Memory Scaling: A Systems Architecture Perspective

Onur Mutlu

Carnegie Mellon University

onur@cmu.edu

http://users.ece.cmu.edu/~omutlu/

Abstract—**The memory system is a fundamental performance and energy bottleneck in almost all computing systems. Recent system design, application, and technology trends that require more capacity, bandwidth, efficiency, and predictability out of the memory system make it an even more important system bottleneck. At the same time, DRAM technology is experiencing difficult *technology scaling* challenges that make the maintenance and enhancement of its capacity, energy-efficiency, and reliability significantly more costly with conventional techniques.**

In this paper, after describing the demands and challenges faced by the memory system, we examine some promising research and design directions to overcome challenges posed by memory scaling. Specifically, we survey three key solution directions: 1) enabling new DRAM architectures, functions, interfaces, and better integration of the DRAM and the rest of the system, 2) designing a memory system that employs emerging memory technologies and takes advantage of multiple different technologies, 3) providing predictable performance and QoS to applications sharing the memory system. We also briefly describe our ongoing related work in combating scaling challenges of NAND flash memory.

I. INTRODUCTION

Main memory is a critical component of all computing systems, whether they be server, embedded, desktop, mobile, sensor. Memory capacity, energy, cost, performance, and management algorithms must scale as we scale the size of the computing system in order to maintain performance growth and enable new applications. Unfortunately, such scaling has become difficult because recent trends in systems, applications, and technology exacerbate the memory system bottleneck.

II. TRENDS AND REQUIREMENTS

In particular, on the *systems/architecture front*, energy and power consumption have become key design limiters as the memory system continues to be responsible for a significant fraction of overall system energy/power [43]. More and increasingly heterogeneous [15, 70, 29] processing cores and agents/clients are sharing the memory system, leading to increasing demand for memory capacity and bandwidth along with a relatively new demand for predictable performance and QoS from the memory system [52, 57, 69]. On the *applications front*, important applications are usually very data intensive and are becoming increasingly so [6], requiring both real-time and offline manipulation of great amounts of data. For example, next-generation genome sequencing technologies produce massive amounts of sequence data that overwhelms memory storage and bandwidth requirements of today's high-end desktop and laptop systems [71, 3, 74] yet researchers have the goal of enabling low-cost personalized medicine. Creation of new *killer applications* and usage models for computers likely depends on how well the memory system can support the efficient storage and manipulation of data in such data-intensive applications. In addition, there is an increasing trend towards consolidation of applications on a chip, which leads to the sharing of the memory system across many heterogeneous applications with diverse performance requirements, exacerbating the aforementioned need for predictable performance guarantees from the memory system. On the *technology front*, two key trends profoundly affect memory systems. First, there is increasing difficulty scaling the well-established charge-based memory technologies, such as DRAM [48, 4, 38, 1] and flash memory [35, 47, 9, 10, 12], to smaller technology nodes. Such scaling has enabled memory systems with reasonable capacity and efficiency; lack of it will make it difficult to achieve high capacity and efficiency at low cost. Second, some emerging resistive memory technologies, such as phase change memory (PCM) [66, 73, 38, 39, 65] or spin-transfer torque magnetic memory (STT-MRAM) [14, 36] appear more scalable, have latency and bandwidth characteristics much closer to DRAM than flash memory and hard disks, and are non-volatile with little idle power consumption. Such emerging technologies can enable new opportunities in system design, including, for example, the unification of memory and storage subsystems. They have the potential to be employed as part of main memory, alongside or in place of less scalable and leaky DRAM, but they also have various shortcomings depending on the technology (e.g., some have cell endurance problems, some have very high write latency/power, some have low density) that need to be overcome.

III. SOLUTION DIRECTIONS

As a result of these systems, applications, and technology trends and the resulting requirements, it is our position that researchers and designers need to fundamentally rethink the way we design memory systems today to 1) overcome scaling challenges with DRAM, 2) enable the use of emerging memory technologies, 3) design memory systems that provide predictable performance and quality of service to applications and users. The rest of the paper describes our solution ideas in these three directions, with pointers to specific techniques when possible. Since scaling challenges themselves arise due to difficulties in enhancing memory components at *solely* one level of the computing stack (e.g., the device and/or circuit levels in case of DRAM scaling), we believe effective solutions to the above challenges will require cooperation across different layers of the computing stack, from algorithms to software to microarchitecture to devices, as well as between different components of the system, including processors, memory controllers, memory chips, and the storage subsystem.

