Chapter 6
Main Memory Scaling: Challenges and Solution Directions

Onur Mutlu, Carnegie Mellon University

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 chapter, 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 describe three major solution directions: 1) enabling new DRAM architectures, functions, interfaces, and better integration of the DRAM and the rest of the system (an approach we call *system-DRAM co-design*), 2) designing a memory system that employs emerging non-volatile memory technologies and takes advantage of multiple different technologies (i.e., *hybrid memory systems*), 3) providing predictable performance and QoS to applications sharing the memory system (i.e., *QoS-aware memory systems*). We also briefly describe our ongoing related work in combating scaling challenges of NAND flash memory.

6.1 Introduction

Main memory is a critical component of all computing systems, employed in server, embedded, desktop, mobile and sensor environments. 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 greatly exacerbate the memory system bottleneck.

6.2 Trends: Systems, Applications, Technology

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 [69]. More and increasingly heterogeneous processing cores and agents/clients are sharing the memory system [6, 21, 107, 45, 46, 36, 23], leading to increasing demand for memory capacity and bandwidth along with a relatively new demand for predictable performance and quality of service (QoS) from the memory system [81, 87, 106].

On the *applications front*, important applications are usually very data intensive and are becoming increasingly so [8], 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 [109, 4, 115] 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 to improve efficiency, 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 [106, 108].

On the *technology front*, two major trends profoundly affect memory systems. First, there is increasing difficulty scaling the well-established charge-based memory technologies, such as DRAM [77, 53, 40, 5, 62, 1] and flash memory [59, 76, 10, 11, 14], 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) [100, 113, 62, 63, 98], spin-transfer torque magnetic memory (STT-MRAM) [19, 60] or resistive RAM (RRAM) [114] 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 [80]. 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 or tolerated.

6.3 Requirements: Traditional and New

System architects and users have always wanted more from the memory system: high performance (ideally, zero latency and infinite bandwidth), infinite capacity, all at zero cost! The aforementioned trends do not only exacerbate and morph the above requirements, but also add some new requirements. We classify the requirements from the memory system into two categories: *exacerbated traditional requirements* and *(relatively) new requirements*.

The traditional requirements of performance, capacity, and cost are greatly exacerbated today due to increased pressure on the memory system, consolidation of multiple applications/agents sharing the memory system, and difficulties in DRAM technology and density scaling. In terms of *performance*, two aspects have changed. First, today's systems and applications not only require low latency and high bandwidth (as traditional memory systems have been optimized for), but they also require new techniques to manage and control memory interference between different cores, agents, and applications that share the memory system [81, 87, 106, 26, 108] in order to provide high system performance as well as predictable performance (or quality of service) to different applications [106]. Second, there is a need for increased memory bandwidth for many applications as the placement of more cores and agents on chip make the memory pin bandwidth an increasingly precious resource that determines system performance [41], especially for memory-bandwidth-intensive workloads, such as GPGPUs [48, 47], heterogeneous systems [6], and consolidated workloads [87, 44, 43]. In terms of *capacity*, the need for memory capacity is greatly increasing due to the placement of multiple data-intensive applications on the same chip and continued increase in the data sets of important applications. One recent work showed that given that the core count is increasing at a faster rate than DRAM capacity, the expected memory capacity per core is to drop by 30% every two years [70], an alarming trend since much of today's software innovations and features rely on increased memory capacity. In terms of *cost*, increasing difficulty in DRAM technology scaling poses a difficult challenge to building higher density (and, as a result, lower cost) main memory systems. Similarly, cost-effective options for providing high reliability and increasing memory bandwidth are needed to scale the systems proportionately with the reliability and data throughput needs of today's data-intensive applications. Hence, the three traditional requirements of performance, capacity, and cost have become exacerbated.

The relatively new requirements from the main memory system are threefold. First, *technology scalability*: there is a new need for finding a technology that is much more scalable than DRAM in terms of capacity, energy, and cost, as described earlier. As DRAM continued to scale well from the above-100-nm to 30-nm technology nodes, the need for finding a more scalable technology was not a prevalent problem. Today, with the significant circuit and device scaling challenges DRAM has been facing below the 30-nm node, it is. Second, there is a relatively new need for providing *performance predictability and QoS* in the shared main memory system. As single-core systems were dominant and memory bandwidth and capacity were much less of a shared resource in the past, the need for predictable perfor-

mance was much less apparent or prevalent [81]. Today, with increasingly more cores/agents on chip sharing the memory system and increasing amounts of workload consolidation, memory fairness, predictable memory performance, and techniques to mitigate memory interference have become first-class design constraints. Third, there is a great need for much higher *energy/power/bandwidth efficiency* in the design of the main memory system. Higher efficiency in terms of energy, power, and bandwidth enables the design of much more scalable systems where main memory is shared between many agents, and can enable new applications in almost all domains where computers are used. Arguably, this is not a new need today, but we believe it is another first-class design constraint that has not been as traditional as performance, capacity, and cost.

6.4 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 this chapter 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. As much as possible, we will give examples of such solutions and directions.

