlation between successive busy periods, which is more significant. It appears that the majority of blocks are re-written and correlate to some degree.



(a) src1_0 (b) proj_2

Figure 4.9: Correlation between the earliest write bursts. Note that for proj_2, blocks with low write counts were omitted for faster plotting.

Soundarajan et Al. [83] also observed that many writes to block devices are in fact overwrites of popular blocks. This is useful because if we decide to move any blocks away from this volume, the write distribution will remain fairly consistent during future bursts. Unfortunately, there is not as much correlation for the first two bursts of proj_2 (Figure 4.9(b)).

Figures of merit

We use three main metrics for comparison of the distributed wear leveling approaches.

- Time to first failure (TTFF) improvement factor. This represents the factor by which the time of the first SSD to fail exceeds the first failure time in the baseline (non-wear-leveled) system. Based on the workload write bandwidth data from the previous section, we can put a theoretical upper bound on the TTFF factor. In an ideal system with no writes caused by data migration, all writes are spread evenly across all SSDs. That way, the TTFF factor = (Maximum volume write BW)/(Average write BW across all volumes). For the

hardware system and benchmarks described in this section, the factor is 12.54.

- Average wear-leveling efficiency. This represents the ratio of number of blocks written to an SSD at the point where the first block fails, to the theoretical maximum number of writes (assuming perfect wear-leveling and equal writes to each location).
- Average access latency. This represents the average response time of an SSD to requests.
- Network slowdown. If remote accesses and migrations ever saturate a network link, the additional delay is accounted for as a slowdown.

Baseline Configuration

The baseline system consists of the 13 servers and 36 volumes (Figure 4.4) covered by the MSR-Cambridge traces. Each volume is represented by a single SSD with 1 TB capacity to accommodate the full address range from each individual trace, as well as from multiple traces after re-mapping. We chose reasonable device parameters for the internal organization of the NAND Flash SSD (Table 4.2).

To evaluate the average response time we use the DiskSim 4.0 simulator [120] with Microsofts add-on SSD module and associated page-mapping FTL [79]. As per their paper, each SSD initially reserves 15% of its pages as unused and invokes garbage collection when capacity falls below 5%.

Performance Assumptions

Because the traces were collected from a hard disk based system, the access times reflect the inherent disk latencies as well as application demand. Because it is hard to evaluate the direct performance gain from replacing HDD with SSD using these traces, and since significant prior work exists [84, 80, 121], we focus on lifetime

Parameter	Value
Technology	NAND Flash
Simulated endurance (writes/block)	10^4
Read/Write/Erase latency (μs)	25/200/1500
Addressable disk block size (LBA) (B)	512
Migration/monitoring block size (KB)	64
Page Size (KB)	8
Pages per Erase block	128
Chips per SSD	16
Planes per Chip	8
Blocks per Plane	8192
Total Capacity (GB)	1024

Table 4.2: Baseline SSD parameters

results. However, because our scheme relies on the network between servers for remote data access, we model any delay due to periods of network activity that exceed the bandwidth of a 1 Gbit/s interface for each server using timestamps from the original traces. We also record the average response time across SSDs to show how distributed wear leveling changes drive latency. We do not add any additional latency for network delay as this is small (microseconds) relative to SSD latency (milliseconds).

4.3.4 Distributed Wear Leveling

Our distributed wear-leveling system leverages (1) a system architecture that monitors physical non-volatile memory activity and (2) algorithms that carefully select a set of data blocks to migrate to other volumes based on write access history and current state of wear-out. The workload analysis was promising in that the majority of future data block writes correlate well with prior intervals, so any block migrated to another drive is likely to exhibit a consistent write frequency for good write bal-

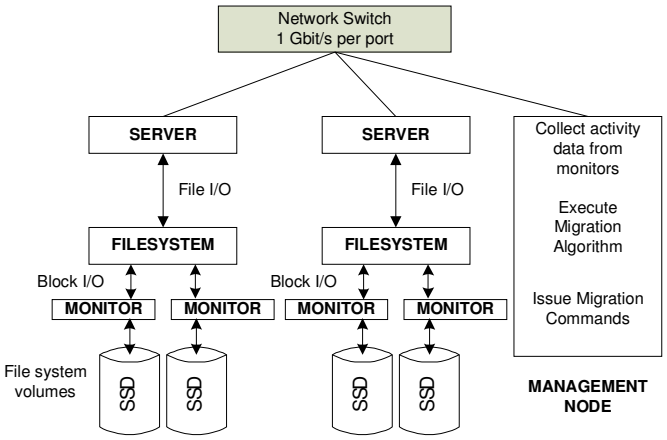


Figure 4.10: Top-level diagram

ancing. The run-time migration system requires two new physical components (see Figure 4.10) in addition to a distributed filesystem. Designing a dedicated filesystem to handle block distribution across networked nodes is beyond the scope of this work. However, we can consider potential implementations based on current proposals. The virtualized DFS [119] address space (which operates across multiple physical Flash drives) could be extended to operate across distributed Flash storage and currently supports $2^{32} - 1$ files. If each 64 KB block was stored as a small file, just over 14% of this limit would be used by the entire 36-disk system. As a temporary solution, the widely-used Network File System [118] in combination with DFS local to each server could provide the distributed file handling. At the Activity Monitoring nodes, each SSD collects the read and write counts of logical blocks at the granularity of a migration block. In our case, these are contiguous 64 KB blocks of data. This requires 128 MB of counters per 1 TB SSD assuming 32 bits per counter. Because SSDs typically have a volatile cache or buffer, we need to record the logical block access count after the cache to see the requests that reach the non-volatile memory. The statistics are then communicated to the Management node periodically (a total

of 4.5 GB for a 36-SSD system). Alternatively, the SSD cache behavior could be emulated at the block device driver layer (without actually storing data) to estimate the miss stream. Note that we do not use any caching for our evaluation, other than to remove the single block at location 48128 that has excessive writes for several volumes. The Management node is responsible for two tasks. The first is using the statistics from the Activity Monitoring nodes to form a migration plan. The migration plan schedules movement of some migrations blocks between servers to improve wear-out balance between SSDs. The second task is to update the distributed filesystem metadata to reflect migrated blocks as they are moved so that future requests are directed to the appropriate SSDs. Evaluating the wear-leveling performance of such a system involves summarization and replay of the workload traces subject to re-mapping. The following sequence of steps will be performed in order;

- Split the traces into discrete time intervals. We found that 3 hours provides enough detail to make good migration decisions.
- Obtain read and write counts for each migration block in each trace interval.
- For each interval, use the read/write count summaries to
  - Calculate the quantity of data written to each volume
  - Record reads and writes to each volume in current interval (re-mapped)
  - Form a migration plan and save the new mapping for the next interval
- Using the sequence of recorded migration plans, play back all traces simultaneously, interleaving in time order.
- Determine peak server-server network bandwidths and corresponding slowdown.
- Output re-mapped traces to DiskSim to find average latency and wear-out performance.

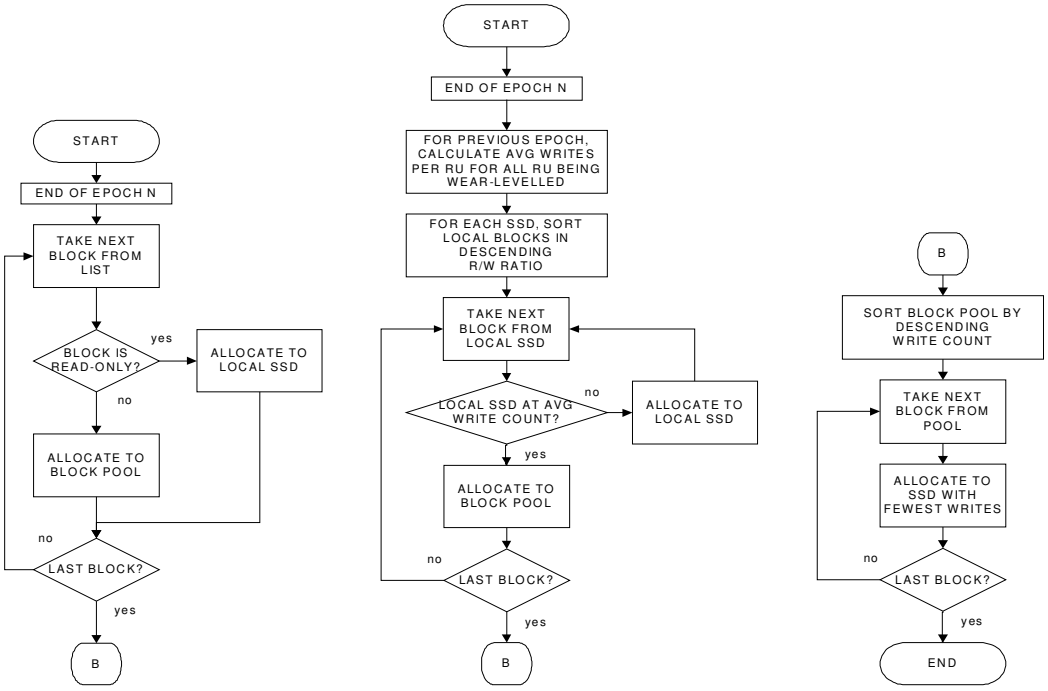


Figure 4.11: Distributed wear leveling algorithms

Figure 4.11 shows the wear-leveling algorithms executed on the management node. The basic algorithm consists of two phases. The algorithms are triggered at a fixed time interval of 3 hours. In the first phase, the read/write count for each migration block on each SSD is examined to decide if it should remain on the drive (local) or be migrated. The left-hand flowchart retains all read-only blocks locally and places all other blocks into a block pool that is common for all drives. As an improved alternative (centre flowchart), the total write count in the ensemble for the previous interval is scaled by the fraction of total storage capacity provided by the current SSD (1/36 in our case). This represents the desired number of writes to the SSD given perfect wear-leveling. We found that due to the daily burst of activity, using the write count 24 hours previously (after the first day) gives a better estimate of total writes than the previous interval, and this is used for our results. These are typical data center characteristics from real large-scale traces, and it is a simple extension

to dynamically detect and exploit a different periodicity, if present. As with any application, more data-intensive workloads could saturate the network, forcing us to consider a faster network or data replication to remove bottlenecks. This was not necessary for the Microsoft data center considered. Next, the blocks are ordered in descending read/write ratio, then allocated in sequence to the originating drive until the target write count is met. That way, the number of reads performed locally is maximized, reducing network read traffic. After that, the remaining blocks are recorded in a common block pool. In the final step for both algorithms, the block pool is sorted by descending write count. The blocks are then allocated in sequence to the volume that had the fewest writes so far, updating the anticipated write counts as every block is allocated (including one write for a migration). The sorting potentially results in a better balance of writes because as the pool runs out of blocks, it is easier to fill in the gaps with blocks having fewer anticipated writes. In summary, the following policies for these two stages of the migration algorithm will be evaluated;

- Local/Pool split: either Random split (RS) where all written blocks are placed in a pool for allocation to SSDs, or Local-First R/W prioritized split (LFPS) where blocks are ordered in descending R/W ratio, then remain on the local SSD up to some maximum write threshold.
- Pool allocation (POOL) Migration blocks are allocated in order of descending write count.

We could have taken same-server locality into account in our algorithm, preferring to make local disk transfers. Our network traffic accounting does consider that some disk transfers do not go off-server. However, we get good performance despite not explicitly considering this locality. The primary operation if the allocation algorithm

algorithm is to sort 32 million records per SSD (requiring less than a second for a modern multi-core processor) which can be easily parallelized. Therefore algorithm performance should be negligible relative to the long re-evaluation interval of three hours.

4.3.5 Results

Distributed wear-leveling

Figures 4.12 and 4.13 compare the performance metrics of distributed wear leveling against the baseline system. The two lines plotted against the left-hand Y-axis are the running average time to first failure for an ideal system with equal write counts sent to each SSD, and the actual time to first failure under each policy. Values are normalized to the TTFF of the baseline system. The actual TTFF value factors in the extra writes due to migration, which are plotted separately against the right Y-axis. The random split (RS) policy is largely ineffective due to the large number of migrations performed at each interval. LFPS performs relatively well, reaching a TTFF that is 4.2 times that of the baseline using past observations alone. In addition, the extra writes due to migrations account for 11% of total writes. The optimal TTFF is not reached because of remaining unevenness in the write counts to each SSD at the end of the 7-day period. The algorithm cannot achieve optimal wear-leveling because of more unpredictable, less correlated bursts of writes (for example, from proj_2 in Figure 4.9(b)). Further, writes to blocks that are never written again may be selected for migration, wasting an additional write.