IV. CHALLENGE 1: NEW DRAM ARCHITECTURES

DRAM has been the choice technology for implementing main memory due to its relatively low latency and low cost. DRAM process technology scaling has for long enabled lower cost per unit area by enabling reductions in DRAM cell size. Unfortunately, further scaling of DRAM cells has become costly [4, 48, 38, 1] due to increased manufacturing complexity/cost, reduced cell reliability, and potentially increased cell leakage leading to high refresh rates. Several key issues to tackle include:

1) reducing the negative impact of refresh on energy, performance, QoS, and density scaling [45],

2) improving DRAM parallelism/bandwidth [34], latency [42], and energy efficiency [34, 42, 45],

3) improving reliability of DRAM cells at low cost,

4) reducing the significant amount of waste present in today's main memories in which much of the fetched/stored data can be unused due to coarse-granularity management [50, 76],

5) minimizing data movement between DRAM and processing elements, which causes high latency, energy, and bandwidth consumption [68].

Traditionally, DRAM devices have been separated from the rest of the system with a rigid interface, and DRAM has been treated as a *passive* slave device that simply responds to the commands given to it by the memory controller. We believe the above key issues can be solved more easily if we rethink the DRAM architecture and functions, and redesign the interface such that DRAM, controllers, and processors closely cooperate. We call this high-level solution approach *system-DRAM co-design*. We believe key technology trends, e.g., the 3D stacking of memory and logic [46, 2] and increasing cost

of scaling DRAM solely via circuit-level approaches, enable such a co-design to become increasingly feasible. We proceed to provide several examples from our recent research that tackle the problems of refresh, parallelism, latency, and energy efficiency.

A. Reducing Refresh Impact and DRAM Error Management

With higher DRAM capacity, more cells need to be refreshed at likely higher rates than today. Our recent work [45] indicates that refresh rate limits DRAM density scaling: a hypothetical 64Gb DRAM device would spend 46% of its time and 47% of all DRAM energy for refreshing its rows, as opposed to typical 4Gb devices of today that spend respectively 8% of the time and 15% of the DRAM energy on refresh. Today's DRAM devices refresh all rows at the same worst-case rate (e.g., every 64ms). However, only a small number of weak rows require a high refresh rate [31, 44] (e.g., only ~1000 rows in 32GB DRAM require to be refreshed more frequently than every 256ms). Retention-Aware Intelligent DRAM Refresh (RAIDR) [45] exploits this observation: it groups DRAM rows into bins (implemented as Bloom filters to minimize hardware overhead) based on the retention time of the weakest cell within each row. Each row is refreshed at a rate corresponding to its retention time bin. Since few rows need high refresh rate, one can use very few bins to achieve large reductions in refresh counts: our results show that RAIDR with three bins (1.25KB hardware cost) reduces refresh operations by ~75%, leading to significant improvements in system performance and energy efficiency as described by Liu et al. [45].

Note that such approaches that exploit non-uniform retention times across DRAM, like RAIDR, require accurate retention time profiling mechanisms. Understanding of retention time as well as error behavior of DRAM devices is a critical research topic, which we believe can enable other mechanisms to tolerate refresh impact and errors at low cost. Liu et al. [44] provides an experimental characterization of retention times in modern DRAM devices to aid such understanding. Our initial results in that work, obtained via the characterization of 248 modern commodity DRAM chips from five different DRAM manufacturers, suggests that the retention time of cells in a modern device is largely affected by two phenomena: 1) Data Pattern Dependence, where the retention time of each DRAM cell is significantly affected by the data stored in other DRAM cells, 2) Variable Retention Time, where the retention time of some DRAM cells changes unpredictably over time. These two phenomena pose challenges against accurate and reliable determination of the retention time of DRAM cells, online or offline, and a promising area of future research is to devise techniques that can identify retention times of DRAM cells in the presence of data pattern dependence and variable retention time.

Looking forward, we believe that increasing cooperation between the DRAM device and the DRAM controller as well as other parts of the system, including system software, is needed to communicate information about *weak* (or, unreliable) cells and the characteristics of different rows or physical memory regions from the device to the system. The system can then use this information to optimize data allocation and movement, refresh rate management, and error tolerance mechanisms. Low-cost error tolerance mechanisms are likely to be enabled more efficiently with such coordination between DRAM and the system. In fact, as DRAM technology scales and error rates increase, it might become increasingly more difficult to maintain the common illusion that DRAM is a perfect, error-free storage device. DRAM may start looking increasingly like flash memory, where the memory controller manages errors such that an acceptable specified uncorrectable bit error rate is satisfied.

B. Improving DRAM Parallelism

A key limiter of DRAM parallelism is bank conflicts. We have recently developed mechanisms, called SALP (subarray level parallelism) [34], that exploit the internal subarray structure of the DRAM bank to *mostly* parallelize two requests that access the same DRAM bank. The key idea is to reduce the hardware sharing between DRAM subarrays slightly such that accesses to the same bank but different subarrays can be initiated in a pipelined manner. This mechanism requires the exposure of the internal subarray structure of DRAM to the controller and the design of the controller to take advantage of this structure. Our results show significant improvements in performance and energy efficiency of main memory due to parallelization of requests and improvement of row buffer hit rates (as row buffers of different subarrays can be kept active) at a low DRAM area overhead of 0.15%. Exploiting SALP achieves most of the benefits of increasing the number of banks at much lower area and power overhead than doing so. Exposing the subarray structure of DRAM to other parts of the system, e.g., to system software or memory allocators, can enable data placement and partitioning mechanisms that can improve performance and efficiency even further.

