DESIGNING EFFICIENT MEMORY FOR FUTURE COMPUTING SYSTEMS

by

Aniruddha N. Udipi

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computing

School of Computing

The University of Utah

May 2012

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Aniruddha N. Udipi**

has been approved by the following supervisory committee members:

Rajeev Balasubramonian , Chair **03/07/2012**
Date Approved

Alan L. Davis , Member **03/07/2012**
Date Approved

Erik L. Brunvand , Member **03/07/2012**
Date Approved

Kenneth S. Stevens , Member **03/07/2012**
Date Approved

Norman P. Jouppi , Member **03/07/2012**
Date Approved

and by **Alan L. Davis** , Chair of

the Department of **Computing**

and by Charles A. Wight, Dean of The Graduate School.

ABSTRACT

The computing landscape is undergoing a major change, primarily enabled by ubiquitous wireless networks and the rapid increase in the use of mobile devices which access a web-based information infrastructure. It is expected that most intensive computing may either happen in servers housed in large datacenters (*warehouse-scale computers*), *e.g.*, cloud computing and other web services, or in many-core high-performance computing (HPC) platforms in scientific labs. It is clear that the primary challenge to scaling such computing systems into the exascale realm is the efficient supply of large amounts of data to hundreds or thousands of compute cores, *i.e.*, building an efficient memory system. Main memory systems are at an inflection point, due to the convergence of several major application and technology trends. Examples include the increasing importance of energy consumption, reduced access stream locality, increasing failure rates, limited pin counts, increasing heterogeneity and complexity, and the diminished importance of cost-per-bit. In light of these trends, the memory system requires a major overhaul. The key to architecting the next generation of memory systems is a combination of the prudent incorporation of novel technologies, and a fundamental rethinking of certain conventional design decisions. In this dissertation, we study every major element of the memory system – the memory chip, the processor-memory channel, the memory access mechanism, and memory reliability, and identify the key bottlenecks to efficiency. Based on this, we propose a novel main memory system with the following innovative features: (i) overfetch-aware re-organized chips, (ii) low-cost silicon photonic memory channels, (iii) largely autonomous memory modules with a packet-based interface to the processor, and (iv) a RAID-based reliability mechanism. Such a system is energy-efficient, high-performance, low-complexity, reliable, and cost-effective, making it ideally suited to meet the requirements of future large-scale computing systems.

To My Parents

CONTENTS

CHAPTERS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I have had the good fortune of having the support of several wonderful people during the course of my Ph.D. It has been both a pleasure and a privilege to work with Rajeev – I couldn't have asked for a better advisor. I learned a great deal from him about academic life and being a good researcher. The humility that has accompanied his success is something I have always admired, and will strive to remember in my own career. Naveen was responsible both for talking me into doing a Ph.D., and for making it a great experience once I got started. He has been a good friend and a great role model, and his advice throughout has been invaluable. My committee members – Al, Norm, Erik, and Ken – provided constant support and encouragement, and I would like to thank them for that. Al and Norm have been co-authors on most of my work, and their detailed feedback at every stage of each project helped immensely. Thanks are also due to Norm for the opportunity of an extended 2-year internship at HP Labs, where the bulk of my dissertation work was completed.

I would like to thank my awesome colleagues in the Utah Arch group – Nil, Manu, Kshitij, Dave, Seth, Manju, and Ali – for countless hours of discussion on pretty much every conceivable topic. They made grad school infinitely more fun. Thanks to my friends and roommates for making my 5 years in Salt Lake City enjoyable. Above all, I am grateful to my family for their constant love and encouragement. They have always been there for me, and I could not have been where I am today without them.

CHAPTER 1

INTRODUCTION

1.1 Emerging Trends

The evolution of the computing landscape is leading to a dichotomy of computers into relatively simple mobile devices, and large server farms, also called *warehouse-scale computers*. Mobile devices are expected primarily to access a web-based information infrastructure, with most intensive computing relegated to servers housed in large datacenters. It is clear that the primary challenge to scaling such computing systems into the exascale realm is the efficient supply of large amounts of data to hundreds or thousands of compute cores. Increasing socket, core, and thread counts, combined with large datasets in modern scientific and commercial workloads will exert extreme pressure on the memory system. Further, these demands will have to be satisfied under increasingly constrained power budgets, while utilizing increasingly unreliable components. The memory interface will thus be operating under challenging conditions, and should be designed carefully in order to provide low latency, high bandwidth, low power, strong fault-tolerance, and extreme scalability. Consider the following technological trends that will strongly influence the design of the next generation of memory systems.