Local SSD wear-leveling

In this section we estimate the impact of the FTL on writes. Due to time constraints, we replay 2 TB worth of writes, or 2x the total SSD capacity, so cold block

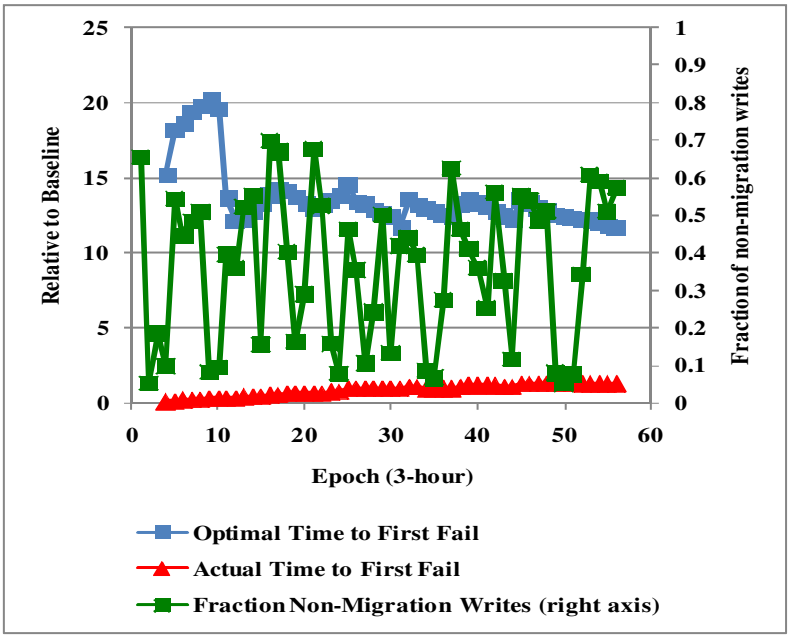


Figure 4.12: (RS+POOL) run-time wear-leveling policy. The x-axis represents elapsed time in 3-hour intervals. The y-axis shows improvement over baseline.

migration (swapping some written blocks with cold blocks [79]) was never invoked. This means that garbage collection is the only cause of block erases. Writing this much data required multiple replays of the 7-day traces, and took a long time to simulate. A larger number of replays that truly stress the FTL are necessary to estimate long-term lifetime, but this experiment indicates some potential lifetime improvements. The result we present in Figure 4.14 for write efficiency are similar to the fraction of non-migration writes from Figures 4.12 and 4.13. This factor is actually the inverse of the write amplification factor, which is the factor by which the writes to the physical SSD exceeds the requested writes. We calculated it by dividing the expected average number of writes per block for optimal wear-leveling i.e. 2, by the actual average number of erases per block. More heavily written SSDs from the baseline such as src1_0 and proj_0 see a slight decrease in write efficiency, but the originally less-heavily written drives such as wdev_1 and wdev_3 have improved

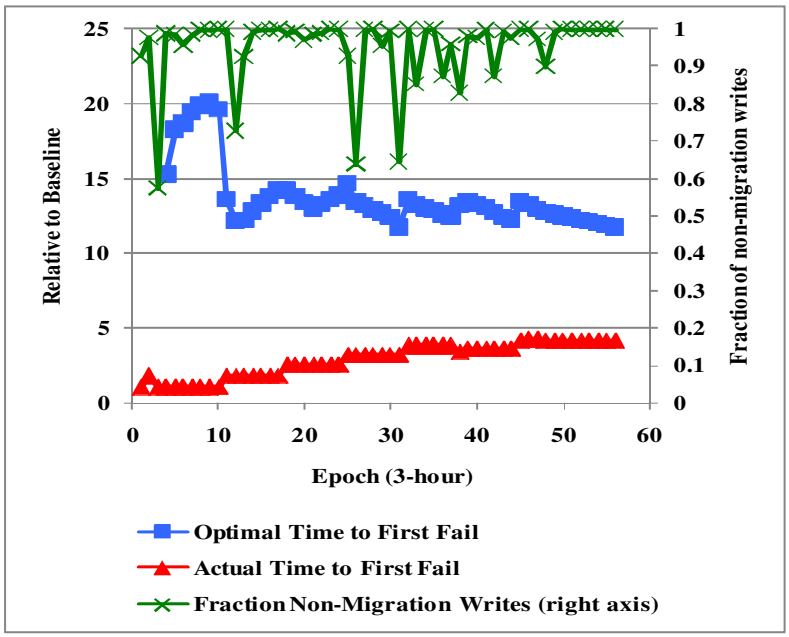


Figure 4.13: (LFPS+POOL) run-time wear-leveling policy. The x-axis represents elapsed time in 3-hour intervals. The y-axis shows improvement over baseline.

efficiency after receiving migrated blocks. On average, distributed wear-leveling improves the write efficiency by 20%, which is also much more uniform between SSDs. Because every SSD sees the same amount of written data, this average represents the true average improvement across all drives. However, because the efficiency of the most write-heavy volume after re-mapping is proj_2 and its write efficiency decreases, the overall TTFF improvement decreases to 3.9x from 4.2x with Flash.

As an experiment, we also examined how a possible PCRAM or other byte-addressable SSD using Start-Gap wear leveling [51] might perform, and show potential synergistic benefits from applying distributed wear-leveling as a first re-mapping step. This analysis is presented in Appendix A.

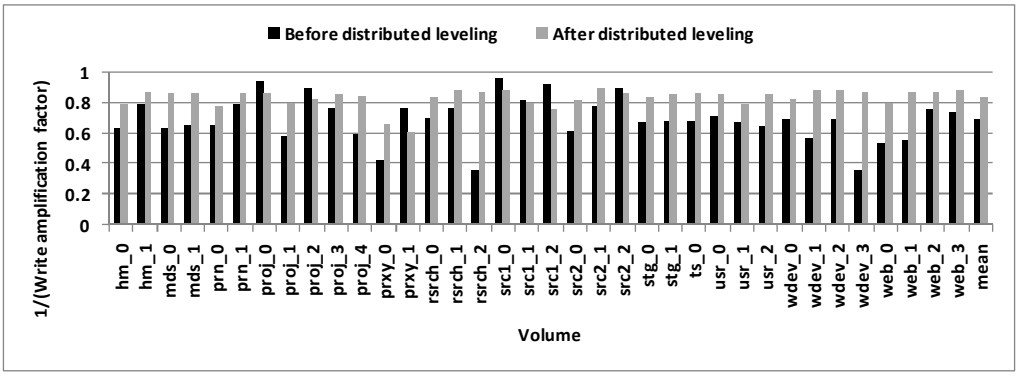


Figure 4.14: Local SSD wear-leveling write efficiency.

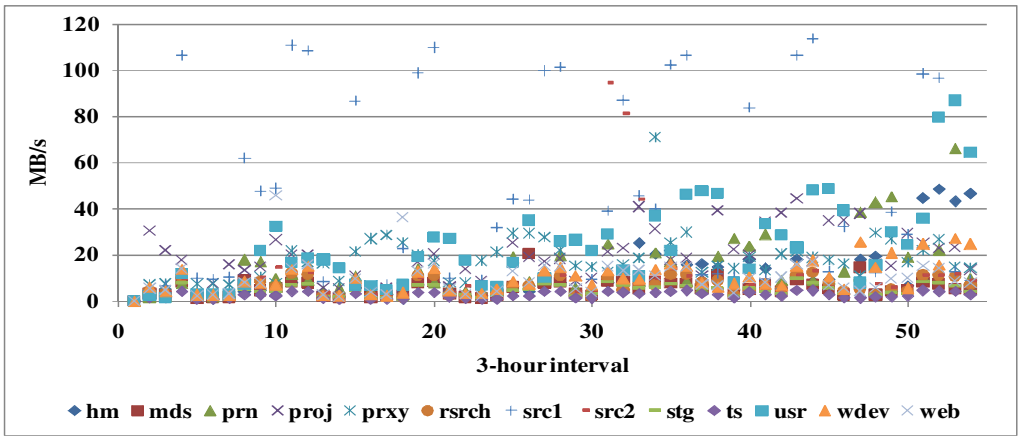


Figure 4.15: Peak network bandwidth

Performance impact

In this section we examine the average latency as reported by DiskSim and peak network bandwidth, to determine any slowdown due to a network bottleneck. For the SSDs of the baseline system, the average latency varied from 0.21 to 2.63 ms with a mean of 0.52 ms. After distributed wear-leveling, this changes to 0.20 to 0.52 ms with an average of 0.30 ms. The decreased range of latencies is indicative of the greater spread of requested addresses and less address contention on individual SSDs. Figure 4.15 shows the peak network bandwidth reached during each 3-hour interval.

Using the (LFPS+POOL) policy the bandwidth in any 1-second interval never exceeds 1 Gbit/s on any servers Ethernet connection. This implies that there will be little slowdown introduced due to a bandwidth bottleneck. The high peak values for src1 indicate the periodic daily burst of activity causing remote accesses. There is a smaller daily fluctuation in the other servers due to the fact that accesses originating from several servers are now spread across remote drives that were previously under-utilized. Overall however, the average SSD utilization is very low for these traces (see Figure 4.5). More intensive workloads may suffer from a loss of data locality, as files are broken into 64 KB chunks and separated across the network. Sequential accesses can provide better performance as they can maximize pipelined bus bandwidth and easily interleave across internal Flash banks, based on their low-order address bits. However, this may be mitigated by using larger contiguous migration block sizes, or more advanced algorithms that retain sequentially-accessed blocks or entire files on the same SSD or on the same server.

4.3.6 Conclusion

This work has motivated the consideration of access patterns across solid-state disks as well as within them, when trying to improve overall lifetime. This should reduce management costs and increase predictability of wear-out failure, reducing the need for replication and data migration. Our distributed block re-mapping scheme achieved a 3.9x improvement in TTFF with a Flash SSD for a small migration overhead. Initial FTL simulations involving write counts equal to twice the SSD capacity indicated a further decrease internal Flash SSD erases by 20%, although identifying the cause will require longer simulations to fully evaluate. It is clear that the distributed wear-leveling policies to use are highly workload and system-dependent. For example, our system did not have the flexibility of a MapReduce

infrastructure where small units of data and the code can be migrated quickly to arbitrary server locations. The "traditional" data center we examined could have benefited from say, a web proxy distributed across multiple servers, which could reduce network traffic relative to our centralized server. In conclusion, as industry is pushing towards cheaper, more energy efficient, high-performance cloud computing infrastructures, non-volatile memories will become indispensible system elements. With the more cost-effective multi-level cell devices suffering decreased endurance, lifetime management and fault tolerance must be addressed system-wide.

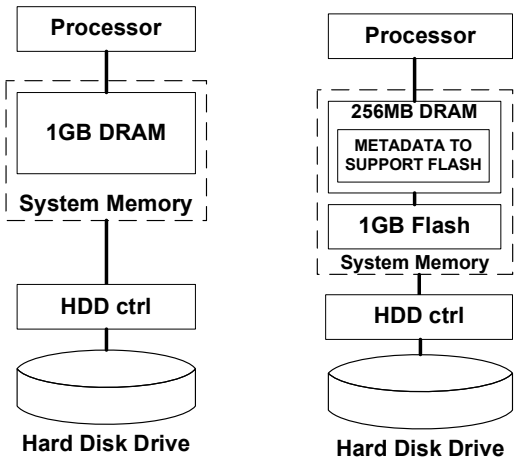
4.4 NAND Flash based Disk Cache Reliability

Flash based disk caches have been proposed as a means to improve performance and reduce energy consumption of hard-disk based systems [7]. Here we propose extensions to this work to improve the reliability of NAND Flash [4, 2] and PCRAM [8] based disk caching. The techniques were evaluated against Web server and database workloads with significant energy efficiency improvements. [1]

The first technique proposed is variable-strength ECC. Conventionally, a random error correcting code such as BCH would be used to correct faulty bits in a block of Flash data. We observed that because emerging errors usually occur one at a time, and that BCH decode latency is roughly proportional to number of corrected bits (Figure 4.17), the best latency can be obtained by using the minimum strength code. The code strength has to be one greater than the current number of errors.

The second technique switches between SLC and MLC modes for Flash in order to either minimize latency or in response to emerging errors, to increase lifetime. Figure 4.18 illustrates that as more area is allocated to the Flash cache, the latency-optimal

[1] The author of this thesis contributed the concepts of variable strength BCH coding and dynamic MLC to SLC switching in [4], and performed the evaluation for PCRAM [8]. Taeho Kgil developed the system-level simulation infrastructure and evaluation for NAND Flash, as well as simulating BCH decode latency (Figure 4.17).



(a) before (b) after

Figure 4.16: NAND Flash based disk cache block diagram

ratio of SLC to MLC pages varies.

When both techniques are applied in the system with appropriate heuristics to choose when to apply them, we obtained up to 20x in improved Flash lifetime (Figure 4.19(b)).