6.5 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 [5, 77, 53, 40, 62, 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 [71, 72, 17],

2) improving DRAM parallelism/bandwidth [57, 17], latency [68], and energy efficiency [57, 68, 71],

3) improving reliability of DRAM at low cost [90, 75, 58, 51],

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 [79, 117, 94, 95, 110],

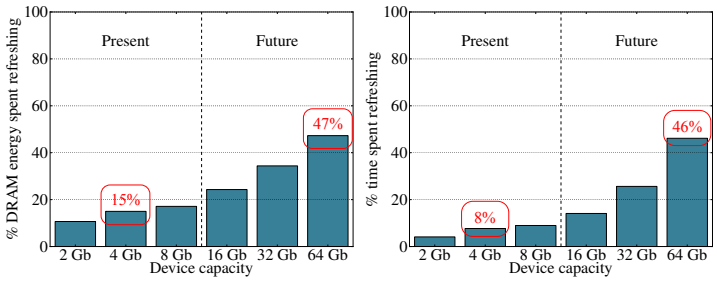
5) minimizing data movement between DRAM and processing elements, which causes high latency, energy, and bandwidth consumption [102].

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 [74, 2, 111] and increasing cost of scaling DRAM solely via circuit-level approaches [77, 53, 40], enable such a co-design to become increasingly more feasible. We proceed to provide several examples from our recent research that tackle the problems of refresh, parallelism, latency, and energy efficiency.

6.5.1 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 [71] 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 (as shown in Figure 6.1). 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 [54, 72, 51] (e.g., only ~1000 rows in 32GB DRAM require to be refreshed more frequently than every 256ms). Retention-Aware Intelligent DRAM Refresh (RAIDR) [71] exploits this observation: it groups DRAM rows into bins (implemented as Bloom filters [7] 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. [71].

Like RAIDR, other approaches have also been proposed to take advantage of the retention time variation of cells across a DRAM chip. For example, some works proposed refreshing weak rows more frequently at a per-row granularity, others proposed not using memory rows with low retention times, and yet others suggested



(a) Power consumption **(b) Throughput Loss**

Fig. 6.1. Impact of refresh in current (DDR3) and projected DRAM devices. Reproduced from [71].

mapping critical data to cells with longer retention times such that critical data is not lost [112, 42, 73, 93, 52, 3] – see [71, 72] for a discussion of such techniques. Such approaches that exploit non-uniform retention times across DRAM 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. [72] 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, suggest 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 a DRAM cell 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. Khan et al.'s recent work [51] provides more analysis of the effectiveness of conventional error mitigation mechanisms for DRAM retention failures and proposes *online retention time profiling* as a solution for identifying retention times of DRAM cells as a potentially promising approach in future DRAM systems.

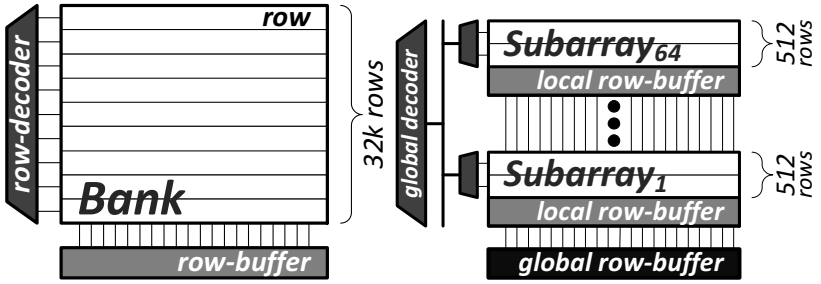
Looking forward, we believe that increasing cooperation between the DRAM device and the DRAM controller as well as other parts of the system, including system software, is needed to communicate information about *weak* (or, unreliable) cells and the characteristics of different rows or physical memory 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 [10, 12]. We envision a *DRAM Translation Layer (DTL)*, not unlike the *Flash Translation Layer (FTL)* of today in spirit (which is decoupled from the processor and performs a wide variety of management functions for flash memory, including error correction, garbage collection, read/write scheduling, etc.), can enable better scaling of DRAM memory into the future by not only enabling easier error management but also opening up new opportunities to perform computation and mapping close to memory. This can become especially feasible in the presence of the trend of combining the DRAM controller and DRAM via 3D stacking. What should the interface be to such a layer and what should be performed in the DTL are promising areas of future research.

6.5.2 Improving DRAM Parallelism

A key limiter of DRAM parallelism is bank conflicts. Today, a bank is the finest-granularity independently accessible memory unit in DRAM. If two accesses go to the same bank, one has to *completely* wait for the other to finish before it can be started (see Figure 6.2). We have recently developed mechanisms, called SALP (subarray level parallelism) [57], that exploit the internal subarray structure of the DRAM bank (Figure 6.2) to *mostly* parallelize two requests that access the same DRAM bank. The key idea is to reduce the hardware sharing between DRAM subarrays 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 a DRAM bank 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.

Note that other approaches to improving DRAM parallelism especially in the presence of refresh and long write latencies are also promising to investigate. Chang et al. [17] discuss mechanisms to improve the parallelism between reads and writes, and Kang et al. [50] discuss the use of SALP as a way of tolerating long write latencies to DRAM, which they identify as one of the three key scaling challenges for DRAM, amongst refresh and variable retention time. We refer the reader to these works for more information about these parallelization techniques.