C. Reducing DRAM Latency and Energy

The DRAM industry has so far been primarily driven by the cost-per-bit metric: provide maximum capacity for a given cost. To mitigate the high area overhead of DRAM sensing structures, commodity DRAMs connect many DRAM cells to each sense-amplifier through a wire called a bitline. These bitlines have a high parasitic capacitance due to their long length, and this bitline capacitance is the dominant source of DRAM latency. Specialized low-latency DRAMs use shorter bitlines with fewer cells, but have a higher cost-per-bit due to greater sense-amplifier area overhead. We have recently shown that we can architect a heterogeneous-latency bitline DRAM, called Tiered-Latency DRAM (TL-DRAM) [42], by dividing a long bitline into two shorter segments using an isolation transistor: a low-latency segment can be accessed with the latency and efficiency of a short-bitline DRAM (by turning off the isolation transistor that separates the two segments) while the high-latency segment enables high density, thereby reducing cost-per-bit (The additional area overhead of TL-DRAM is approximately 3% over commodity DRAM). Significant performance and energy improvements can be achieved by exposing the two segments to the memory controller and system software such that appropriate data is cached or allocated into the low-latency segment. We expect such approaches that design and exploit heterogeneity to enable/achieve the best of multiple worlds [55] in the memory system can lead to other novel mechanisms that can overcome difficult contradictory tradeoffs in design.

D. Exporting Bulk Data Operations to DRAM

Today's systems waste significant amount of energy, DRAM bandwidth, time (as well as valuable on-chip cache space) by unnecessarily moving data from main memory to processor caches. One example of such wastage sometimes occurs for bulk data copy and initialization operations in which a page is copied to another or initialized to a value. If the copied or initialized data is not immediately needed by the processor, performing such operations within DRAM (with relatively small changes to DRAM) can save significant amounts of energy, bandwidth, and time. We observe that a DRAM chip internally operates on bulk data at a row granularity. Exploiting this internal structure of DRAM can enable page copy and initialization to be performed entirely within DRAM without bringing any data off the DRAM chip [68]. If the source and destination page reside within the same DRAM subarray, our results show that a page copy can be accelerated by more than an order of magnitude, leading to an energy reduction of ~74 times and *no* wastage of DRAM data bus bandwidth [68]. The key idea is to capture the contents of the source row in the sense amplifiers by activating the row, then *deactivating* the source row (using a new command which introduces very little hardware cost, amounting to less than 0.03% of DRAM chip area), and immediately activating the destination row, which causes the sense amplifiers to drive their contents into the destination row, effectively accomplishing the page copy.

Going forward, we believe acceleration of other bulk data movement and computation operations in or very close to DRAM can enable promising savings in system energy, latency, and bandwidth. Given the trends and requirements described in Section II, it is likely time to re-examine the partitioning of computation between processors and DRAM, treating memory as a first-class accelerator as an integral part of a heterogeneous parallel computing system [55].

E. Minimizing Capacity and Bandwidth Waste

Storing and transferring data at large granularities (e.g., pages, cache blocks) within the memory system leads to large inefficiency when most of the large granularity is not needed [76, 77, 50, 49, 72, 67, 37, 63, 30]. In addition, much of the data stored in memory has significant redundancy [75, 25, 61]. Two promising research directions are to develop techniques that can 1) efficiently provide fine granularity access/storage when enough and large granularity access/storage only when needed, 2) efficiently compress data in main memory and caches without significantly increasing latency and system complexity. Our results with new low-cost low-latency cache compression [61] and memory compression [62] techniques and frameworks are promising, providing high compression ratios at low complexity and latency. For example, the key idea of *Base-Delta-Immediate compression* [61] is that many cache blocks have low dynamic range in the values they store, i.e., the differences between values stored in the cache block are small. Such a cache block can be encoded using a base value and an array of much smaller (in size) differences from that base value, which together occupy much less space than storing the full values in the original cache block. This compression algorithm has low decompression latency as the cache block can be reconstructed using a vector addition. It reduces memory bandwidth requirements, better utilizes memory/cache space, while minimally impacting the latency to access data. Granularity management and data compression support can potentially be integrated into DRAM controllers or partially provided within DRAM, and such mechanisms can be exposed to software, which can enable higher energy savings and higher performance improvements.