Because of the inherent scalability limits of Flash-based storage, we compared the performance of a PCRAM-based secondary disk cache with SLC and MLC NAND Flash [8]. Our intention was to see the effects of PCRAMs lower latency and removing the need to erase blocks of pages. We applied similar setups to those used for our Flash studies. It should be noted that these preliminary simulations were again scaled down relative to the capacity of real servers because of resource limitations on the host system. However, the simulations are sufficient to highlight the trade-offs between the different memory technologies. Figure 4.20 compares the relative network bandwidth achieved in servers using Flash or PCRAM as a secondary disk cache. We examined dbt2, the most disk intensive benchmark. This is not an in-memory database so there are significant numbers of file accesses at run-time to stress

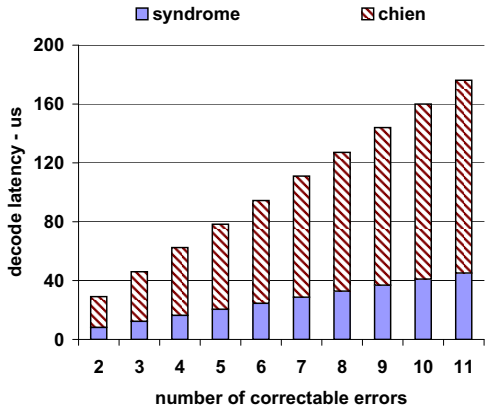
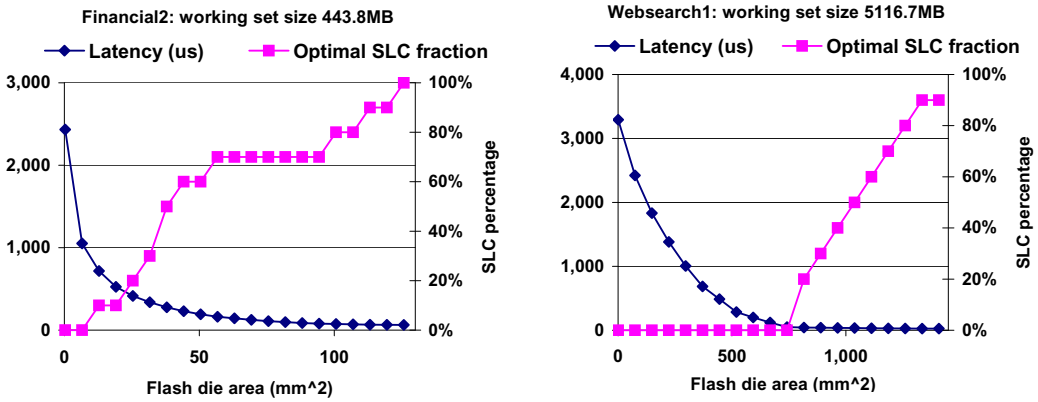


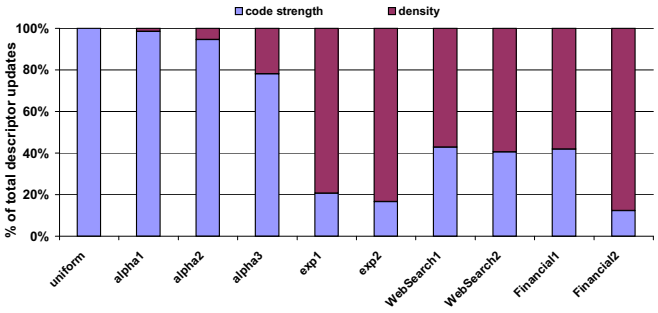
Figure 4.17: BCH decoding latency for varying numbers of correctable bits. Source: Taeho Kgil [4]



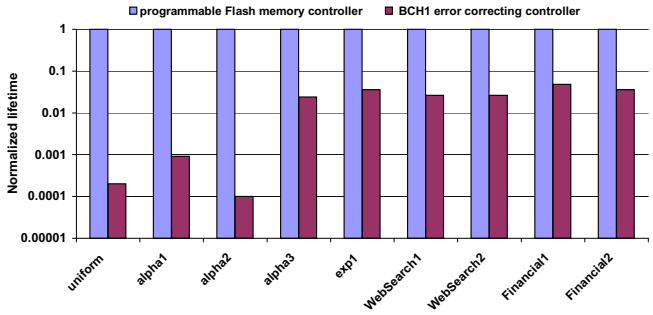
(a) Financial2 (b) WebSearch1

Figure 4.18: Optimal access latency and SLC/MLC partition for various multimode MLC Flash sizes

the storage subsystem. Doubling the DRAM capacity (which includes the primary page cache) increases performance by around 35% when combined with an SLC Flash secondary cache. At any particular size of main memory, MLC Flash performs slightly worse than SLC Flash as expected, and PCRAM performs up to 65% better than SLC Flash. The increased performance due to PCRAM also translates to a very significant total energy saving. Assuming that the server is in a low power state during idle periods, completing the work faster means that less energy is consumed



(a) Breakdown of page reconfiguration events



(b) Normalized expected lifetime for a given access rate and the point of total Flash failure

Figure 4.19: Relative frequency of lifetime mitigation events and the corresponding improvements in lifetime. Source: Taeho Kgil [4]

by the power supply, processors, memory and disk drives in the system. It could be argued from this data that simply increasing the amount of system DRAM is a way to increase performance. This would be true if cost, density and power consumption were not constraints. In a server requiring multiple Gigabytes of main memory, the much lower cost per Gigabyte and greater density of NAND Flash (presently almost 4x) mean that it is cost effective and a lower power solution to construct main memory with more Flash memory than DRAM as our studies show. Furthermore, if PCRAM continues to scale as predicted, it will then become a candidate to replace or supplement Flash in our disk cache usage model.

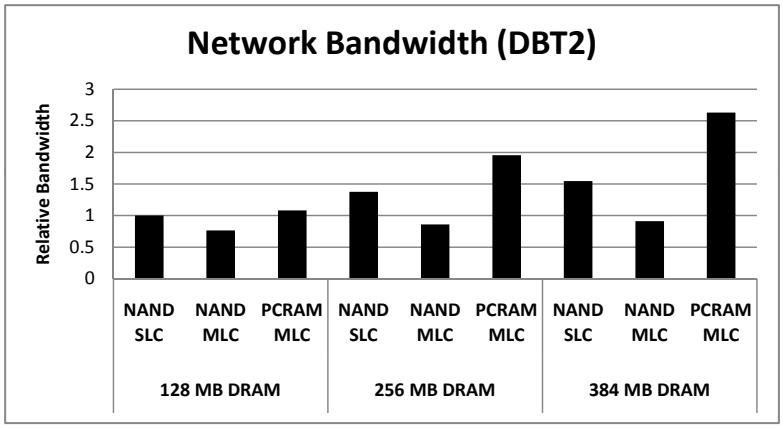


Figure 4.20: Network bandwidth as a function of DRAM size (including primary disk cache) and secondary disk cache technology. The system was provisioned with 1GB of secondary disk cache.

CHAPTER V

On-Chip Cache Memory Reliability for Energy Reduction

The CPU has been shown to be a major power contributor in data centers (Section 3.4). Improving the energy efficiency of this component is therefore important, but voltage reduction is limited by certain outlying logic and memory element characteristics due to process variation. Timing fault-tolerance such as Razor [9] addresses logic errors. In this section, we address the on-chip SRAM cache errors which occur during aggressive CPU voltage scaling. Being able to reduce the voltage down to a level tolerable by logic circuits has proven difficult because of process variation-induced SRAM cell failures [10]. We approach this problem by comparing different cache fault-tolerance techniques to determine which will be most effective under voltage scaling, or when on-chip memory cell defect probabilities exceed those of current technologies due to process scaling. Our most significant finding from this study is that the devices in on-chip memory cells cannot be scaled at the same rate as devices in logic circuits due to the increasing number of erroneous memory cells with voltage scaling, requiring strong fault-tolerance techniques. Second, we propose a technique [14, 15, 16] to maximize the available cache capacity at a given voltage, minimizing performance impacts. Our scheme, called Block Grouping, works by merging non-faulty segments from a number of faulty cache lines to form a single working cache

line. The scheme performs better than triple-modular redundancy (TMR) at high error rates. We also estimate up to 28% energy savings at low voltage, relative to a contemporary fault-tolerance scheme [92].[1]

5.1 Introduction

As microarchitects demand larger on-chip caches for higher performance, continuous device scaling has provided improved memory density for multi-megabyte upper-level on-chip caches at a reasonable die cost. However, the device scaling comes at a price. The reduced device feature size causes exponentially increasing subthreshold and gate-leakage power problems in on- chip caches fabricated with sub-90 nm process technology resulting in more static power consumption [122]. Furthermore, process parameter variations, e.g. random dopant fluctuations causing threshold voltage variations or mismatches across the devices used in a on-chip memory cell and more oxide defects in devices during the manufacturing process have worsened yield problems in on-chip caches manufactured with sub-90 nm technology [123]. To overcome low yield problems caused by scaling device sizes and integrating more on-chip memory cells, there have been several proposed techniques. One is to implement redundant memory columns; there are one or two redundant columns per memory sub-bank or sub-array. If a defective cell is found during the manufacturing test, the entire column containing the defective cell is replaced with a redundant column. This wastes many memory cells to fix one defective cell and requires fuses to replace the column containing the defective cell with the redundant column. The second technique is to use error correction codes (ECC). Currently, a single error correction (SEC) and double error detection (DED) technique is used. Even though this can fix one defective cell per sub-array row, the memory array is made more

[1] This work was performed at the University of Michigan and funded by ARM Ltd.

vulnerable to soft errors since the correction capability of the code has been used up by fixing defective memory cells. The third technique is to disable a part of the on-chip cache memory array resulting in a smaller size. An example is the Intel Celeron processor. It is very similar to the Pentium processor, but it has the entire or half of the L2 cache disabled as a result of memory sub-arrays containing defective cells that could not be fixed using the redundant columns in the disabled part of the on-chip cache memory block. All these techniques are only effective when there are a small number of defective cells in the on-chip cache. However, the number of defective cells in large on-chip caches will rise if we want to continue scaling memory cell size along with technology scaling. Hard-wired redundancy is becoming a less attractive option due to limited area available for spare memory cells. In addition, it will no longer be possible to find a single set of cache blocks which consistently fail at each operating point [123]. Prior work suggests that avoiding defective cache memory cells at the block level can be very cost-effective in terms of both area and performance overheads. However, these studies were performed with either outdated cache hierarchies and benchmarks [87] or for direct-mapped caches only [92]. Under aggressive voltage scaling and on-chip memory cell sizing, we show that higher defect rates with existing fault-tolerance schemes result in significant processor performance degradation. A dynamic voltage scaling (DVS) environment adds to the complexity of working with on-chip caches containing unpredictable defective memory cells; as the operating voltage changes, so does the number of defective cells. In this paper we begin with an analysis of L2 cache activity in a modern processor architecture based on the Intel Pentium 4. Emphasis is placed on L2 caches because of their widespread use and relatively large area compared to L1 (L1 caches are also relevant, and the error analysis within this paper can also be applied to other levels besides L2). We show the impacts of defective cache blocks on performance and compare ways of

addressing this problem. The major contributions of this paper are;

- Trade-off analysis between performance and area for different cell sizes and fault-tolerance techniques.
- A novel cache block grouping scheme for good performance at higher fault probabilities.

The rest of the section is organized as follows. Section 5.2 explains in detail the basic fault-tolerance scheme upon which this work is based. Section 5.3 explains the problems encountered with on-chip cache memory reliability in new processes and its impact on performance of set-associative caches when defects are present. Based upon this analysis, we show existing and proposed techniques of reducing performance impacts in the presence of defects in on-chip caches in Section 5.4. The techniques are compared in Section 5.5, and concluding remarks are presented in Section 5.6.

5.2 Background

In this section, we explain the baseline fault-tolerance scheme. For related work, please see Section 2.3.2.

Figure 5.1 is the same figure as Figure 7 in [92] and illustrates the fault tolerance scheme presented in that work. It is based on a direct-mapped cache consisting of lines organized in rows and columns. Rows are addressed as usual using part of the incoming address. However, the column address may be re-mapped to avoid a known faulty block. This is achieved by performing a look-up in the "config storage" which contains a map of defective block locations. In this instance, there is one bit per block (hence one bit implementation, or OBI) which is set to 1 if the corresponding block is defective. When the cache is accessed, the controller uses the OBI to select

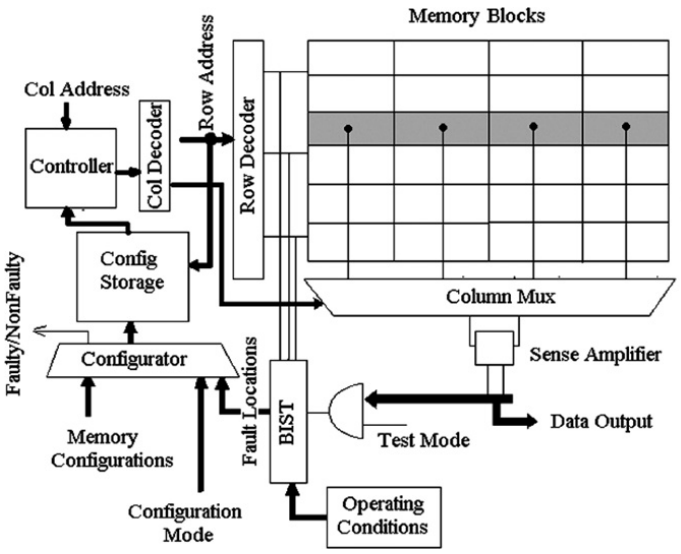


Figure 5.1: The one bit implementation (OBI) technique

a non-defective column to store data to, using a fixed mapping. Additional bits are required in the tag to indicate the column in which data is stored. This prevents faulty blocks from being read.

5.3 Impact of on-chip cache failure rate on processor performance

5.3.1 On-chip cache device scaling and failure rate

Currently, the feature sizes (e.g. 45 nm) are so small that it is very difficult to control the uniformity of device parameters across dies and wafers. In particular, smaller devices that are extensively used in on-chip cache memory cells are increasingly sensitive to parameter variations. Furthermore, dynamic voltage scaling is very commonly used to reduce power consumption of the processors and their on-chip caches should be able to operate at the same voltage as the processor core, to avoid adding overhead to allow separate voltage domains. However, as the supply voltage of on- chip cache memory cells decreases, we find more memory cells failing due to increased sensitivity to process variation at lower supply voltage. Failure types

are read failures (flipping of the stored state during read operations), write failures (inability to write a state during write operations), access time failures (an in- crease in the access time of the cell resulting in the violation of the delay requirement), and/or retention failure (losing the stored state in standby mode) [124, 125]. As a result, the lowest operating voltage (called Vccmin) of processors employing DVS is usually determined by the lowest supply voltage that keeps all on-chip memory cells functional. However, lower Vccmin is desirable if static and dynamic power consumption are to be reduced. The best way to improve Vccmin is to increase memory cell size to reduce the process variation sensitivity of the memory cells. However, a larger memory cell size increases the area occupied by on-chip caches resulting in increasing die cost or decreasing the on-chip cache size at a given die size (e.g. 12 MB instead of 16 MB for a L3 cache). Hence, the memory cell size must be balanced to give proper Vccmin, yield, and on-chip cache capacity.