(a) Logical abstraction of a bank **(b) Implementation of a DRAM bank**

Fig. 6.2. DRAM Bank Organization. Reproduced from [57].

6.5.3 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. As shown in Figure 6.3, DRAM chip capacity has increased by approximately 16X in 12 years while the DRAM latency reduced by only approximately 20%. This is the result of a deliberate choice to maximize capacity of a DRAM chip while minimizing its cost. We believe this choice needs to be revisited in the presence of at least two key trends. First, DRAM latency is becoming more important especially for response-time critical workloads that require QoS guarantees [28]. Second, DRAM capacity is becoming very hard to scale and as a result manufacturers likely need to provide new values for the DRAM chips, leading to more incentives for the production of DRAMs that are optimized for objectives other than mainly capacity maximization.

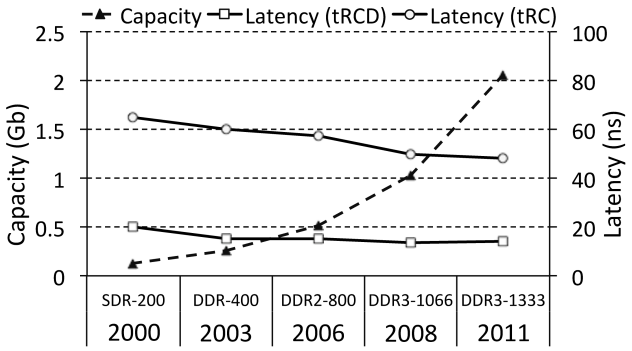


Fig. 6.3. DRAM Capacity & Latency Over Time. Reproduced from [68].

To mitigate the high area overhead of DRAM sensing structures, commodity DRAMs (shown in Figure 6.4a) 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 (shown in Figure 6.4b) 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) [68], shown in Figure 6.4c, 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 [84] in the memory system can lead to other novel mechanisms that can overcome difficult contradictory tradeoffs in design.

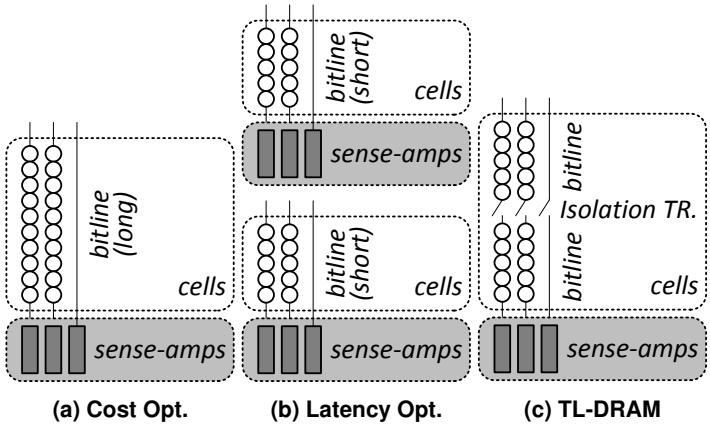


Fig. 6.4. Cost Optimized Commodity DRAM (a), Latency Optimized DRAM (b), Tiered-Latency DRAM (c). Reproduced from [68].

Another promising approach to reduce DRAM energy is the use of dynamic voltage and frequency scaling (DVFS) in main memory [27, 29]. David et al. [27] make the observation that at low memory bandwidth utilization, lowering memory frequency/voltage does not significantly alter memory access latency. Relatively recent works have shown that adjusting memory voltage and frequency based on predicted memory bandwidth utilization can provide significant energy savings on both real [27] and simulated [29] systems. Going forward, memory DVFS can enable dynamic heterogeneity in DRAM channels leading to new customization and optimization mechanisms. Also promising is the investigation of more fine-grained

power management methods within the DRAM rank and chips for both active and idle low power modes.

6.5.4 Exporting Bulk Data Operations to DRAM

Today's systems waste significant amount of energy, DRAM bandwidth and time (as well as valuable on-chip cache space) by sometimes 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, as we have shown in recent work [102]. 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 (~11 times), leading to an energy reduction of ~74 times and *no* wastage of DRAM data bus bandwidth [102]. 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 (shown at a high level in 6.5). Doing so reduces the latency of a 4KB page copy operation from ~1000ns to less than 100ns in an existing DRAM chip. Applications that have significant page copy and initialization lead to large system performance and energy efficiency improvements [102]. Future software can be designed in ways that can take advantage of such fast page copy and initialization operations, leading to benefits that may not be apparent in today's software that tends to minimize such operations due to their current high cost.

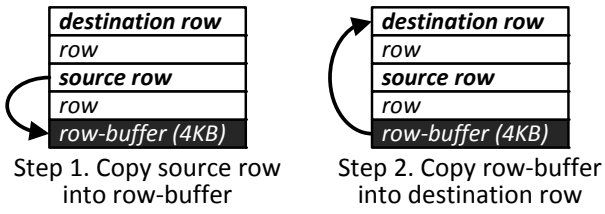


Fig. 6.5. High-level idea behind RowClone's in-DRAM page copy mechanism.