V. CHALLENGE 2: EMERGING MEMORY TECHNOLOGIES

While DRAM scaling is in jeopardy, some emerging technologies seem more scalable. These include PCM and STT-MRAM. These emerging technologies usually provide a tradeoff, and seem unlikely to completely replace DRAM (evaluated in [38, 39, 40] for PCM and in [36] for STT-MRAM), as they are not strictly superior to DRAM. For example, PCM is advantageous over DRAM because it 1) has been demonstrated to scale to much smaller feature sizes and can store multiple bits per cell [79], promising higher density, 2) is non-volatile and as such requires no refresh (which is a key scaling challenge of DRAM as we discussed in Section IV-A), and 3) has low idle power consumption. On the other hand, PCM has significant shortcomings compared to DRAM, which include 1) higher read latency and read energy, 2) much higher write latency and write energy, and 3) limited endurance for a given PCM cell, a problem that does not exist (practically) for a DRAM cell. As a result, a research challenge is how to utilize such emerging technologies at the system and architecture levels such that they can augment or perhaps even replace DRAM.

Our initial experiments and analyses [38, 39, 40] that evaluated the complete replacement of DRAM with PCM showed that one would require reorganization of peripheral circuitry of PCM chips (with the goal of absorbing writes and reads before they update or access the PCM cell array) to enable PCM to get close to DRAM performance and efficiency. These initial results are reported in Lee et al. [38, 39, 40]. We have also reached a similar conclusion upon evaluation of the complete replacement of DRAM with STT-MRAM [36]: reorganization of peripheral circuitry of STT-MRAM chips (with the goal of minimizing the number of writes to the STT-MRAM cell array, as write operations are high-latency and high-energy in STT-MRAM) enables an STT-MRAM based main memory to be much more energy-efficient than a DRAM-based main memory.

One can achieve more efficient designs of PCM (or STT-MRAM) chips by taking advantage of the non-destructive nature of reads, which enables simpler and narrower row buffer organizations [49] Unlike in DRAM, the entire memory row does not need to be buffered in a device where reading a memory row does not destroy the data stored in the row. Meza et al. [49] shows that having narrow row buffers in emerging non-volatile devices can greatly reduce main memory dynamic energy compared to a DRAM baseline with large row sizes, without greatly affecting endurance, and for some NVM technologies, leading to improved performance. Going forward, designing systems, memory controllers and memory chips taking advantage of the specific property of non-volatility of emerging technology seems promising.

We believe emerging technologies enable at least three major system-level opportunities that can improve overall system efficiency: 1) hybrid main memory systems, 2) non-volatile main memory, 3) merging of memory and storage. We briefly touch upon each.

A. Hybrid Main Memory

A hybrid main memory system [65, 20, 50, 78] consists of multiple different technologies with differing characteristics, e.g., performance, cost, energy, reliability, endurance. A key question is how to manage data allocation and movement between the different technologies such that one can achieve the best of (or close to the best of) the desired performance metrics. Put another way, we would like to achieve the advantages of each technology while hiding the disadvantages of each technology.

The design space of hybrid memory systems is large, and many potential questions exist. For example, should all memories be part of main memory or should some of them be used as a cache of main memory (or should there be configurability)? What technologies should be software visible? What component of the system should manage data allocation and movement? Should these tasks be done in hardware, software, or collaboratively? At what granularity should data moved between different memory technologies? Some of these questions are tackled in [50, 78, 65, 20], among other works recently published in the computer architecture community. For example, Yoon et al. [78] makes the key observation that row buffers are present in both DRAM and PCM, and they have (or can be designed to have) the same latency and bandwidth in both DRAM and PCM. Yet, row buffer misses are much more costly in terms of latency, bandwidth, and energy in PCM than in DRAM. To exploit this, we devise a policy that avoids accessing in PCM data that frequently causes row buffer misses. Hardware or software can dynamically keep track of such data and allocate/cache it in DRAM whereas keeping data that frequently hits in the row buffer in PCM. PCM also has much higher write latency/power than read latency/power: to take this into account, the allocation/caching policy is biased such that pages that are written to more likely stay in DRAM [78].

B. Exploiting and Securing Non-volatile Main Memory

Non-volatility of main memory opens up new opportunities that can be exploited by higher levels of the system stack to improve performance and reliability/consistency [21, 16]. Researching how to adapt applications and system software to utilize fast, byte-addressable non-volatile main memory is an important research direction to pursue [51].

On the flip side, the same non-volatility can lead to potentially unforeseen security and privacy issues: critical and private data can persist long after the system is powered down, and an attacker can take advantage of this fact. Wearout issues of emerging technology can also cause attacks that can intentionally degrade memory capacity in the system [64]. Securing non-volatile main memory is therefore an important systems challenge.