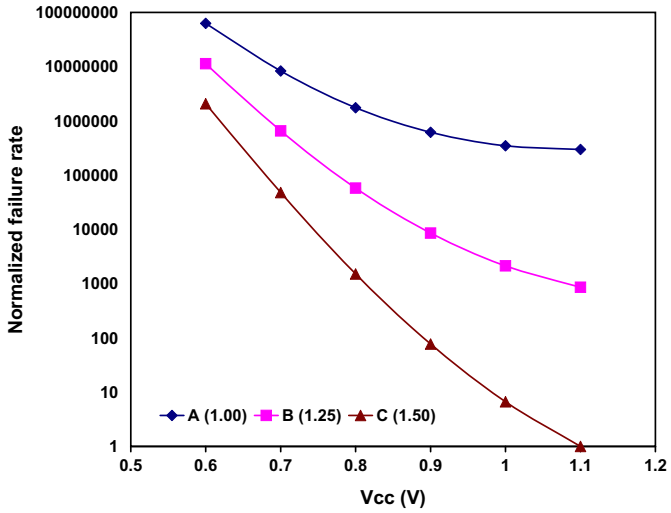


Figure 5.2: Normalized cell failure rates as a function of voltage for three different memory cells

Figure 5.2 shows normalized on-chip memory cell failure rates for 3 different memory cell sizes (A, B and C with relative areas of 1, 1.25, and 1.5, respectively) as a

function of memory cell supply voltage. Note that depending on how we tune the sizes of the six transistors in the memory cell, the result varies significantly. They are obtained using Monte-Carlo simulations on memory cells designed with a 45 nm technology and process parameter variations corresponding to the technology. The failure rate we assume is significantly higher than the data presented in other work [92], however the increased failure rates can be expected in future smaller semiconductor process technology (e.g., 32 nm technology). As shown in Figure 5.2, as either voltage or cell size decreases, the failure rate starts to increase exponentially. In other words, a larger cell can achieve a much lower Vccmin at the same failure rate. Finally, defect rate is proportional to die size. Hence, when we integrate more on-chip memory cells on a die along with device scaling, there will be a much greater chance that some memory cells contain defects and fail during post-manufacturing tests resulting in poor yield. The next section examines the relationship between on-chip cache memory cell failure and performance impact to determine the number of tolerable, non-corrected faults.

5.3.2 Performance impact of set-associative cache defects

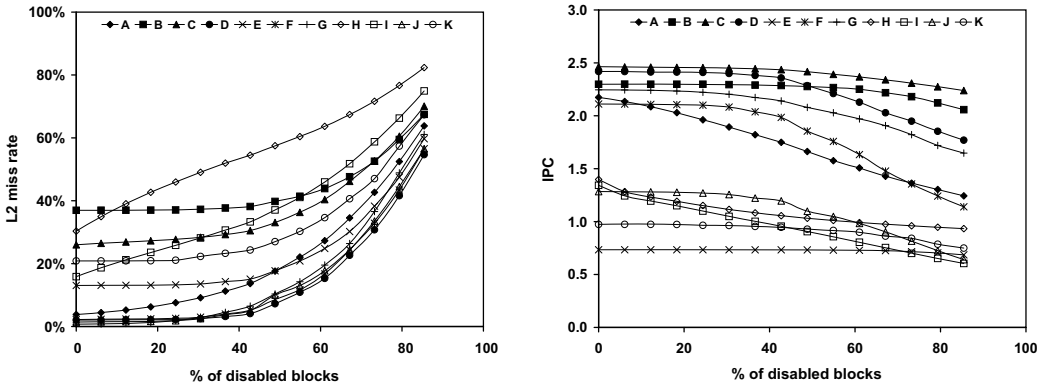
The performance impact of on-chip cache memory cell failure partly depends upon the fault-tolerance technique employed. While some impact arises from increased miss rates as faulty regions are disabled, others incur a performance penalty when error-correcting codes are decoded. We perform an analysis of block level fault-tolerance schemes. All data was obtained using the M5 simulator [126]. The simulator was configured to represent a modern out-of-order pipeline with similar specifications to a Pentium 4 (Table 5.1). The memory latency is relatively low, although this will not significantly affect L2 miss rates.

Figures 5.3(a) and 5.3(b) present the impact of defective blocks in L2 (each con-

Parameter	Value
Clock frequency	2 GHz
Pipeline width	4
Branch prediction/BTB	Hybrid 4-way, 2K entries
ROB/LSQ size	196/32 entries
INT ALUs/multi-divs/mem ports	6/2/4
FP ALUs/multi-divs	4/2
Functional unit latencies	INT: mul 3, div 20, all others 1, FP: adder 2, mul 4, div 12, sqrt 24
IL1 cache	16 kB, 2-way, 64B blocks, 1-cycle lat.
DL1 cache	16 kB, 2-way, 64B blocks, 3-cycle lat.
L2 cache	1 MB, 8-way, 64B blocks, 19-cycle lat.
Memory bus/latency	16 bytes with 6-cycle lat./100 cycles

Table 5.1: M5 CPU Configuration

taining 1 or more defective cells) on miss rate and instructions per cycle (IPC), respectively. We use the SPLASH-2 benchmark suite [127] as a workload representative of both memory and compute-intensive applications. In these graphs, defective block locations are allocated randomly, but consistently between benchmarks. The LRU scheme was modified so that defective (non-correctable) blocks are not considered for replacement. If all ways in a set are defective, accesses to that set bypass the cache and are forwarded to the next level of the memory hierarchy. This has the effect of reducing the number of available ways in a cache set, while using the standard tag matching mechanism to determine hits or misses within the remaining good blocks, which is identical to [128]. The data in Figure 5.3(a) and 5.3(b) confirm the previous study [87] in that high block failure rates are required before there is any significant performance penalty. With this in mind, the next section compares existing and our fault-tolerance techniques for their performance and area at significant failure rates.



(a) (b)

Figure 5.3: L2 miss rate in (a) and IPC in (b) as a function of number of randomly disabled blocks. A-Cholesky, B-FFT, C-LUContig, D-LUNoncontig, E-Radix, F-Barnes, G-FMM, H-OceanContig, I-OceanNoncontig, J-Raytrace, and KWaterNSquared, respectively.

5.4 Comparison of fault-tolerance techniques

The one bit implementation (OBI) mapping table model [92] is effective for low failure rates, but for higher rates we show that it rapidly becomes ineffective. We aim to allow more faults with stronger error correction, and observe the trade-off with area cost. In this section we derive an analytical model representing the fraction of good blocks remaining in the cache at different cell failure probabilities (and sizes). We consider tag bits as additional bits contained in each block. We compared several cache fault-tolerance schemes in order to determine their area efficiency at different error rates (voltages and cell sizes). The model used represents the fraction of fault-free blocks available in the cache, denoted by F_{avail}. As a minimum, we decided to first apply the OBI scheme, followed by other error correction. From a storage standpoint, OBI provides the minimum data needed to identify where faulty blocks are, for avoidance. Since it has already been proven superior to redundant rows and SECDED ECC, all of our models build on the OBI baseline. OBI does not affect the cache access time and has minimum effect on processor performance [92].

Throughout the paper we refer to p_{fault} as the probability of failure of a single on-chip memory cell. Cell failures are assumed to be independent. As a first approximation this assumption is valid and has been widely used in other published cache-error related work ([1,5,8]). To improve the effectiveness of schemes requiring an additional storage table (e.g. OBI) which must contain correct bits, we introduce a factor OBIff. This factor reduces the bit failure probability, representing larger size or higher voltage on-chip cache memory cells (see Figure 5.2) used specifically for that table. We refer to this as "guaranteed correct" storage because one fault in this table could lead to bad cells being accessed. Using large cells is viable as long as the table does not contain too much data. In addition, de- lay does not vary significantly with cell size. Basic storage refers to the cells used in the cache itself. The fault-tolerance schemes (Table 5.2) were chosen from a range of candidates, most of which are widely used today. We only model storage-related reliability while logic reliability is beyond the scope of this paper. The following sections explain the fault model and storage overhead of each scheme.

Each cache consists of M sets and N ways where each block contains B bits (including tag bits). We also derive an area efficiency E_{area} which takes into account the probability of failure of the guaranteed correct storage which includes the OBI table and any additional bits added by a scheme which must be correct for the cache to operate reliably. The fraction of available blocks is divided by die area consumed by all SRAM cells, then scaled by the probability of the guaranteed correct storage containing no faults Eq. (1).

$$E_{area} = \frac{F_{avail}}{area}.p_{non\_faulty\_GC} \tag{5.1}$$

The $p_{non\_faulty\_GC}$ value is the probability that the guaranteed correct cells are

Scheme	Storage (MB)	Description
OBI	1.002	One bit implementation [92] indicates a faulty block with a single bit
Hamming SEC	1.020	Single-error correcting (SEC) hamming code
BCH DEC	1.037	Double-error correcting (DEC) Bose-Chaudhury-Hocquenghem
log(B)	1.022	Bad block table contains index of one bad bit and a spare cell to store the value of that faulty bit. If there is more than one defective bit, the block is disabled
Triple-modular redundancy (TMR)	1.002	Faulty blocks are combined in groups of 3 inside the 1 MB cache, with a majority vote on each bit
Block Grouping (GRP2)	Up to 1.528	Pairs of faulty blocks are combined to form single good blocks. A paired block is 'good' if there is only one faulty block segment for each corresponding pair of segments.

Table 5.2: Candidate cache fault-tolerance schemes and their storage overheads for an 1MB cache

fault-free, as a function of the probability of the large-size cell failure p_{fault}_GC and the number of guaranteed correct bits (GC_bits).

$$p_{non\_faulty\_GC} = (1 - p_{fault\_GC})^{GC\_bits} \tag{5.2}$$

In all of these schemes,

$$p_{fault\_GC} = OBI_{ff}.p_{fault} = 10^{-5}.p_{fault} \tag{5.3}$$

5.4.1 Existing fault-tolerance schemes

OBI

The "one bit implementation" consists of a table of bits, one per block, indicating whether or not each block contains 1 or more faulty bits. All of our schemes incur

this storage overhead, because we use an OBI to indicate whether a block can be corrected or is unusable and cannot be accessed.

Storage overhead

$GC\_bits = M.N$ bits (guaranteed correct storage).

Fault model

The probability of a faulty bit is p_{fault}. The probability of a non-faulty block is the likelihood of every bit being fault-free in that block. We assume that this probability represents the fraction of non-faulty cache blocks, as follows;

$$F_{avail} = (1 - p_{fault})^B \tag{5.4}$$

SEC

Single error-correcting (SEC) codes were included due to their low overhead and widespread use in existing devices.

Storage Overhead

$$b = \lceil log_2(B) \rceil \tag{5.5}$$

where b is the number of added ECC bits per cache block (basic storage).

Fault model

The model is modified to account for the increased block size (for check bit storage) and the ability to correct 0 or 1 bits.

$$F_{avail} = (1 - p_{fault})^{B+b} + \binom{B+b}{1} . p_{fault} . (1 - p_{fault})^{B+b-1} \tag{5.6}$$

BCH double error correction (DEC)

The Bose-Chaudhuri-Hocquenghem (BCH) error-correcting code was selected as a candidate DEC scheme. Alternatives such as Reed-Solomon and Golay codes are mentioned in [129]. BCH was chosen because of its low storage overhead. However, in practise a less compute-intensive code can be used depending on the sensitivity of performance on L2 latency. We modeled a DEC BCH code storage overhead (with minimum distance $d_{min} = 5$) based upon the equations in [130].

Storage overhead

$$b = 2.\lceil log_2(B) \rceil \tag{5.7}$$

where b is the number of added ECC bits per cache block (basic storage).

Fault model

The probability of a faulty block is modified to account for the extra check bit storage and the ability to correct 2 bits.

$$\begin{aligned} F_{avail} = (1 - p_{fault})^{B+b} &+ \binom{B+b}{1} . p_{fault} . (1 - p_{fault})^{B+b-1} \\ &+ \binom{B+b}{2} . p_{fault}^2 . (1 - p_{fault})^{B+b-2} \end{aligned}$$

Log(B)

The log(B) scheme is an alternative single error-correcting scheme. A table stores the index of one faulty cell location per block, along with an additional bit to hold the correct state of that cell. This is equivalent to the distant repair scheme of [131] using one spare unit.

Storage overhead

$$b = \lceil log_2(B) \rceil + 1 \tag{5.8}$$

These b bits per block are held in guaranteed correct storage.

Fault model

The fraction of available blocks is identical to that of SEC except that the additional bits are in guaranteed correct storage.

$$F_{avail} = (1 - p_{fault})^B + \binom{B}{1}.p_{fault}.(1 - p_{fault})^{B-1} \tag{5.9}$$

Triple-modular redundancy (TMR)

Our triple modular redundancy implementation assigns faulty blocks to groups of three blocks with a majority vote on every bit (0 or 1 errors can be corrected per bit position). At most, 1/3 of logical bits can be recovered from the physical bits which are combined for a majority vote. In a hardware implementation, the bit comparison for the majority vote is performed at the final cache output stage. Therefore logic overhead will be small.

Storage overhead

No additional storage is allocated to identify which blocks are combined for TMR. For this typical case analysis, we assume that faulty blocks are combined with other arbitrarily located faulty blocks.