Going forward, we believe acceleration of other bulk data movement and computation operations in or very close to DRAM, via similar low-cost architectural support mechanisms, can enable promising savings in system energy, latency, and bandwidth. Given the trends and requirements described in Section 6.2, 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 [84].

6.5.5 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 [117, 118, 79, 78, 110, 101, 61, 96, 49]. In addition, much of the data stored in memory has significant redundancy [116, 35, 94, 95]. 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 [94] and memory compression [95] techniques and frameworks are promising, providing high compression ratios at low complexity and latency. For example, the key idea of *Base-Delta-Immediate compression* [94] 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 (or even potentially vector concatenation). 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.

6.5.6 Co-Designing DRAM Controllers and Processor-Side Resources

Since memory bandwidth is a precious resource, coordinating the decisions made by processor-side resources better with the decisions made by memory controllers to maximize memory bandwidth utilization and memory locality is a promising area of more efficiently utilizing DRAM. Lee et al. [67] and Stuecheli et al. [105] both

show that orchestrating last-level cache writebacks such that dirty cache lines to the same row are evicted together from the cache improves DRAM row buffer locality of write accesses, thereby improving system performance. Going forward, we believe such coordinated techniques between the processor-side resources and memory controllers will become increasingly more effective as DRAM bandwidth becomes even more precious. Mechanisms that predict and convey slack in memory requests [25], that orchestrate the on-chip scheduling of memory requests to improve memory bank parallelism [65] and that reorganize cache metadata for more efficient bulk (DRAM row granularity) tag lookups [103] can also enable more efficient memory bandwidth utilization.

6.6 Challenge 2: Emerging Memory Technologies

While DRAM technology 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 [62, 63, 64] for PCM and in [60] 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 [120], 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 6.5.1), 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, an important 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 [62, 63, 64] 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. [62, 63, 64]. We have also reached a similar conclusion upon evaluation of the complete replacement of DRAM with STT-MRAM [60]: 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 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 [78] 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. [78] show 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.

6.6.1 Hybrid Main Memory

A hybrid main memory system [98, 30, 79, 119] consists of multiple different technologies or multiple different types of the same technology 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. In other words, we would like to exercise the advantages of each technology as much as possible while hiding the disadvantages of any technology. Potential technologies include DRAM, 3D-stacked DRAM, embedded DRAM, PCM, STT-MRAM, other resistive memories, flash memory, forms of DRAM that are optimized for different metrics and purposes, etc. An example hybrid main memory system consisting of a large amount of PCM as main memory and a small amount of DRAM as its cache is depicted in Figure 6.6.

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 [79, 119, 98, 30, 99], among other works recently published in the computer architecture community. For example, Yoon et al. [119] make 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 while keeping data that frequently hits in row buffers in PCM. PCM also has much higher write latency/power than read latency/power: to take this into account, the alloca-

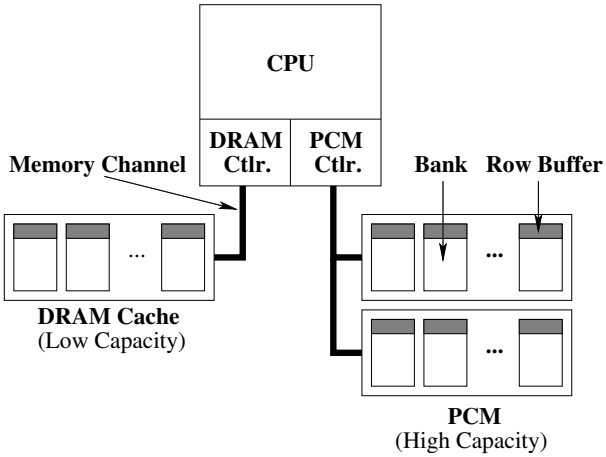


Fig. 6.6. An example hybrid main memory system organization using PCM and DRAM chips. Reproduced from [119].

tion/caching policy is biased such that pages that are written to more likely stay in DRAM [119].

Note that hybrid memory need not consist of completely different underlying technologies. A promising approach is to combine multiple different DRAM chips, optimized for different purposes. For example, recent works proposed the use of low-latency and high-latency DIMMs in separate memory channels and allocating performance-critical data to low-latency DIMMs to improve performance and energy-efficiency at the same time [18], or the use of highly-reliable DIMMs (protected with ECC) and unreliable DIMMs in separate memory channels and allocating error-vulnerable data to highly-reliable DIMMs to maximize server availability while minimizing server memory cost [75]. We believe these approaches are quite promising for scaling the DRAM technology into the future by specializing DRAM chips for different purposes. These approaches that exploit heterogeneity do increase system complexity but that complexity may be warranted if it is lower than the complexity of scaling DRAM chips using the same optimization techniques the DRAM industry has been using so far.

6.6.2 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 (see, for example, [31, 22]. Researching how to adapt applications and system software to utilize fast, byte-addressable non-volatile main memory is an important research direction to pursue [80].

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 [20], 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 [97, 104]. Securing non-volatile main memory is therefore an important systems challenge.

6.6.3 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 [80]. 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 memory).