C. Merging of Memory and Storage

Traditional computer systems have a two-level storage model: they access and manipulate 1) volatile data in main memory (DRAM, today) with a fast load/store interface, 2) persistent data in storage media (flash and hard disks, today) with a slower file system interface. Unfortunately, such a decoupled memory/storage model managed via vastly different techniques (fast, hardware-accelerated memory management units on one hand, and slow operating/file system (OS/FS) software on the other) suffers from large inefficiencies in locating data, moving data, and translating data between the different formats of these two levels of storage that are accessed via two vastly different interfacesleading to potentially large amounts of wasted work and energy [51]. The two different interfaces arose largely due to the large discrepancy in the access latencies of conventional technologies used

to construct volatile memory (DRAM) and persistent storage (hard disks and flash).

Today, new non-volatile memory technologies (NVM), e.g, PCM, STT-MRAM, RRAM, show the promise of storage capacity and endurance similar to or better than flash memory at latencies comparable to DRAM. This makes them prime candidates for providing applications a persistent *single-level store* with a single load/store-like interface to access all system data (including volatile and persistent data). In fact, if we keep the traditional two-level memory/storage model in the presence of these fast NVM devices as part of storage, the operating system and file system code for locating, moving, and translating persistent data from the non-volatile NVM devices to volatile DRAM for manipulation purposes becomes a great bottleneck, causing most of the energy consumption and degrading performance by an order of magnitude in some data-intensive workloads, as we showed in recent work [51]. With energy as a key constraint, and in light of modern high-density NVM devices, a promising research direction is to unify and coordinate the management of volatile memory and persistent storage in a single level, to eliminate wasted energy and performance, and to simplify the programming model at the same time.

To this end, Meza et al. [51] describe the vision and research challenges of a persistent memory manager (PMM), a hardware acceleration unit that coordinates and unifies memory/storage management in a single address space that spans potentially multiple different memory technologies (DRAM, NVM, flash) via hardware/software cooperation. The spirit of the PMM unit is much like the virtual memory management unit of a modern virtual memory system used for managing working memory, but it is fundamentally different in that it redesigns/rethinks the virtual memory and storage abstractions and unifies them in a different interface supported by scalable hardware mechanisms. The PMM: 1) exposes a load/store interface to access persistent data, 2) manages data placement, location, persistence semantics, and protection (across multiple memory devices) using both dynamic access information and hints from the application and system software, 3) manages metadata storage and retrieval, needed to support efficient location and movement of persistent data, and 4) exposes hooks and interfaces for applications and system software to enable intelligent data placement and persistence management. Our preliminary evaluations show that the use of such a unit, if scalable and efficient, can eliminate the energy inefficiency and performance overheads of the two-level storage model, improving both performance and energy-efficiency of the overall system, especially for data-intensive workloads [51].

We believe there are challenges to be overcome in the design, use, and adoption of such a unit that unifies working memory and persistent storage. These challenges include:

1. How to devise efficient and scalable data mapping, placement, and location mechanisms (which likely need to be hardware/software cooperative).

2. How to ensure that the consistency and protection requirements of different types of data are adequately, correctly, and reliably satisfied. How to enable the reliable and effective coexistence and manipulation of volatile and persistent data.

3. How to redesign applications such that they can take advantage of the unified memory/storage interface and make the best use of it by providing appropriate hints for data allocation and placement to the persistent memory manager.

4. How to provide efficient and high-performance backward compatibility mechanisms for enabling and enhancing existing memory and storage interfaces in a single-level store. These techniques can seamlessly enable applications targeting traditional two-level storage systems to take advantage of the performance and energy-efficiency benefits of systems employing single-level stores. We believe such techniques are needed to ease the software transition to a radically different storage interface.

VI. Challenge 3: Predictable Performance

Since memory is a shared resource between multiple cores (or, agents, threads, or applications), different applications contend for memory bandwidth and capacity. As such, memory contention, or memory interference, between different cores critically affects both the overall system performance and each application's performance. Providing the appropriate bandwidth and capacity allocation to each application such that its performance requirements are satisfied is important to satisfy user expectations and service level agreements, and at the same time enable better system performance. Our past work [52, 57, 58] showed that application-unaware design of memory controllers, and in particular memory scheduling algorithms, leads to uncontrolled interference of applications in the memory system. Such uncontrolled interference can lead to denial of service to some applications [52], low system performance [57, 58], and an inability to satisfy performance requirements [57, 69, 22], which makes the system uncontrollable and unpredictable. In fact, an application's performance depends on what other applications its sharing resources with: an application can sometimes have very high performance and other times very low performance on the same system, solely depending on its co-runners. A critical research challenge is therefore how to design the memory system (including all shared resources such as main memory, caches, and interconnects) such that 1) the performance of each application is predictable and controllable, 2) while the performance and efficiency of the entire system are as high as needed or possible.