Fault model. We first consider each bit index as 3 bits which must have 0 or 1 faults to be corrected. This applies for all B bit indexes. However, because we only combine known faulty blocks after determining fault-free blocks, none of the three blocks are ever error-free and this probability (p_{good}) is subtracted from the main expression. The probability of a non-faulty block is $p_{nfb} = (1 - p_{fault})^B$.

$$p_{good} = \binom{3}{1}.p_{nfb}.\left[(1-p_{fault}^2)+\binom{2}{1}.(1-p_{fault}).p_{fault}\right]^B - 3.(1-p_{nfb}).(p_{nfb})^2 - 2.(p_{nfb})^3$$

(5.10)

$$fraction\_repaired_{TMR} = \frac{1}{3}.\left(\left(\binom{3}{1}.p_{fault}.(1-p_{fault})^2+(1-p_{fault}^3)\right)^B - p_{good}\right)$$

(5.11)

$$F_{avail} = p_{nfb} + (1-p_{nfb}).fraction\_repaired_{TMR}$$

5.4.2 Proposed fault-tolerance scheme

Block grouping

For high error rates, we propose a new scheme (Figure 5.4). Faulty physical blocks are grouped together (in groups of size G) to form a new, fully working logical block. In the rest of our analysis, we assume pairs (G = 2). Using larger groups is beneficial at extremely high error rates, but the analysis is beyond the scope of this paper. The concept is a similar to [8] except that knowledge of failure polarities is not required. Compatible blocks have up to one faulty bit between them, at every corresponding bit index. This means that an additional selector bit, which is known to be correct, can specify which bit contains a good value when reading data. A grouping table is accessed as an additional step before a cache access, to identify the paired block. To read a grouped block, all blocks in the group are read. The selector bits then indicate which block in the pair contains good data, at each bit index. A single logical block is then returned to the processor. To write a grouped block, the same value is written to every component block.

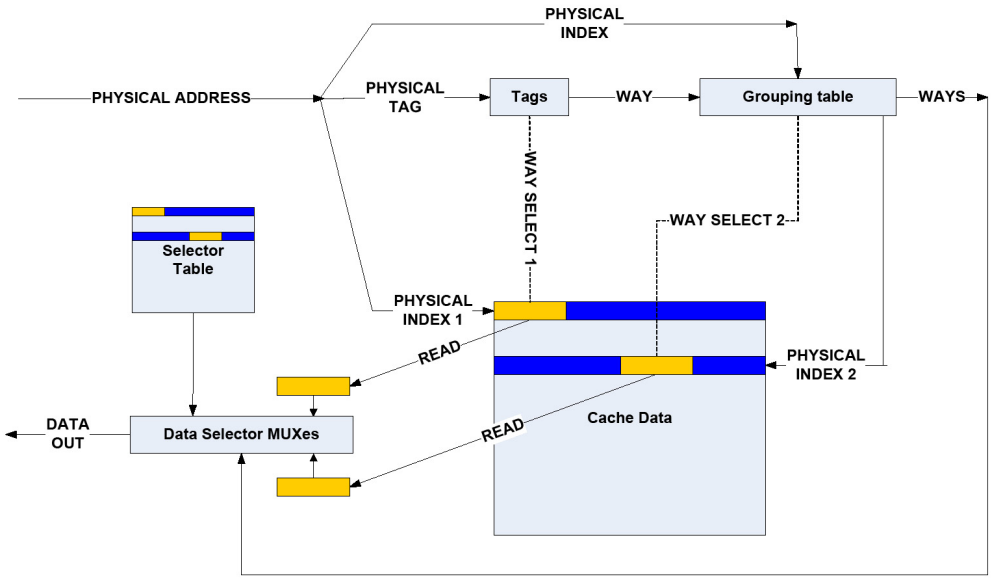


Figure 5.4: The proposed block grouping scheme

Grouping table

This table is used to look-up the location of the other block in a group. If there is more freedom to combine faulty blocks that are compatible, more blocks can be recovered. They can be physically adjacent, in the same set or in any location inside the cache (depending on the desired complexity of block selection hardware). Each alternative has performance tradeoffs, discussed later.

Selector bit table

Selection of the block which the data bit is read from is performed using a table of selector bits. These are stored in guaranteed correct cells, and can either cover every bit index in the block or a number of adjacent bits (e.g. two data bits per selector in Figure 5.5). In this example, each selector bit indicates which block should be

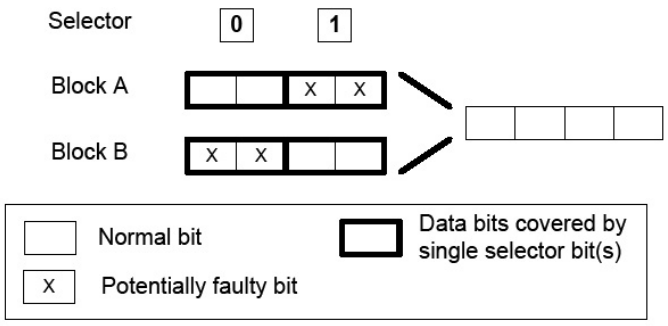


Figure 5.5: Example of selector usage (2 bits/selector).

accessed for every pair of adjacent bits. Using few- er selector bits reduces the number of defects which can be tolerated but reduces storage overhead. For example, a single selector bit covering two adjacent data bits cannot handle the case where there is a fault in both blocks at that position. Later, we discuss off-chip caching of selector bits to reduce the die area of on-chip SRAM.

Table configuration

The tables are programmed during system start-up. Self-test routines determine whether cache cells are operating reliably at each voltage and frequency point, and the map is stored in main memory or on disk. When performance settings change, the cache is flushed and a new selector table loaded. These tables could also be hard-wired at manufacturing test.

Storage overhead

We call the first block to be accessed the "primary" block, and its paired compatible block the secondary block. For the grouping table, we first consider the most storage- intensive scenario where blocks are paired anywhere in the cache.

The grouping table has a number of entries equal to the number of blocks in the cache. Each entry stores the set and way index of a compatible block, to be looked up on a read access. The equation below assumes that there is an entry pointing to

another block for every block position in the cache.

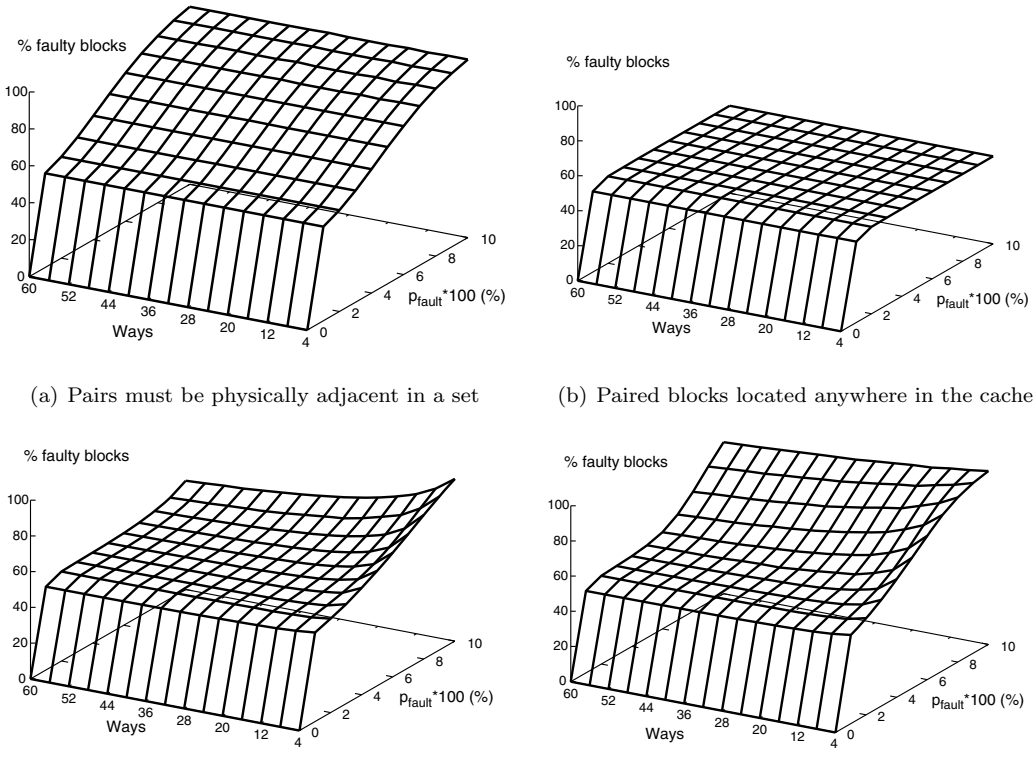
$$group\_table\_size = M.N.log_2(M.N) \tag{5.12}$$

As a lower-cost alternative, the storage requirement for pairs limited to the same set is given below. When implemented as an asso- ciative look-up, half of the blocks in a set have a pointer to another block in the same set.

$$group\_table\_size = \frac{N}{2}.log_2(N).M \tag{5.13}$$

Instead of using a grouping table, one could use the existing tag matching mechanism to simultaneously hit multiple blocks of the same group since their address tags are identical. It requires that the group resides in a single set so that address indexes are identical for each block. A banked cache design where ways are in different banks would allow fast parallel access to a group of blocks. A sequential access model is still feasible however, because an extra cycle to look-up a secondary block does not significantly impact performance for low-level caches (e.g. L2). Another, less effective zero-overhead alternative is to use a fixed grouping, for example, pairing together adjacent blocks.

The error-correcting ability of each variant is shown in Figure 5.6. The results were derived from simulation, and pairs were formed using a greedy algorithm that allocates each faulty block with the next free compatible faulty block in sequential order. An optimal grouping will be even more effective. In paper [102] they determined that the optimal block grouping algorithm is an instance of the *maximum matching problem*, which can be solved in polynomial time with optimal algorithms. However, they went on to use the same greedy algorithm that we propose and demonstrate that



(a) Pairs must be physically adjacent in a set

(b) Paired blocks located anywhere in the cache

(c) Pairs must be in same set, 1 logical bit per selector bit

(d) Pairs must be in same set, 2 logical bits per selector bit

Figure 5.6: Percentage of faulty blocks using block pairing (G = 2) for different group restrictions. Block size = 32 bytes.

it gives a near-optimal number of pairings, dropping off shortly before the optimal algorithm as failed bit count increases. Adjacent pairing is least effective and is not improved with associativity (Figure 5.6(a)). It is clear that arbitrary pairing (Figure 5.6(b)) is most effective at high error rates, although the per-set limitation (Figure 5.6(c)) can be almost as effective. Increased associativity helps in this case by providing more pairing alternatives and can be seen in processors such as Niagara, with a 12-way L2 cache [132]. The final plot (Figure 5.6(d)) represents blocks restricted to a set, and one selector bit is used for every two bits in the block. Fault tolerance is obviously reduced but there are now half as many selector bits. The number of selector bits (used to choose one good bit from a group of G bits at each bit index)

is the logarithm of the number of bits in the group. There are B selectors per cache block, and (MN)/ G logical blocks after grouping.

$$selector\_bits = \frac{M.N}{G}.(log_2(G).B) \tag{5.14}$$

where G is the number of blocks in each group.

(5.15)

$$GC\_bits = M.N + group\_table\_size + selector\_bits(\text{guaranteed correct storage}) \tag{5.15}$$

Fault model

The parameter G can be varied, but all analysis in this work uses block pairs (G = 2).

$$p_{good} = p_{nfb}^2 + \binom{2}{1}.p_{nfb}.(1 - p_{nfb}) \tag{5.16}$$

$$fraction\_repaired_{GRP2} = \frac{1}{2}.((1 - p_{fault}^2)^B - p_{good}) \tag{5.17}$$

$$F_{avail} = p_{nfb} + (1 - p_{nfb}).fraction\_repaired_{GRP2} \tag{5.18}$$

Each logical bit is formed from two bits. A set of two faulty physical blocks are compatible and can be recovered into a single logical block when;

- At most one of the grouped bits at each index are faulty, and
- This is true at every bit index in the block of B bits.

Note that we make an adjustment p_{good} to remove the impossible cases where any block contains no faults, as per TMR.

Selector bit caching for block grouping

Compared with the other schemes, block grouping has strong fault tolerance characteristics but a potentially large storage overhead. By caching the working set of selector bits in on-chip SRAM and keeping less frequently used bits off-chip, area overheads can be reduced without significant performance impact. Although this appears only to move the reliability problem elsewhere, there are several benefits;

- Write accesses to the off-chip storage are infrequent (for example, at manufacturing test, or if BIST is re-run when there is a significant temperature change). Therefore the table could be stored in FLASH memory, for example. Off-chip DRAM is another possibility and in both cases there will be a reduction in total area and energy consumption. Although off-chip bus traffic when retrieving cached selector bits requires more energy than an on-chip access, this is an infrequent event.
- Moving the storage off-chip decouples most of the guaranteed correct storage from the CPU manufacturing process. This allows for a larger proportion of on-chip SRAM cells to be scaled down with the process technology. A different technology can be used off-chip.