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 memory energy consumption and degrading performance by an order of magnitude in some data-intensive workloads, as we showed in recent work [80]. 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. [80] 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. Figure 6.7 depicts an example PMM, programmed using a load/store interface (with persistent objects) and managing an array of heterogeneous devices.

```
1 int main(void) {
2   // data in file.dat is persistent
3   FILE myData = "file.dat";
4   myData = new int[64];
5 }
6 void updateValue(int n, int value) {
7   FILE myData = "file.dat";
8   myData[n] = value; // value is persistent
9 }
```

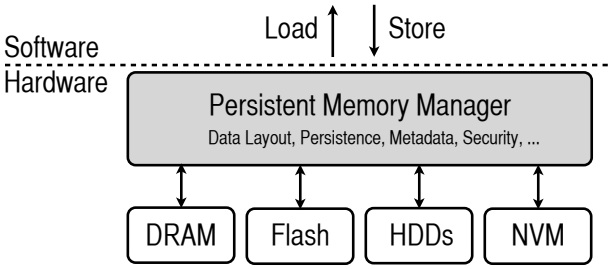


Fig. 6.7. An example Persistent Memory Manager (PMM). Reproduced from [80].

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 greatly reduce 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 [80].

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.

6.7 Challenge 3: Predictable Performance

Since memory is a shared resource between multiple cores (or, agents, threads, or applications and virtual machines), 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 (e.g., [81, 87, 88]) 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 [81], low system performance [87, 88], and an inability to satisfy performance requirements [87, 106, 32], which makes the system uncontrollable and unpredictable. In fact, an application's performance depends on what other applications it is sharing resources with: an application can sometimes have very high performance and at 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 [87, 88, 82, 55, 56, 83, 6, 106, 66, 33], interconnects [24, 25, 38, 39, 16, 91, 92, 26], and entire memory systems including caches, interconnect, and memory [32, 34, 26]. These works enhanced our understanding of memory interference in multi-core and heterogeneous systems and provide viable and effective mechanisms that improve overall system performance, while also providing a fairness substrate that can enable fair memory service, which can be configured to enforce different application priorities.

A promising direction going forward is to devise mechanisms that are effective and accurate at 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. [106] provides a simple method, called MISE (Memory-interference Induced Slowdown Estimation), for estimating application slowdowns in the presence of main memory interference. We observe that an application's memory request service rate is a good proxy for its performance, as depicted in Figure 6.8, which shows the measured performance versus memory request service rate for three applications on a real system [106]. 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. Figure 6.9 shows the actual versus predicted slowdowns over time, for *astar*, a representative application from among the many applications examined, when it is run alongside three other applications on a simulated 4-core 1-channel system. As we can see, MISE's slowdown estimates track the actual slowdown closely. Extending such simple techniques like MISE to the entire memory and storage system is a promising area of future research in both homogeneous and heterogeneous systems. Devising memory devices and architectures that can support predictability and QoS also appears promising.

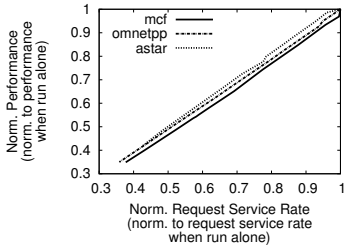


Fig. 6.8. Request service rate vs. performance. Reproduced from [106].

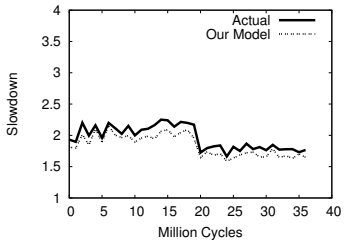


Fig. 6.9. Actual vs. predicted slowdowns. Reproduced from [106].

6.8 Aside: Flash Memory 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 [9]. 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 [10], 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) [11]. 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 [14]. 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 [37], 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 [13], 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. [13].

Most recently, to improve flash memory lifetime, we have developed a mechanism called Neighbor-Cell Assisted Correction (NAC) [15], which uses the value information of cells in a neighboring page to correct errors found on a page when reading. This mechanism takes advantage of the new empirical observation that identifying the value stored in the immediate-neighbor cell makes it easier to determine the data value stored in the cell that is being read. The key idea is to re-read a flash memory page that fails error correction codes (ECC) with the set of read reference voltage values corresponding to the conditional threshold voltage distribution assuming a neighbor cell value and use the re-read values to correct the cells that have neighbors with that value. Our simulations show that NAC effectively improves flash memory lifetime by 33% while having no (at nominal lifetime) or very modest (less than 5% at extended lifetime) performance overhead.

Going forward, we believe more accurate and detailed characterization of flash memory error mechanisms are 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 [11], read reference voltage prediction [13], described earlier, are early forms of such predictive error tolerance mechanisms.

6.9 Conclusion

We have described several research directions and ideas to enhance memory scaling via system and architecture-level approaches. A promising approach is the co-design of memory and other system components to enable better system optimization. Enabling better cooperation across multiple levels of the computing stack, including software, microarchitecture, and devices can help scale the memory system by exposing more of the memory device characteristics to higher levels of the system stack such that the latter can tolerate and exploit such characteristics. Finally, heterogeneity in the design of the memory system can help overcome the memory scaling challenges at the device level by enabling better specialization of the memory system and its dynamic adaptation to different demands of various applications. We believe such approaches will become increasingly important and effective as the underlying memory technology nears its scaling limits at the physical level and envision a near future full of innovation in main memory architecture, enabled by the co-design of the system and main memory.