To achieve these goals, we have designed various solutions including QoS-aware memory controllers [57, 58, 53, 32, 33, 54, 5, 69, 41, 23], interconnects [17, 18, 27, 28, 13, 59, 60, 19], and entire memory systems including caches, interconnect, and memory [22, 24, 19]. These works enhanced our understanding and provide viable and effective mechanisms that improve system performance and predictability, but we are far from perfectly 1) estimating and predicting application performance in the presence of interference and a dynamic system with continuously incoming and outgoing applications and 2) enforcing *end-to-end performance guarantees* within the entire shared memory system. Subramanian et al. [69] provides a simple method for estimating application slowdowns in the presence of main memory interference. We observe that an application's request service rate is a good proxy for performance. As such, an application's slowdown can be accurately estimated by estimating its uninterfered request service rate, which can be done by prioritizing that application's requests in the memory system during some execution intervals. Results show that average error in slowdown estimation with this relatively simple technique is approximately 8% across a wide variety of workloads. Extending such simple techniques to the entire memory and storage system is a promising area of future research. Devising memory devices and architectures that can support predictability and QoS also appears promising.

VII. Aside: Flash Scaling Challenges

Flash memory, another successful charge-based memory like DRAM, has been commonly employed as part of the storage system. In part of our research, we aim to develop new techniques that overcome reliability and endurance challenges of flash memory to enable its scaling beyond the 20nm technology generations. To this end, we experimentally measure, characterize, analyze, and model error patterns that occur in existing flash chips, using an experimental flash memory testing and characterization platform [8]. Based on the understanding we develop from our experiments, we aim to develop error management techniques that mitigate the fundamental types of errors that are likely to increase as flash memory scales.

We have recently experimentally characterized complex flash errors that occur at 30-40nm flash technologies [9], categorizing them into four types: retention errors, program interference errors, read errors, and erase errors. Our characterization shows the relationship between various types of errors and demonstrates empirically using real 3x-nm flash chips that retention errors are the most dominant error type. Our results demonstrate that different flash errors have distinct patterns: retention errors and program interference errors are program/erase-(P/E)-cycle-dependent, memory-location-dependent, and data-value-dependent. Since the observed error patterns are due to fundamental circuit and device behavior inherent in flash memory, we expect our observations and error patterns to also hold in flash memories beyond 30-nm technology.

Based on our experimental characterization results that show that the retention errors are the most dominant errors, we have developed a suite of techniques to mitigate the effects of such errors, called Flash Correct-and-Refresh (FCR) [10]. The key idea is to periodically read each page in flash memory, correct its errors using simple error correcting codes (ECC), and either remap (copy/move) the page to a different location or reprogram it in its original location by recharging the floating gates, before the page accumulates more errors than can be corrected with simple ECC. Our simulation experiments using real I/O workload traces from a variety of file system, database, and search applications show that FCR can provide 46x flash memory lifetime improvement at only 1.5% energy overhead, with no additional hardware cost.

Recently, we have also experimentally investigated and characterized the threshold voltage distribution of different logical states in MLC NAND flash memory [12]. We have developed new models that can predict the shifts in the threshold voltage distribution based on the number of P/E cycles endured by flash memory cells. Our data shows that the threshold voltage distribution of flash cells that store the same value can be approximated, with reasonable accuracy, as a Gaussian distribution. The threshold voltage distribution of flash cells that store the same value gets distorted as the number of P/E cycles increases, causing threshold voltages of cells storing different values to overlap with each other, which can lead to the incorrect reading of values of some cells as flash cells accumulate P/E cycles. We find that this distortion can be accurately modeled and predicted as an exponential function of the P/E cycles, with more than 95% accuracy. Such predictive models can aid the design of more sophisticated error correction methods, such as LDPC codes [26], which are likely needed for reliable operation of future flash memories.

We are currently investigating another increasingly significant obstacle to MLC NAND flash scaling, which is the increasing cell-to-cell program interference due to increasing parasitic capacitances between the cells' floating gates. Accurate characterization and modeling of this phenomenon are needed to find effective techniques to combat program interference. In recent work [11], we leverage the *read retry* mechanism found in some flash designs to obtain measured threshold voltage distributions from state-of-the-art 2Y-nm (i.e., 24-20 nm) MLC NAND flash chips. These results are then used to characterize the cell-to-cell program interference under various programming conditions. We show that program interference can be accurately modeled as additive noise following Gaussian-mixture distributions, which can be predicted with 96.8% accuracy using linear regression models. We use these models to develop and evaluate a read reference voltage prediction technique that reduces the raw flash bit error rate by 64% and increases the flash lifetime by 30%. More detail can be found in Cai et al. [11].

Going forward, we believe more accurate and detailed characterization of flash memory error mechanisms is needed to devise models that can aid the design of more efficient and effective mechanisms to tolerate errors found in sub-20nm flash memories. A promising direction is the design of predictive models that the system (e.g., the flash controller or system software) can use to proactively estimate the occurrence of errors and take action to prevent the error before it happens. Flash-correct-and-refresh [10], described earlier, can be thought of as an early form of such a predictive error tolerance mechanism.