We ran an initial study to see what performance and storage impact selector bit caching would have. The design in Figure 5.4 was extended to have the working set of selector bit pages (stored in a parallel structure to the TLB) on-chip. On a TLB miss, we assume that the page table is accessed from main memory, so access latency to a small off-chip DRAM holding pages of selector bits is already accounted for. It is possible to use a small, standard cache rather than page-based method of finding the working set. This requires the use of tags, but makes more efficient use of storage space. Using the same M5 configuration as before, we recorded TLB miss rates for

Parameter	Value
Cache size	1024 kB
Linux page size	8 kB
Logical bits per selector bit	1
Selector page size	4 kB
Full grouping table size/OBI table size	4 kB/2 kB
Set-restricted grouping table size	3 kB
On-chip SRAM selector bit storage	96 kB
Off-chip DRAM selector bit storage	512 kB
ITLB/DTLB entries	8/16
Selector bit table DRAM throughput	32 bytes/cycle
Total storage overhead of full group table	12.3%
Total storage overhead of set-limited group table	9.9%
Total storage overhead of tag-based grouping table	9.6%

Table 5.3: Selector bit caching parameters and results

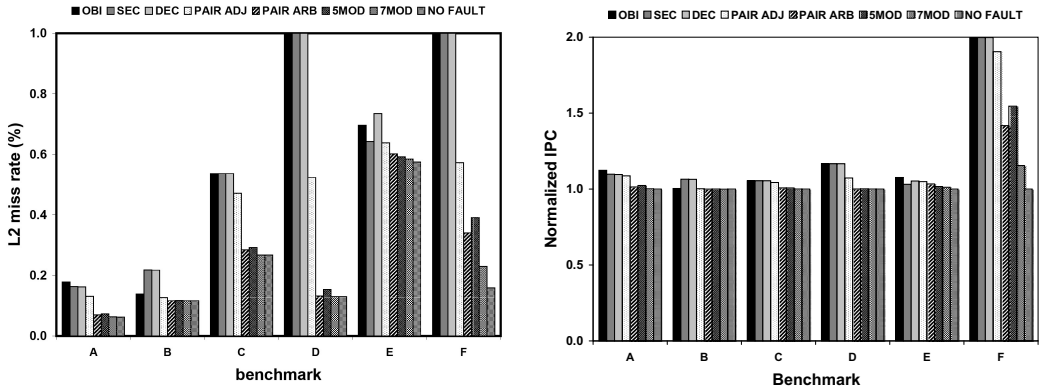
varying numbers of TLB entries, then derived the performance hit for off-chip selector bit loading. For more realism and to support virtual memory, the simulator was run in full-system mode and benchmarks were run to completion under Linux. The results are given in Table 5.3. By keeping just the working set of cache pages in on-chip SRAM we have reduced the on-die storage overhead from 50% to less than 10%. Note that we used one selector bit per logical bit (see Figure 5.6(c)). We opted to use 24 TLB entries because the performance data indicated much smaller slowdowns of 2% and 8% respectively for the Cholesky and Ocean-NonContig benchmarks. Due to the larger working set of Ocean-NonContig, increasing the number of data TLB entries does not significantly reduce miss rate.

5.5 Results

5.5.1 Performance and area under voltage scaling

We performed performance and power simulations using cell type C (1.5x the area of the smallest cell we considered) and a rotating voltage schedule for the CPU. Every 20 ms (4.10^7 cycles @ 2 GHz), the voltage was changed to the next level in a in sequence from 0.7, 0.8, 0.9, 1.0 and 1.1V. In reality, voltage changes will be less frequent in a DVS system. Because a large number of identical voltage changes occurred over the duration of each benchmark, a first-order approximation of energy con- sumption can be obtained by comparing overall execution times (assuming equivalent average power in each benchmark). It is envisioned that each performance change will require the following additional steps;

- Invalidate dirty blocks and write back to the next level of the memory hierarchy (or high-speed local storage), if they are known to contain un-correctable faults.
- Enable the appropriate bad block map for the new performance level.
- Reinstate saved blocks, or allow to be fetched when next accessed, as usual.

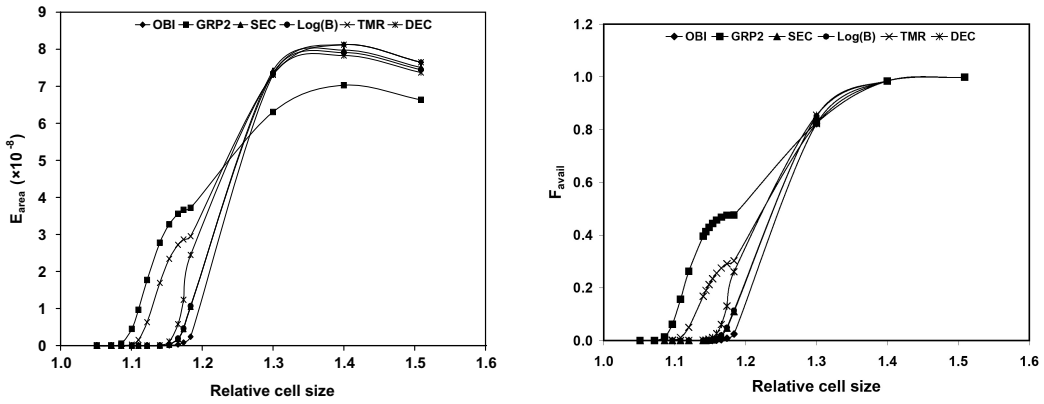


(a) L2 miss rate

(b) Normalized IPC

Figure 5.7: L2 miss rate in (a) and normalized IPC in (b) for each technique. A-Cholesky, B-FFT, C-LUContig, D-Radix, E-OceanContig, and F-OceanNoncontig, respectively.

Figure 5.7 compares the L2 miss rate for each benchmark with the same voltage schedule using different fault-tolerance schemes. PAIR ADJ and PAIR ARB refer to 2-block grouping schemes where ADJ means only physically adjacent bad blocks are paired, and ARB means that blocks are paired arbitrarily throughout the cache using the same greedy algorithm used for Figure 5.6. 5MOD and 7MOD implement 5- and 7-modular redundancy for each cell (representing 5 MB and 7 MB of physical storage), and NO FAULT refers to an ordinary error-free cache. It is clear that execution time (and IPC) are relatively insensitive to L2 miss rates (see Figure 5.7(b)). Note that arbitrary pairing generally performs better than 5-modular redundancy in most cases, with-out the large, fixed area overhead. Relative to OBI, PAIR ARB achieves an average 48% reduction in L2 miss rate and 13% reduction in execution time.

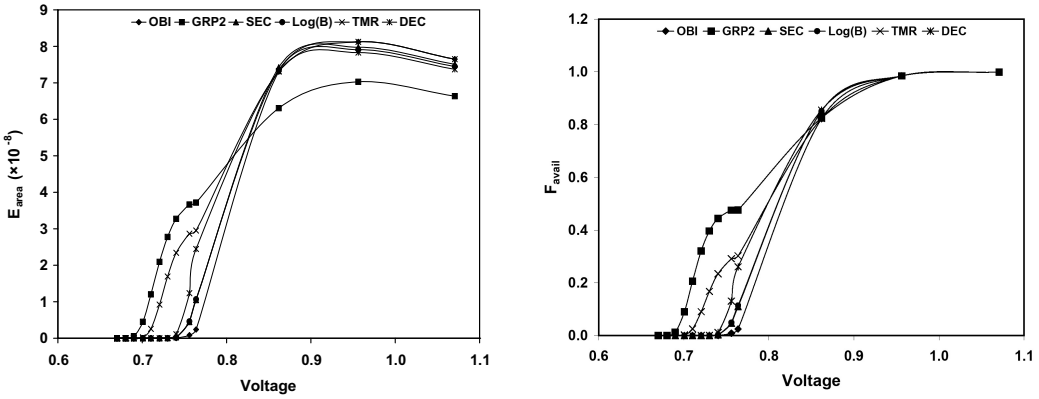


(a) E_{area} with cell scaling at 1.1V (64 byte block) (b) F_{avail} with cell scaling at 1.1V (64 byte block)

Figure 5.8: Comparison of schemes when scaling cell size. Cell size is relative to the smallest considered size from Figure 5.2

5.5.2 Performance under cell scaling

The E_{area} metric is shown in Figure 5.8(a). The most area efficient scheme is to use an OBI with TMR, as long as a cell is not scaled below size 1.4. This can



(a) E_{area} with voltage scaling (64 byte block, cell size C).

(b) F_{avail} with voltage scaling (64 byte block, cell size C).

Figure 5.9: Comparison of schemes when scaling voltage

be seen in the figure as the point of greatest E_{area} value. In fact, there is only a marginal improvement over using an OBI alone. Therefore, the area overhead of stronger error correction offsets the benefit of cell shrinking. The off-chip caching and arbitrary pairing variant was used for the grouping (GRP2) scheme. Therefore it initially has the worst E_{area} value due to the on-chip selector and grouping tables, but outperforms the others at smaller cell sizes due to superior fault-tolerance. In Figure 5.5.1 we examine E_{area} and F_{avail} as voltage is varied. This shows the same trends as Figure 5.5.1. This means that cache performance will drop significantly depending on the fault-tolerance scheme and how far voltage is scaled in low-power (or low activity) modes. CPUs using DVS should dynamically select a fault-tolerance scheme with the highest F_{avail} at the operating voltage. For example, Figure 5.9(b) indicates that DEC should be used down to 0.86 V and GRP2 below that.

5.5.3 Energy saving using block grouping at low voltage

An example of the energy benefits of block grouping is as follows. In an ultra-low voltage mode of 0.76 V (Figure 5.9(b)) conventional SEC code has an F_{avail} of around 0.02 while GRP2 is approximately 0.45. This means that GRP2 provides much more

Parameter	Value
Pentium M thermal design power	24.5 W
Selector table overhead power	2.6 W
Speedup of grouping relative to SEC	35%
Power with SEC	24.5 W
Power with grouping table	$24.5 + 2.6 = 27.1$ W
Average power with grouping	$27.1 * (1 - 0.35) = 17.6W$
Energy savings with grouping	28%

Table 5.4: Block grouping energy saving example

L2 cache at that voltage, reducing miss rate and improving IPC. Considering the Barnes benchmark in Figure 5.3(b), IPC for 86% disabled blocks is at most 1.14 while for grouping the IPC is 1.76 (55% disabled blocks). This means that execution using grouping completes at least $(1 - 1.14/1.76) = 35\%$ sooner. Energy savings are offset by the overhead of the selector bits and grouping table. If these are on-chip with one selector bit per data bit position, they will consume approximately 512 kB (or half the cache size). Despite this overhead, there will be a net energy saving. Using the power consumption of a 512 kB, 130 nm cache to represent the overhead of grouping [133], and that of a Pentium M running at around the same frequency [134] we estimate power consumption for the SEC and grouping schemes (Table 5.4). We use the thermal design power of the processor which implies the CPU is 100% utilised by the workload. In addition, we assume that the processor uses SEC fault-tolerance. Even though we theoretically scale voltage down to 0.76 V, both the processor and selector table have their voltages scaled by the same factor, so the power consumption ratio between the two is approximately the same.

5.6 Conclusions

The analysis in the first part of this paper compared several ways of maximizing the number of usable cache lines in the presence of faults. These faults can be a combination of permanent manufacturing faults as well as ineffective operation at low voltages. Increasing the size of an SRAM cell increases this reliability at the cost of extra area. Next, we proposed a novel fault-tolerance scheme that takes advantage of a region of larger or higher voltage SRAM cells to attain high reliability. The scheme works by grouping two or more cache lines divided into smaller regions. Selector bits in the high reliability memory cells are used to specify where the faulty bits are in the grouped blocks. It was determined that the previously published OBI-based fault tolerance is the most area efficient scheme for fault-tolerance at a single voltage. However, as voltage is scaled down, maximum performance and energy savings are obtained by switching from DEC to our GRP2 scheme. Instead of taking advantage of future scaling to reduce SRAM cell size, scaling should not go beyond a certain point. This is because the area overhead of trying to protect the smaller, but much less reliable cells is greater than that of not scaling the cells at all. However, error correction still has a place in cache design for low voltage performance and soft error tolerance. The strength of this error correction will be a function of expected soft error rates and how aggressively DVS is applied.

CHAPTER VI

Conclusions and Future Work

Data-center energy optimization clearly requires innovation at all levels of the component hierarchy from the network down to the individual processor sockets. Further, when taking advantage of emerging technologies to attain the most benefits, the new and inherent drawbacks of those devices must be dealt with efficiently. Specifically, emerging memories that suffer from wear-out require mechanisms for lifetime extension, and process technology scaling in nanometre regimes, while enabling low-power processors, require mechanisms for hiding process variation-induced faults. This thesis has demonstrated solutions across this design space, successfully leveraging emerging non-volatile memories through system-level optimization, and extending wear-leveling to the data center to improve lifetimes. With 2x to 10x improved data center energy efficiency, 3.9x improvement in TTFF for an SSD-based data center and more than 30% energy savings obtained for the CPU via cache fault-tolerance, the ensemble-level energy efficiency has been significantly improved. It is difficult to estimate the overall energy savings of cache fault tolerance applied to the Nanostores study. Because our models were tailored to 22nm technology in the 2015 timeframe, and the cache studies were evaluated at 45nm, the voltage-scaling limits of the two technologies differ significantly. However, we have estimated the

impact of cache fault-tolerance as applied to contemporary servers. From recent data [110], over 30% of server power is allocated to the CPU, which is the dominant power consumer. The majority of remaining power is attributed to main memory, power supply efficiency loss and PCI slots. A contemporary implementation of block grouping caches [10] states an approximate 50% decrease in energy per instruction (EPI) when operating with the L1 cache using block pairing and the L2 cache with another scheme called bit-fix. Since both schemes provide approximately the same performance (IPC) and voltage reduction (down to 500mV), a conservative estimate for server power reduction is $30\% * 0.5 = 15\%$, not including the extra savings in the reduced cooling infrastructure.

Despite addressing some of the biggest limitations and sources of inefficiency in the data center, there is scope for improvement of our designs as well as other system components which can be enhanced or re-designed. For Nanostores, the software infrastructure needs special consideration. The relatively small capacity of each networked node requires an efficient task distribution and load balancing mechanism to handle over an order of magnitude scale-out factor. Infrastructures such as MapReduce currently address this need, but they are geared towards hard disks where sequential accesses are preferred. Taking advantage of the extreme random-access performance of individual nodes remains a compelling research direction. Combining multiple types of NV memory (e.g. STT-RAM or multi-level cells) may yield different optimal design points. Network topologies other than fat-tree could be applied to the system, as well as non-uniform systems with high performance board-level networking and alternative off-board interconnects, including optical networks. Another direction is investigating how applications, file systems and operating systems should be written for a single-level memory/data store. For example, many file copy operations can be eliminated. An NV memory-based file system could assign filenames

or handles to output data in memory regions after an algorithm has finished, avoiding a redundant copy operation to a "permanent" file. We modified our benchmark execution plans with this in mind when executing on the Nanostore vs a disk-based system.