Acknowledgments

I would like to thank my PhD students Rachata Ausavarungnirun and Lavanya Subramanian for logistic help in preparing this chapter and earlier versions of it. Many thanks to all my students in my SAFARI research group and collaborators at Carnegie Mellon as well as other universities, whom all contributed to the works outlined in this chapter. Thanks also go to my research group's industrial sponsors over the past six years, including AMD, HP Labs, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Samsung. Some of the research reported here was also partially supported by GSRC, Intel URO Memory Hierarchy Program, Intel Science and Technology Center on Cloud Computing, NIH, NSF, and SRC.

This chapter is a significantly extended and revised version of an invited paper that appeared at the 5th International Memory Workshop [85], which was also presented at MemCon 2013 [86]. Part of the structure of this chapter is based on an evolving set of talks I have delivered at various venues on *Scaling the Memory System in the Many-Core Era* and *Rethinking Memory System Design for Data-Intensive Computing* between 2010-2014, including invited talks at the 2011 International Symposium on Memory Management and ACM SIGPLAN Workshop on Memory System Performance and Correctness [89] and the 2012 DAC More Than Moore Technologies Workshop. Section 6.8 of this article is a condensed and slightly revised version of the introduction of an invited article that appeared in a special issue of the Intel Technology Journal, titled *Error Analysis and Retention-Aware Error Management for NAND Flash Memory* [12].

References

1. International technology roadmap for semiconductors (ITRS) (2011)
2. Hybrid Memory Consortium (2012). `http://www.hybridmemorycube.org`
3. Ahn, J.H., et al.: Adaptive self refresh scheme for battery operated high-density mobile dram applications. In: ASSCC (2006)
4. Alkan, C., et al.: Personalized copy-number and segmental duplication maps using next-generation sequencing. In: Nature Genetics (2009)
5. Atwood, G.: Current and emerging memory technology landscape. In: Flash Memory Summit (2011)
6. Ausavarungnirun, R., et al.: Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems. In: ISCA (2012)
7. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)
8. Bryant, R.: Data-intensive supercomputing: The case for DISC. CMU CS Tech. Report 07-128 (2007)
9. Cai, Y., et al.: FPGA-based solid-state drive prototyping platform. In: FCCM (2011)
10. Cai, Y., et al.: Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In: DATE (2012)
11. Cai, Y., et al.: Flash Correct-and-Refresh: Retention-aware error management for increased flash memory lifetime. In: ICCD (2012)
12. Cai, Y., et al.: Error analysis and retention-aware error management for nand flash memory. Intel technology Journal **17**(1) (2013)

13. Cai, Y., et al.: Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In: ICCD (2013)
14. Cai, Y., et al.: Threshold voltage distribution in MLC NAND flash memory: Characterization, analysis and modeling. In: DATE (2013)
15. Cai, Y., et al.: Neighbor-cell assisted error correction for MLC NAND flash memories. In: SIGMETRICS (2014)
16. Chang, K., et al.: HAT: Heterogeneous adaptive throttling for on-chip networks. In: SBAC-PAD (2012)
17. Chang, K., et al.: Improving DRAM performance by parallelizing refreshes with accesses. In: HPCA (2014)
18. Chatterjee, N., et al.: Leveraging heterogeneity in DRAM main memories to accelerate critical word access. In: MICRO (2012)
19. Chen, E., et al.: Advances and future prospects of spin-transfer torque random access memory. IEEE Transactions on Magnetics **46**(6) (2010)
20. Chhabra, S., Solihin, Y.: i-nvmm: a secure non-volatile main memory system with incremental encryption. In: ISCA (2011)
21. Chung, E., et al.: Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPUs? In: MICRO (2010)
22. Condit, J., et al.: Better I/O through byte-addressable, persistent memory. In: SOSP (2009)
23. Craeynest, K.V., et al.: Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In: ISCA (2012)
24. Das, R., et al.: Application-aware prioritization mechanisms for on-chip networks. In: MICRO (2009)
25. Das, R., et al.: Aergia: Exploiting packet latency slack in on-chip networks. In: ISCA (2010)
26. Das, R., et al.: Application-to-core mapping policies to reduce memory system interference in multi-core systems. In: HPCA (2013)
27. David, H., et al.: Memory power management via dynamic voltage/frequency scaling. In: ICAC (2011)
28. Dean, J., Barroso, L.A.: The tail at scale. Communications of the ACM **56**(2), 74–80 (2013)
29. Deng, Q., et al.: MemScale: active low-power modes for main memory. In: ASPLOS (2011)
30. Dhiman, G.: PDRAM: A hybrid PRAM and DRAM main memory system. In: DAC (2009)
31. Dong, X., et al.: Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exascale systems. In: SC (2009)
32. Ebrahimi, E., et al.: Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. In: ASPLOS (2010)
33. Ebrahimi, E., et al.: Parallel application memory scheduling. In: MICRO (2011)
34. Ebrahimi, E., et al.: Prefetch-aware shared-resource management for multi-core systems. In: ISCA (2011)
35. Ekman, M.: A robust main-memory compression scheme. In: ISCA (2005)
36. Eyerman, S., Eeckhout, L.: Modeling critical sections in amdahl's law and its implications for multicore design. In: ISCA (2010)
37. Gallager, R.: Low density parity check codes (1963). MIT Press
38. Grot, B., et al.: Preemptive virtual clock: A flexible, efcient, and cost-effective qos scheme for networks-on-chip. In: MICRO (2009)
39. Grot, B., et al.: Kilo-NOC: A heterogeneous network-on-chip architecture for scalability and service guarantees. In: ISCA (2011)
40. Hong, S.: Memory technology trend and future challenges. In: IEDM (2010)
41. Ipek, E., Mutlu, O., Martinez, J.F., Caruana, R.: Self-optimizing memory controllers: A reinforcement learning approach. In: ISCA (2008)
42. Isen, C., John, L.K.: Eskimo: Energy savings using semantic knowledge of inconsequential memory occupancy for dram subsystem. In: MICRO (2009)
43. Iyer, R.: CQoS: a framework for enabling QoS in shared caches of CMP platforms. In: ICS (2004)
44. Iyer, R., et al.: QoS policies and architecture for cache/memory in cmp platforms. In: SIGMETRICS (2007)