VIII. Conclusion

We have described several research directions and ideas to enhance memory scaling via system and architecture-level approaches, by co-designing memory and other system components as well as with cooperation across multiple levels of the computing stack, including software, microarchitecture, and devices. We believe such approaches will become increasingly important and effective as the underlying memory technology nears its scaling limits at the physical level.

IX. Acknowledgments

Part of the structure of this paper is based on talks I have delivered at various venues on *Scaling the Memory System in the Many-Core Era* between 2010-2013, including at the 2011 International Symposium on Memory Management and ACM SIGPLAN Workshop on Memory System Performance and Correctness [56]. Section VII of this article is a much condensed and slightly revised version of the introduction of an invited article that is to appear in a special issue of the Intel Technology Journal, titled *Error Analysis and Retention-Aware Error Management for NAND Flash Memory* [7].

References

[1] International technology roadmap for semiconductors (ITRS). 2011.
[2] *Hybrid Memory Consortium*, 2012. http://www.hybridmemorycube.org.
[3] C. Alkan et al. Personalized copy-number and segmental duplication maps using next-generation sequencing. In *Nature Genetics*, 2009.
[4] G. Atwood. Current and emerging memory technology landscape. In *Flash Memory Summit*, 2011.
[5] R. Ausavarungnirun et al. Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems. In *ISCA*, 2012.
[6] R. Bryant. Data-intensive supercomputing: The case for DISC. *CMU CS Tech. Report 07-128*, 2007.
[7] Y. Cai et al. Error analysis and retention-aware error management for nand flash memory. To appear in *Intel Technology Journal*, 2013.
[8] Y. Cai et al. FPGA-based solid-state drive prototyping platform. In *FCCM*, 2011.
[9] Y. Cai et al. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *DATE*, 2012.
[10] Y. Cai et al. Flash Correct-and-Refresh: Retention-aware error management for increased flash memory lifetime. In *ICCD*, 2012.
[11] Y. Cai et al. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *ICCD*, 2013.
[12] Y. Cai et al. Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis and modeling. In *DATE*, 2013.
[13] K. Chang et al. HAT: Heterogeneous adaptive throttling for on-chip networks. In *SBAC-PAD*, 2012.
[14] E. Chen et al. Advances and future prospects of spin-transfer torque random access memory. *IEEE Transactions on Magnetics*, 46(6), 2010.
[15] E. Chung et al. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPUs? In *MICRO*, 2010.
[16] J. Condit et al. Better I/O through byte-addressable, persistent memory. In *SOSP*, 2009.
[17] R. Das et al. Application-aware prioritization mechanisms for on-chip networks. In *MICRO*, 2009.
[18] R. Das et al. Aergia: Exploiting packet latency slack in on-chip networks. In *ISCA*, 2010.
[19] R. Das et al. Application-to-core mapping policies to reduce memory system interference in multi-core systems. In *HPCA*, 2013.
[20] G. Dhiman. PDRAM: A hybrid PRAM and DRAM main memory system. In *DAC*, 2009.
[21] X. Dong et al. Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems. In *SC*, 2009.
[22] E. Ebrahimi et al. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. In *ASPLOS*, 2010.
[23] E. Ebrahimi et al. Parallel application memory scheduling. In *MICRO*, 2011.
[24] E. Ebrahimi et al. Prefetch-aware shared-resource management for multi-core systems. In *ISCA*, 2011.
[25] M. Ekman. A robust main-memory compression scheme. In *ISCA*, 2005.
[26] R. Gallager. Low density parity check codes. 1963. MIT Press.
[27] B. Grot et al. Preemptive virtual clock: A flexible, efcient, and cost-effective qos scheme for networks-on-chip. In *MICRO*, 2009.
[28] B. Grot et al. Kilo-NOC: A heterogeneous network-on-chip architecture for scalability and service guarantees. In *ISCA*, 2011.
[29] J. A. Joao et al. Bottleneck identification and scheduling in multithreaded applications. In *ASPLOS*, 2012.
[30] T. L. Johnson et al. Run-time spatial locality detection and optimization. In *MICRO*, 1997.
[31] K. Kim et al. A new investigation of data retention time in truly nanoscaled DRAMs. *IEEE Electron Device Letters*, 30(8), Aug. 2009.

[32] Y. Kim et al. ATLAS: a scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA*, 2010.

[33] Y. Kim et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *MICRO*, 2010.

[34] Y. Kim et al. A case for subarray-level parallelism (SALP) in DRAM. In *ISCA*, 2012.

[35] Y. Koh. NAND Flash Scaling Beyond 20nm. In *IMW*, 2009.

[36] E. Kultursay et al. Evaluating STT-RAM as an energy-efficient main memory alternative. In *ISPASS*, 2013.

[37] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *ISCA*, 1998.