Regarding wear-out mitigation, there have been many proposed policies for block (Flash) and byte-addressable (e.g. PCM) memories, but most have not been considered at the data-center (distributed) level. We suggested that existing networked and virtualized file systems could be used to handle our proposed data distribution scheme. However, there are potential performance gains to be had from designing a dedicated NV-storage oriented network filesystem. Although we took the first steps towards ensemble-level wear-leveling, existing and novel wear-leveling mechanisms should be re-examined in the distributed context. Hybrid Flash+hard disk drives as well as PCM+Flash drives have been proposed in prior work, and may need special consideration the the context of our proposed schemes.

On the theme of fault-tolerance, there has been recent interest in the use of large, dense embedded DRAM caches to augment lower-capacity SRAM caches. One proposed fault-tolerance technique was used to reduce the frequency of refreshing the storage cells, rather than for SRAM cache voltage scaling [68]. Meanwhile, near-optimal error-correction schemes for PCM [103] serve to extend the lifetime of resistive memories as far as possible. Designing memory systems to fit lifetime, power and performance targets and integrating them into our optimization framework, given technology-specific fault tolerance, opens up new dimensions for system co-design.

APPENDICES

APPENDIX A

Implications of Global Wear-Leveling on Local (Start-Gap) Wear-Leveling

As an experiment, we examined how a possible PCRAM based SSD using distributed wear leveling (Section 4.3) before local Start-Gap wear leveling [51] would perform. In doing so, we highlight the potential for improved endurance. It should be noted that the principle of flattening the write count distribution (used in distributed wear-leveling) can also be applied to other levels of the hierarchy e.g. main memory or caches. Start-Gap wear-leveling is a simple scheme devised for PCRAM where a gap or empty block is moved one place to the right every N writes, by copying the block to the right of the gap one place to the left. To break up incoming block references that may be sequential and improve endurance, the incoming logical addresses are re-mapped randomly using a Feistel network. Figure 7.1 shows the normalized endurance of each SSD under Start-Gap wear leveling using a 64KB block granularity and gap movements every 50 writes, where block endurance is 10^8 writes. Normalized endurance represents the fraction of actual writes achievable before the first block fails, relative to perfect wear-leveling. Although real implementations would probably use smaller blocks, this is sufficient to illustrate the benefit of data re-organization to improve local wear-leveling. This large granularity introduces wear imbalance inside the blocks, in the case where only a few addressable LBAs are

written to. Other proposed schemes such as line-level writes and fine-grained wear-leveling [33] could be used to address this. As for Flash, we assume that the access pattern form the 7-day traces remains the same and can be repeated until the SSDs fail. The bars for before leveling represent Normalized Endurance for the baseline. For distributed wear leveling, the incoming local requests are first randomized using Feistel re-mapping. The logical addresses remapped blocks from other volumes are randomly picked from the unoccupied block locations (the file system would have made these address allocations earlier, under direction of the Management Node). Overall, distributed wear leveling increases the average normalized endurance from 0.48 to 0.82, an improvement of 70%. Without distributed wear leveling, some SSDs have a relatively small fraction of heavily accessed blocks. By the time a single gap rotation has completed, those blocks have exceeded 10^8 writes and therefore the normalized endurance is almost zero. A side-effect of the distributed wear leveling is to spread the blocks more evenly across volumes, reducing the standard deviation of block write counts on a drive by 38% on average. A lower standard deviation indicates more effective wear leveling [51].

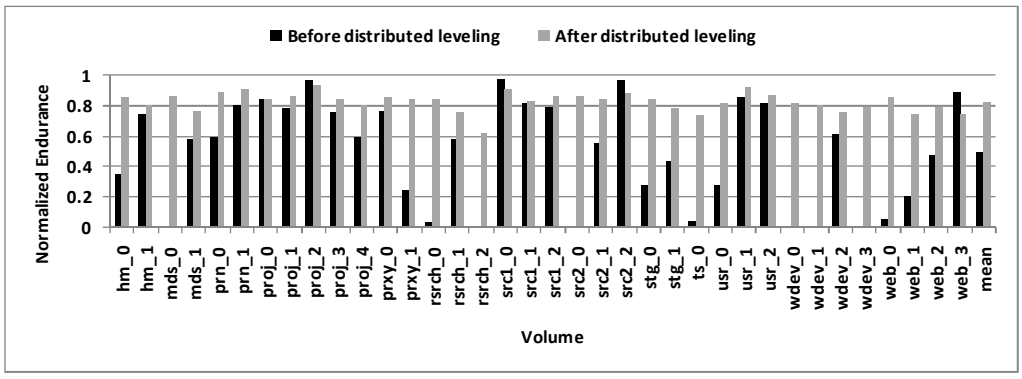


Figure 7.1: Start-Gap Normalized Endurance

It is challenging to quickly evaluate normalized endurance under wear-leveling

due to the large number of writes required to an SSD to reach the point of wear-out. Since writes are evenly distributed, every block needs to be written almost 10^8 times before the first block fails. Using brute-force trace replay until failure would take months of simulation time. Therefore we applied the following algorithm to quickly estimate the normalized endurance[1]. The algorithm description uses C-like pseudo-code.

1. Let A be floating-point vectors with one element per block in the storage device, where entries at a block index i are denoted by A_i.
2. Let *nblocks* be the number of blocks in the storage device.
3. Let *nwrites* be the total number of writes performed in the trace.
4. Let *interval* be the gap movement interval (number of writes to the device before the gap is moved by one position).
5. Let *endurance* be the number of writes that one block can sustain before failure.
6. After applying an address randomization function (e.g. using a Feistel network [51]), store the number of writes to each re-mapped block i of the trace in A_i.
7. Normalize the array A to represent the number of writes that would have occurred to each block in a single gap rotation interval (including one write per block for each gap movement).

$$\forall i : A_i = 1.0 + A_i * \left(\frac{interval * nblocks}{nwrites} \right)$$

8. Define a function *check*() which returns false if no element exceeds *endurance*, otherwise return true.
9. If $check(A) == true$, a block has failed before a single gap rotation has occurred.

[1] We acknowledge Moinuddin Qureshi for the idea of combining multiple gap rotations together

10. Let $B = A$

11. Let $shifts = sqrt(endurance)$

12. Define a function $rotate(X, j)$ that returns vector X rotated by j block positions.

13. Combine $shifts$ number of gap rotations together in vector B. Having completed this step, it is possible to advance wear-out in steps of $shifts$ gap rotations by combining shifted copies of vector B. After each rotation, check to see if any block exceeds *endurance*. If so, this is the maximum number of sustainable gap rotations.

```
for( j = 1; j <= shifts; j=j+1) {
        B = B + rotate(A, j);
        if( check(B) == true ) maximum_rotations = j-1;
}
```

14. Let $C = B$

15. As above, rotate and add the B vector to C until a block exceeds maximum endurance. This results in an approximate endurance within an error margin of $sqrt(endurance)$ writes.

```
for( k = 1; k <= shifts; k=k+1) {
        C = C + rotate(B, k*shifts);
        if( check(C) == true ) maximum_rotations = (k-1)*shifts;
}
```

16. If more accuracy is required, backtrack by one iteration ($shifts$ number of gap rotations) once endurance has been exceeded (by subtracting $rotate(B, k * shifts)$ from C). Then proceed to add vector A to C in single-shift steps until a block fails, to determine maximum_rotations to the nearest gap rotation.

17. Calculate normalized endurance;

$$normalized\_endurance = \frac{maximum\_rotations * interval}{endurance}$$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] J Scaramella. Enabling technologies for power and cooling. In *http://h71028.www7.hp.com/enterprise/downloads/thermal_logic.pdf*.

[2] David Roberts, Taeho Kgil, and Trevor Mudge. Integrating nand flash devices onto servers. *Commun. ACM*, 52(4):98–103, 2009.

[3] Itrs roadmap. In *http://www.itrs.net/*, 2009.

[4] T. Kgil, D. Roberts, and T. Mudge. Improving NAND Flash based Disk Caches. In *Proc. Int'l Symp. on Computer Architecture (ISCA)*, 2008.

[5] Vladimir Stojanovic. Silicon photonics and memories. In *Hot Chips, Optical Interconnect tutorial*, 2010.

[6] Ed Doller. Forging a future in memory - new technologies, new markets, new applications. In *Hot Chips tutorial*, 2010.

[7] T. Kgil and T. Mudge. FlashCache: a NAND flash memory file cache for low power web servers. In *Proc. Int'l Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, 2006.

[8] David Roberts, Taeho Kgil, and Trevor N. Mudge. Using non-volatile memory to save energy in servers. In *DATE*, pages 743–748, 2009.

[9] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pages 7–18, Dec. 2003.

[10] Chris Wilkerson, Hongliang Gao, Alaa R. Alameldeen, Zeshan Chishti, Muhammad Khellah, and Shih-Lien Lu. Trading off cache capacity for reliability to enable low voltage operation. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 203–214, 2008.

[11] David Roberts, Jichuan Chang, and Parthasarathy Ranganathan. Apparatus Having a Flattened-Level Data Storage Hierarchy and methods for its use. In *US Patent application 61/3111744*.

[12] Mehul Shah, Parthasarathy Ranganathan, Jichuan Chang, Niraj Tolia, David Roberts, and Trevor Mudge. Data dwarfs: Motivating a coverage set for future large data center workloads. In *Architectural Concerns in Large Datacenters (ACLD) Workshop, ISCA*, 2010.

[13] David Roberts, Jichuan Chang, and Parthasarathy Ranganathan. Managing Wear on Independent Storage Devices. In *US Patent application 12/813772*.

[14] David Roberts, Ganesh Dasika, and Trevor Mudge. Storage of data in Data Stores having some faulty storage locations. In *US Patent application 20080077824*.

[15] David Roberts, Nam Sung Kim, and Trevor Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. In *DSD '07: Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 570–578, 2007.

[16] David Roberts, Nam Sung Kim, and Trevor Mudge. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. *Microprocessors and Microsystems*, 2008.

[17] M.K. Patterson. The effect of data center temperature on energy efficiency. pages 1167 –1174, may. 2008.

[18] Yahoo opens energy efficient 'chicken coop' data center in n.y. In *http://www.pcmag.com/article2/0,2817,2369466,00.asp*.

[19] Google squared. In *http://www.google.com/squared/*.

[20] James Hamilton. Internet-scale service infrastructure efficiency. In *Keynote, ISCA*, 2009.

[21] Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 365–376, 2007.

[22] David Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. Fawn: A fast array of wimpy nodes. In *Proc. SOSP*, 2009.

[23] Adrian M. Caulfield, Laura M. Grupp, and Steven Swanson. Gordon: An improved architecture for data-intensive applications. *Micro, IEEE*, 30(1):121 –130, jan.-feb. 2010.

[24] John Ousterhout et. al. The case for ramclouds: Scalable high-performance storage entirely in dram.

[25] Richard Winter. Why are data warehouses growing so fast? In *http://www.b-eye-network.com/view/7188*, 2008.

[26] Marissa Mayer. The physics of data. In *Talk at Xerox PARC*, August 2009.

[27] Peter Lyman and Hal R. Varian. How much information. In *http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/*, 2003.

[28] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*, 2004.

[29] Fusionio. In *http://www.fusionio.com*.

[30] MetaRAM. `http://www.metaram.com`.

[31] Adam Leventhal. Flash storage memory. *Commun. ACM*, 51(7):47–51, 2008.

[32] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, 2009.

[33] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 24–33, 2009.

[34] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 14–23, 2009.

[35] Dennis Abts, Michael Marty, Philip Wells, Peter Klausler, and Hong Liu. Energy proportional datacenter networks. 2010.

[36] Xin Yuan. On nonblocking folded-clos networks in computer communication environments. 2010.

[37] Xuan-Yi Lin, Yeh-Ching Chung, and Tai-Yi Huang. A multiple lid routing scheme for fat-tree-based infiniband networks. *Parallel and Distributed Processing Symposium, International*, 1, 2004.

[38] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N.P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R.G. Beausoleil, and J.H. Ahn. Corona: System implications of emerging nanophotonic technology. In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pages 153–164, June 2008.

[39] David Roberts, Jichuan Chang, and Parthasarathy Ranganathan. Optical Data Path Systems. In *US Patent application 12/898798*.

[40] T. Kgil et Al. PicoServer: Using 3D Stacking Technology To Enable A Compact Energy Efficient Chip Multiprocessor. In *ASPLOS*, 2006.

[41] Marco Facchini, Trevor Carlson, Anselme Vignon, Martin Palkovic, Francky Catthoor, Wim Dehaene, Luca Benini, and Paul Marchal. System-level power/performance evaluation of 3d stacked drams for mobile applications. In *DATE*, pages 923–928.

[42] N. Madan, Li Zhao, N. Muralimanohar, A. Udipi, R. Balasubramonian, R. Iyer, S. Makineni, and D. Newell. Optimizing communication and capacity in a 3d stacked reconfigurable cache hierarchy. pages 262–274, March 2009.

[43] Dean Lewis and Hsien-Hsin Lee. Architectural evaluation of 3d stacked rram caches. *IEEE 3D System Integration Conference*, 2009.

[44] Wangyuan Zhang and Tao Li. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. *Parallel Architectures and Compilation Techniques, International Conference on*, 0:101–112, 2009.

[45] Chung Lam. Cell design considerations for phase change memory as a universal memory. In *VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on*, pages 132 –133, april 2008.

[46] David Lammers. Resistive ram gains ground. Sept 2010.

[47] D.B. Stukov, G.S. Snider, D.R. Steward, and R.S. Williams. The missing memristor found. In *Nature*, volume 453, pages 80–83, 2008.