45. Joao, J.A., et al.: Bottleneck identification and scheduling in multithreaded applications. In: ASPLOS (2012)
46. Joao, J.A., et al.: Utility-based acceleration of multithreaded applications on asymmetric cmps. In: ISCA (2013)
47. Jog, A., et al.: Orchestrated scheduling and prefetching for GPGPUs. In: ISCA (2013)
48. Jog, A., et al.: OWL: Cooperative thread array aware scheduling techniques for improving GPGPU performance. In: ASPLOS (2013)
49. Johnson, T.L., et al.: Run-time spatial locality detection and optimization. In: MICRO (1997)
50. Kang, U., et al.: Co-architecting controllers and DRAM to enhance DRAM process scaling. In: The Memory Forum (2014)
51. Khan, S., et al.: The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study. In: SIGMETRICS (2014)
52. Kim, J., Papaefthymiou, M.C.: Dynamic memory design for low data-retention power. In: PATMOS (2000)
53. Kim, K.: Future memory technology: challenges and opportunities. In: VLSI-TSA (2008)
54. Kim, K., et al.: A new investigation of data retention time in truly nanoscaled DRAMs. IEEE Electron Device Letters **30**(8) (2009)
55. Kim, Y., et al.: ATLAS: a scalable and high-performance scheduling algorithm for multiple memory controllers. In: HPCA (2010)
56. Kim, Y., et al.: Thread cluster memory scheduling: Exploiting differences in memory access behavior. In: MICRO (2010)
57. Kim, Y., et al.: A case for subarray-level parallelism (SALP) in DRAM. In: ISCA (2012)
58. Kim, Y., et al.: Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In: ISCA (2014)
59. Koh, Y.: NAND Flash Scaling Beyond 20nm. In: IMW (2009)
60. Kultursay, E., et al.: Evaluating STT-RAM as an energy-efficient main memory alternative. In: ISPASS (2013)
61. Kumar, S., Wilkerson, C.: Exploiting spatial locality in data caches using spatial footprints. In: ISCA (1998)
62. Lee, B.C., et al.: Architecting Phase Change Memory as a Scalable DRAM Alternative. In: ISCA (2009)
63. Lee, B.C., et al.: Phase change memory architecture and the quest for scalability. Communications of the ACM **53**(7), 99–106 (2010)
64. Lee, B.C., et al.: Phase change technology and the future of main memory. IEEE Micro (Top Picks Issue) **30**(1) (2010)
65. Lee, C.J., Narasiman, V., Mutlu, O., Patt, Y.N.: Improving memory bank-level parallelism in the presence of prefetching. In: MICRO (2009)
66. Lee, C.J., et al.: Prefetch-aware DRAM controllers. In: MICRO (2008)
67. Lee, C.J., et al.: DRAM-aware last-level cache writeback: Reducing write-caused interference in memory systems. Tech. Rep. TR-HPS-2010-002, HPS (2010)
68. Lee, D., et al.: Tiered-latency DRAM: A low latency and low cost DRAM architecture. In: HPCA (2013)
69. Lefurgy, C., et al.: Energy management for commercial servers. In: IEEE Computer (2003)
70. Lim, K., et al.: Disaggregated memory for expansion and sharing in blade servers. In: ISCA (2009)
71. Liu, J., et al.: RAIDR: Retention-aware intelligent DRAM refresh. In: ISCA (2012)
72. Liu, J., et al.: An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms. In: ISCA (2013)
73. Liu, S., et al.: Flikker: saving dram refresh-power through critical data partitioning. In: ASPLOS (2011)
74. Loh, G.: 3D-stacked memory architectures for multi-core processors. In: ISCA (2008)
75. Luo, Y., et al.: Characterizing application memory error vulnerability to optimize data center cost via heterogeneous-reliability memory. In: DSN (2014)
76. Maislos, A., et al.: A new era in embedded flash memory. In: FMS (2011)