[38] B. C. Lee et al. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ISCA*, 2009.

[39] B. C. Lee et al. Phase change memory architecture and the quest for scalability. *Communications of the ACM*, 53(7):99–106, 2010.

[40] B. C. Lee et al. Phase change technology and the future of main memory. *IEEE Micro (Top Picks Issue)*, 30(1), 2010.

[41] C. J. Lee et al. Prefetch-aware DRAM controllers. In *MICRO*, 2008.

[42] D. Lee et al. Tiered-latency DRAM: A low latency and low cost DRAM architecture. In *HPCA*, 2013.

[43] C. Lefurgy et al. Energy management for commercial servers. In *IEEE Computer*, 2003.

[44] J. Liu et al. An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. To appear in *ISCA*, 2013.

[45] J. Liu et al. RAIDR: Retention-aware intelligent DRAM refresh. In *ISCA*, 2012.

[46] G. Loh. 3D-stacked memory architectures for multi-core processors. In *ISCA*, 2008.

[47] A. Maislos et al. A new era in embedded flash memory. In *FMS*, 2011.

[48] J. Mandelman et al. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). In *IBM JR&D*, 2002.

[49] J. Meza et al. A case for small row buffers in non-volatile main memories. In *ICCD*, 2012.

[50] J. Meza et al. Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management. *IEEE CAL*, 2012.

[51] J. Meza et al. A case for efficient hardware-software cooperative management of storage and memory. In *WEED*, 2013.

[52] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX Security*, 2007.

[53] T. Moscibroda and O. Mutlu. Distributed order scheduling and its application to multi-core DRAM controllers. In *PODC*, 2008.

[54] S. Muralidhara et al. Reducing memory interference in multi-core systems via application-aware memory channel partitioning. In *MICRO*, 2011.

[55] O. Mutlu. Asymmetry everywhere (with automatic resource management). In *CRA Workshop on Adv. Comp. Arch. Research*, 2010.

[56] O. Mutlu et al. Memory systems in the many-core era: Challenges, opportunities, and solution directions. In *ISMM*, 2011. http://users.ece.cmu.edu/~omutlu/pub/onur-ismm-mspc-keynote-june-5-2011-short.pptx.

[57] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *MICRO*, 2007.

[58] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In *ISCA*, 2008.

[59] G. Nychis et al. Next generation on-chip networks: What kind of congestion control do we need? In *HotNets*, 2010.

[60] G. Nychis et al. On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects. In *SIGCOMM*, 2012.

[61] G. Pekhimenko et al. Base-delta-immediate compression: A practical data compression mechanism for on-chip caches. In *PACT*, 2012.

[62] G. Pekhimenko et al. Linearly compressed pages: A main memory compression framework with low complexity and low latency. *CMU SAFARI Tech. Report*, 2012.

[63] M. K. Qureshi et al. Line distillation: Increasing cache capacity by filtering unused words in cache lines. In *HPCA*, 2007.

[64] M. K. Qureshi et al. Enhancing lifetime and security of phase change memories via start-gap wear leveling. In *MICRO*, 2009.

[65] M. K. Qureshi et al. Scalable high performance main memory system using phase-change memory technology. In *ISCA*, 2009.

[66] S. Raoux et al. Phase-change random access memory: A scalable technology. *IBM JR&D*, 52, Jul/Sep 2008.

[67] V. Seshadri et al. The evicted-address filter: A unified mechanism to address both cache pollution and thrashing. In *PACT*, 2012.

[68] V. Seshadri et al. RowClone: Fast and efficient In-DRAM copy and initialization of bulk data. *CMU SAFARI Tech. Report*, 2013.

[69] L. Subramanian et al. MISE: Providing performance predictability and improving fairness in shared main memory systems. In *HPCA*, 2013.

[70] M. A. Suleman et al. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS*, 2009.

[71] T. Treangen and S. Salzberg. Repetitive DNA and next-generation sequencing: computational challenges and solutions. In *Nature Reviews Genetics*, 2012.

[72] A. Udipi et al. Rethinking DRAM design and organization for energy-constrained multi-cores. In *ISCA*, 2010.

[73] H.-S. P. Wong. Phase change memory. In *Proceedings of the IEEE*, 2010.

[74] H. Xin et al. Accelerating read mapping with FastHASH. In *BMC Genomics*, 2013.

[75] J. Yang et al. Frequent value compression in data caches. In *MICRO-33*, 2000.

[76] D. Yoon et al. Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In *ISCA*, 2011.

[77] D. Yoon et al. The dynamic granularity memory system. In *ISCA*, 2012.

[78] H. Yoon et al. Row buffer locality aware caching policies for hybrid memories. In *ICCD*, 2012.

[79] H. Yoon et al. Data mapping and buffering in multi-level cell memory for higher performance and energy efficiency. *CMU SAFARI Tech. Report*, 2013.