[48] T. Raja and S. Mourad. Digital logic implementation in memristor-based crossbars. In *Communications, Circuits and Systems, 2009. ICCCAS 2009. International Conference on*, pages 939–943, July 2009.

[49] Xiangyu Dong, Naveen Muralimanohar, Norm Jouppi, Richard Kaufmann, and Yuan Xie. Leveraging 3d pcram technologies to reduce checkpoint overhead for future exascale systems. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, 2009.

[50] Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. Improving flash wear-leveling by proactively moving static data. *IEEE Trans. Comput.*, 59(1):53–65, 2010.

[51] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–23, 2009.

[52] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning dvs processor using delay-error detection and correction. *IEEE Journal of Solid-State Circuits*, 41:792–804, 2006.

[53] D. Roberts, T. Austin, D. Blaauw, T. Mudge, and K. Flautner. Error analysis for the support of robust voltage scaling. *ISQED*, 2005.

[54] D. Roberts, R. G. Dreslinski, E. Karl, T. Mudge, D. Sylvester, and D. Blaauw. When homogeneous becomes heterogeneous. *OSHMA Workshop, PACT*, 2007.

[55] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: the terasys massively parallel pim array. *Computer*, 28(4):23–31, Apr 1995.

[56] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. A Case for Intelligent RAM: IRAM. *IEEE Micro*, 17(2), Apr 1997.

[57] E. Riedel, C. Faloutsos, G.A. Gibson, and D. Nagle. Active disks for large-scale data processing. In *IEEE Computer*, volume 34, pages 68 –74, jun 2001.

[58] Adrian Cockcroft. Millicomputing: The future in your pocket and your datacenter. In *USENIX Conference, invited talk*, 2008.

[59] Kevin Lim, Parthasarathy Ranganathan, Jichuan Chang, Chandrakant Patel, Trevor Mudge, and Steven Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. *Computer Architecture, International Symposium on*, 0:315–326, 2008.

[60] Vijay Reddi, Benjamin Lee, Trishul Chilimbi, and Kushagra Vaid. Web Search Using Small Cores: Quantifying the Price of Efficiency. In *Microsoft Tech Report MSR-TR-2009-105*, August 2009.

[61] Bo Zhai, Ronald G. Dreslinski, David Blaauw, Trevor Mudge, and Dennis Sylvester. Energy efficient near-threshold chip multi-processing. In *ISLPED '07: Proceedings of the 2007 international symposium on Low power electronics and design*, pages 32–37, 2007.

[62] Wenjing Ma and Gagan Agrawal. A translation system for enabling data mining applications on gpus. In *ICS '09: Proceedings of the 23rd international conference on Supercomputing*, pages 400–409, 2009.

[63] Netezza. In *http://www.netezza.com*.

[64] Shinsuke Azuma, Takao Sakuma, Takashi Nakano, Takaaki Ando, and Kenji Shirai. High-performance sort chip. 1999.

[65] Peter A. Boncz, Martin L. Kersten, and Stefan Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, 2008.

[66] Asit Mishra, Joseph Hellerstein, and Walfredo Cirne. Towards characterizing cloud backend workloads: Insights from google compute clusters. 2009.

[67] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid cache architecture with disparate memory technologies. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 34–45, 2009.

[68] Chris Wilkerson, Alaa R. Alameldeen, Zeshan Chishti, Wei Wu, Dinesh Somasekhar, and Shih-lien Lu. Reducing cache power with low-cost, multi-bit error-correcting codes. In *ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture*, pages 83–93, 2010.

[69] In *ITRS Roadmap*, 2007.

[70] G.C. Han, J.J. Qiu, L. Wang, W.K. Yeo, and C.C. Wang. Perspectives of read head technology for 10 tb/in recording. volume 46, 2010.

[71] Personal discussions with memristor engineers. 2009.

[72] Guangyu Sun, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. A novel architecture of the 3d stacked mram l2 cache for cmps. pages 239 –249, feb. 2009.

[73] Xiaochen Guo, Engin Ipek, and Tolga Soyata. Resistive computation: avoiding the power wall with low-leakage, stt-mram based computing. In *ISCA '10: Proceedings of the 37th annual international symposium on Computer architecture*, pages 371–382, 2010.

[74] Taeho Kgil and Trevor Mudge. Flashcache: a nand flash memory file cache for low power web servers. *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2006.

[75] W. Robinett, G.S. Snider, P.J. Kuekes, and R. S. Williams. Computing with a trillion crummy components. In *Communications of the ACM*, volume 50, pages 35–39, 2007.

[76] Sung Hyun Jo, Kuk-Hwan Kim, and Wei Lu. High-density crossbar arrays based on a si memristive system. volume 9, pages 870–874, 2009.

[77] Dong Hyuk Woo, Nak Hee Seong, D.L. Lewis, and H.-H.S. Lee. An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth. pages 1 –12, jan. 2010.

[78] Simona Boboila and Peter Desnoyers. Write endurance in flash drives: Measurements and analysis. In *FAST'10*, 2010.

[79] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for ssd performance. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 57–70, 2008.

[80] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating server storage to ssds: analysis of tradeoffs. In *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158, 2009.

[81] Timothy Pritchett and Mithuna Thottethodi. Sievestore: A highly-selective, ensemble-level disk cache for cost-performance. In *ISCA'10, to appear*, 2010.

[82] Asim Kadav, Mahesh Balakrishnan, Vijayan Prabhakaran, and Dahlia Malkhi. Differential raid: rethinking raid for ssd reliability. In *HotStorage '09*, 2009.

[83] Gokul Soundararajan, Vijayan Prabhakaran, Mahesh Balakrishnan, and Ted Wobber. Extending ssd lifetimes with disk-based write caches. *FAST'10*, 2010.

[84] Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write off-loading: Practical power management for enterprise storage. *Trans. Storage*, 4(3):1–23, 2008.

[85] Tao Xie. Sea: A striping-based energy-aware strategy for data placement in raid-structured storage systems. *IEEE Trans. Comput.*, 57(6):748–761, 2008.

[86] Tao Xie and Yao Sun. Dynamic data reallocation in hybrid disk arrays. *IEEE Transactions on Parallel and Distributed Systems*, 99, 2009.

[87] A. Pour and M. Hill. Performance implications of tolerating cache faults. In *IEEE Trans. Comput. 42 (3) 257267*, 1993.

[88] S. Mukhopadhyay et al. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. In *IEEE Trans. CAD 24 (12)*, 2005.

[89] D. Bossen, J. Tendler, and K. Reick. Power4 system design for high reliability. In *IEEE Micro. 22 (2) 1624*, 2002.

[90] A. KleinOsowski and D. Lilja. The NanoBox project: exploring fabrics of self- correcting logic blocks for high defect rate molecular device technologies. In *IEEE Computer Society Annual Symposium on VLSI*, 2004.

[91] M. Nicolaidis, N. Achouri, and L. Anghel. A memory built-in self-repair for high defect densities based on error polarities. *18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 459–466, 2003.

[92] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy. A process-tolerant cache architecture for improved yield in nanoscale technologies. *IEEE Trans. Very Large Scale Integr. Syst.*, 13(1):27–38, 2005.

[93] Avesta Sasan, Houman Homayoun, Ahmed Eltawil, and Fadi Kurdahi. A fault tolerant cache architecture for sub 500mv operation: resizable data composer cache (rdc-cache). In *CASES '09: Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 251–260, 2009.

[94] M.A. Makhzan, A Khajeh, A Eltawil, and F Kurdahi. Limits on voltage scaling for caches utilizing fault tolerant techniques. In *ICCD*, Oct 2007.

[95] C. Wilkerson, Hongliang Gao, A.R. Alameldeen, Z. Chishti, M. Khellah, and Shih-Lien Lu. Trading off cache capacity for low-voltage operation. *Micro (Top Picks), IEEE*, 29(1):96 –103, jan. 2009.

[96] Jaume Abella, Javier Carretero, Pedro Chaparro, Xavier Vera, and Antonio Gonzalez. Low Vccmin Fault-Tolerant Cache with Highly Predictable Performance. In *MICRO*, December 2009.

[97] Cheng-Kok Koh, Weng-Fai Wong, Yiran Chen, and Hai Li. Tolerating process variations in large, set-associative caches: The buddy cache. *ACM Trans. Archit. Code Optim.*, 6(2), 2009.

[98] Cheng-Kok Koh, Weng-Fai Wong, Yiran Chen, and Hai Li. The salvage cache: a fault-tolerant cache architecture for next-generation memory technologies. In *ICCD'09: Proceedings of the 2009 IEEE international conference on Computer design*, 2009.

[99] Amin Ansari, Shuguang Feng, Shantanu Gupta, and Scott Mahlke. Enabling ultra low voltage system operation by tolerating on-chip cache failures. In *ISLPED '09: Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, 2009.

[100] Amin Ansari, Shantanu Gupta, Shuguang Feng, and Scott Mahlke. Zerehcache: armoring cache architectures in high defect density technologies. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[101] Zeshan Chishti, Alaa R. Alameldeen, Chris Wilkerson, Wei Wu, and Shih-Lien Lu. Improving cache lifetime reliability at ultra-low voltages. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[102] Engin Ipek, Jeremy Condit, Edmund B Nightingale, Doug Burger, and Thomas Moscibroda. Dynamically Replicated Memory: Building Reliable Systems from Nanoscale Resistive Memories. In *ASPLOS*, 2010.

[103] Stuart Schechter, Gabriel H. Loh, Karin Strauss, and Doug Burger. Use ecp, not ecc, for hard failures in memories. In *ISCA*, 2010.

[104] Robert T. Greenway, Kwangok Jeong, Andrew B. Kahng, Chul-Hong Park, and John S. Petersen. 32nm 1-d regular pitch sram bitcell design for interference-assisted lithography. *Photomask Technology*, 2008.

[105] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale Parallel Collaborative Filtering for the Netflix Prize. In *Algorithmic Aspects in Information and Management*, 2008.

[106] Fan Yang, Wen Su, Huibiao Zhu, and Qin Li. Formalizing mapreduce with csp. 2010.

[107] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Exploring power-performance tradeoffs in database systems. 2010.

[108] Cotson: Infrastructure for system-level simulation. In *MICRO 41*, 2008.

[109] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. Mcpat: An integrated power, area and timing modeling framework for multicore and manycore architectures. In *MICRO '09, to appear*, 2009.

[110] Lauri Minas and Brad Ellison. The problem of power consumption in servers. In *http://www.drdobbs.com/215800830*, 2009.

[111] Bert Hubert. Wondershaper. In *http://lartc.org/wondershaper/*, 2002.

[112] Nsort. In *http://www.ordinal.com/*.

[113] oprofile. In *http://oprofile.sourceforge.net/*.

[114] William Whitted and Gerald Aigner. Modular data center. In *United States Patent 7,278,273*, 2007.

[115] Sun Microsystems. Sun's modular data center. In *http://www.sun.com/service/sunmd/*.

[116] HP. Pod. In *http://h20338.www2.hp.com/enterprise/cache/595887-0-0-0-121.html*.

[117] Texas Memory Systems. Ramsan-300. In *http://www.ramsan.com/*.

[118] Nfs. In *http://en.wikipedia.org/wiki/Network_File_System_%28protocol%29*.

[119] William K. Josephson, Lars A. Bongo, David Flynn, and Kai Li. Dfs: A file system for virtualized flash storage. *FAST'10*, 2010.

[120] Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. The disksim simulation environment version 4.0 reference manual. In *University Parallel Data Lab Technical Report CMU-PDL-08-101*, 2008.

[121] Stan Park and Kai Shen. A performance evaluation of scientific i/o workloads on flash-based ssds. In *In IASDS at CLUSTER*, 2009.

[122] N. Kim et al. Leakage current Moores law meets static power. In *IEEE Comput. 36 (12) 6875*, 2003.

[123] M. Agostinelli et al. Erratic fluctuations of sram cache vmin at the 90nm process technology node. *IEEE Electron Devices Meeting (IEDM)*, December 2005.

[124] D. Burnett et al. Implementations of fundamental threshold voltage variations for high-density SRAM and logic circuits. In *IEEE International Symposium on VLSI Technology*, 1994.

[125] S. Mukhopadhyay et al. Modeling and estimation of failure probability due to parameter variation in nano-scale SRAMs for yield enhancement. In *IEEE International Symposium on VLSI Circuits*, 2004.

[126] Nathan L. Binkert, Erik G. Hallnor, and Steve K. Reinhardt. Network-oriented full-system simulation using m5, February 2003.

[127] S. Woo et al. The SPLASH-2 programs: characterization and methodological considerations. In *International Symposium on Computer Architecture (ISCA)*, 1995.

[128] D. Lamet and J. Frenzel. Defect-tolerant cache memory design. In *IEEE VLSI Test Symposium*, 1993.

[129] D. Lamet and J. Frenzel. Design of a fault-tolerant three-dimensional dynamic random- access memory with on-chip error-correcting circuit. In *IEEE Trans. Comput 42 (12) 14531468*, 1993.

[130] L. Joiner and J. Komo. Decoding binary BCH codes. In *IEEE SoutheastCon (March) 6773*, 1995.

[131] M. Nicolaidis, N. Achouri, and S. Boutobza. Dynamic data-bit memory built-in self-repair. In *International Conference on Computer Aided Design (ICCAD)*, 2003.

[132] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: a 32-way multithreaded Sparc processor. In *IEEE Micro. 25 (2) 2129*, 2005.

[133] J. Shin et al. Design and implementation of an embedded 512-kB level-2 cache subsystem. In *IEEE J. Solid-State Circuits (JSSC)*, 2005.

[134] Intel Corp. Pentium M Power Data, 1.6 GHz, Technology, 1 MB L2 Cache. In *http://www.intel.com/design/intarch/pentiumm/pentiumm.htm/*, 2007.