77. Mandelman, J., et al.: Challenges and future directions for the scaling of dynamic random-access memory (DRAM). In: IBM JR&D (2002)
78. Meza, J., et al.: A case for small row buffers in non-volatile main memories. In: ICCD (2012)
79. Meza, J., et al.: Enabling efficient and scalable hybrid memories using fine-granularity DRAM cache management. IEEE CAL (2012)
80. Meza, J., et al.: A case for efficient hardware-software cooperative management of storage and memory. In: WEED (2013)
81. Moscibroda, T., Mutlu, O.: Memory performance attacks: Denial of memory service in multi-core systems. In: USENIX Security (2007)
82. Moscibroda, T., Mutlu, O.: Distributed order scheduling and its application to multi-core DRAM controllers. In: PODC (2008)
83. Muralidhara, S., et al.: Reducing memory interference in multi-core systems via application-aware memory channel partitioning. In: MICRO (2011)
84. Mutlu, O.: Asymmetry everywhere (with automatic resource management). In: CRA Workshop on Adv. Comp. Arch. Research (2010)
85. Mutlu, O.: Memory scaling: A systems architecture perspective. In: IMW (2013)
86. Mutlu, O.: Memory scaling: A systems architecture perspective. In: MemCon (2013)
87. Mutlu, O., Moscibroda, T.: Stall-time fair memory access scheduling for chip multiprocessors. In: MICRO (2007)
88. Mutlu, O., Moscibroda, T.: Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems. In: ISCA (2008)
89. Mutlu, O., 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
90. Nair, P.J., et al.: ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates. In: ISCA (2013)
91. Nychis, G., et al.: Next generation on-chip networks: What kind of congestion control do we need? In: HotNets (2010)
92. Nychis, G., et al.: On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects. In: SIGCOMM (2012)
93. Ohsawa, T., et al.: Optimizing the DRAM refresh count for merged DRAM/logic LSIs. In: ISLPED (1998)
94. Pekhimenko, G., et al.: Base-delta-immediate compression: A practical data compression mechanism for on-chip caches. In: PACT (2012)
95. Pekhimenko, G., et al.: Linearly compressed pages: A main memory compression framework with low complexity and low latency. MICRO (2013)
96. Qureshi, M.K., et al.: Line distillation: Increasing cache capacity by filtering unused words in cache lines. In: HPCA (2007)
97. Qureshi, M.K., et al.: Enhancing lifetime and security of phase change memories via start-gap wear leveling. In: MICRO (2009)
98. Qureshi, M.K., et al.: Scalable high performance main memory system using phase-change memory technology. In: ISCA (2009)
99. Ramos, L.E., et al.: Page placement in hybrid memory systems. In: ICS (2011)
100. Raoux, S., et al.: Phase-change random access memory: A scalable technology. IBM JR&D **52** (2008)
101. Seshadri, V., et al.: The evicted-address filter: A unified mechanism to address both cache pollution and thrashing. In: PACT (2012)
102. Seshadri, V., et al.: RowClone: Fast and efficient In-DRAM copy and initialization of bulk data. MICRO (2013)
103. Seshadri, V., et al.: The dirty-block index. In: ISCA (2014)
104. Song, N.H., et al.: Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping. In: ISCA (2010)
105. Stuecheli, J., et al.: The virtual write queue: Coordinating DRAM and last-level cache policies. In: ISCA-37 (2010)

106. Subramanian, L., et al.: MISE: Providing performance predictability and improving fairness in shared main memory systems. In: HPCA (2013)
107. Suleman, M.A., et al.: Accelerating critical section execution with asymmetric multi-core architectures. In: ASPLOS (2009)
108. Tang, L., et al.: The impact of memory subsystem resource sharing on datacenter applications. In: ISCA (2011)
109. Treangen, T., Salzberg, S.: Repetitive DNA and next-generation sequencing: computational challenges and solutions. In: Nature Reviews Genetics (2012)
110. Udipi, A., et al.: Rethinking DRAM design and organization for energy-constrained multi-cores. In: ISCA (2010)
111. Udipi, A., et al.: Combining memory and a controller with photonics through 3d-stacking to enable scalable and energy-efficient systems. In: ISCA (2011)
112. Venkatesan, R.K., et al.: Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM. In: HPCA (2006)
113. Wong, H.S.P.: Phase change memory. In: Proceedings of the IEEE (2010)
114. Wong, H.S.P.: Metal-oxide rram. In: Proceedings of the IEEE (2012)
115. Xin, H., et al.: Accelerating read mapping with FastHASH. In: BMC Genomics (2013)
116. Yang, J., et al.: Frequent value compression in data caches. In: MICRO-33 (2000)
117. Yoon, D., et al.: Adaptive granularity memory systems: A tradeoff between storage efficiency and throughput. In: ISCA (2011)
118. Yoon, D., et al.: The dynamic granularity memory system. In: ISCA (2012)
119. Yoon, H., et al.: Row buffer locality aware caching policies for hybrid memories. In: ICCD (2012)
120. Yoon, H., et al.: Data mapping and buffering in multi-level cell memory for higher performance and energy efficiency. CMU SAFARI Tech. Report (2013)