1.1.1 Energy Constraints

Energy consumption in datacenters has been highlighted as a problem of national importance by the Environment Protection Agency (EPA) [2]. A large datacenter can house several thousands of servers, several terabytes of main memory, and several petabytes of storage, consuming up to 30 MW of power [3]. It is estimated that currently, datacenters consume just under 2% of all power generated in the United States, for operation and cooling. This equals approximately 100 Billion kWh of energy (in 2011), at an annual cost of approximately $7.4 Billion [2].

Processors have typically been the largest consumer of energy in large systems. For example, in IBM's midrange servers, they consume 30% of total system power [4]. Memory follows as the next big consumer, accounting for 28% of power in IBM's servers (other studies have put this figure between 25% and 40% [5, 6, 3, 7]). Recently, there has been a trend towards the use of simpler cores in servers, such as the Intel Atom [8], or even low-power ARM cores [9]. Processors are also becoming more energy-proportional [3], being capable of switching to low-power states in a fine-grained manner. As a result, the power bottleneck will shift from the processors to the memory system. To put the numbers in perspective, a single 256 MB memory chip consumes approximately 535 mW [10], whereas a "wimpy" ARM Cortex-A5 core consumes just 72 mW [11]. Therefore, while energy was never a first-order design constraint in prior memory systems, it has certainly emerged as the primary constraint today, and will likely be even more important going forward.

Modern DRAM architectures are ill-suited for energy-efficient operation for several reasons. They are designed to fetch much more data than required, wasting dynamic energy. They also employ coarse-grained power-down tactics to reduce area and cost, but finer grained approaches can further reduce background energy. These problems are exacerbated due to the imposition of access granularity constraints to provide strong reliability. Another major consumer of energy in the memory system is the processor-memory off-chip interconnect, largely due to power-hungry SERDES and pin drivers on both sides of the channel [12]. These issues will have to be effectively addressed by future memory system architectures.

1.1.2 Performance Demands

The size of datasets in modern applications is constantly increasing. The killer commercial applications of tomorrow are expected to be in the field of "big data" information management and analytics, where thousands of petabytes of structured or unstructured data are mined to extract useful insight or information. On the other hand, the scientific computation community is creating more detailed and complex models of natural phenomena, which need to be analyzed at close to real-time speeds to be useful to society. Systems such as RAMCloud [13] are demonstrating the suitability of DRAM main memory to store a majority if not all of the information in such massive data-intensive applications, creating a new approach to datacenter storage. Additionally, increasing transistor counts, combined with novel architectures such as GPGPUs, are significantly increasing the compute power per socket, and as a result, the bandwidth demand. Meanwhile, the number of pins per socket is growing very slowly. The 2007 ITRS Road-map [14] expects a 1.47x increase in the number of pins over an 8-year time-frame – over the same period, Moore's Law dictates at least a 16x increase in the number of cores. The number of memory channels per socket, therefore, will stay mostly constant, limiting the bandwidth per core. Additionally, the number of DIMMs per channel is decreasing due to signal integrity concerns [15]. As a result, the memory capacity per core is projected to drop by about 30% every 2 years, further limiting performance [5]. It is thus clear that requests from many cores will compete to utilize limited pin resources, severely constraining bandwidth, capacity, and latency, and place enormous pressure on the memory system.

Today's main memory architectures, constrained by the inherent characteristics of off-chip electrical interconnects, are unlikely to be able to meet such demands. Novel emerging technologies such as silicon photonics will be essential to build an adequate memory system for the future. The study of these technologies at the architectural level will be important in order to exploit their advantages in a manner that is optimal in terms of energy, performance, and cost.

1.1.3 Reduced Locality

The high degree of multithreading in future multicores [16] implies that memory requests from multiple access streams get multiplexed at the memory controller, thus destroying a large fraction of the available locality. The severity of this problem will increase with increased core and memory controller counts that are expected for future microprocessor chips. This trend is exacerbated by the increased use of aggregated memory pools ("memory blades" comprised of many commodity DIMMs) that serve several CPU sockets in an effort to increase resource utilization [5].

Current DRAM architectures are based on standards set in the early 1990s, when single-core workloads typically exhibited high locality. Consequently, they fetch many kilobytes of data on every access and keep them in open row buffers so that subsequent requests to neighboring data elements can be serviced quickly. This effort yields minimal benefits with today's workloads, and in fact, increases energy consumption significantly. This mandates that future DRAM architectures place a lower priority on locality and a higher priority on parallelism.

1.1.4 Reliability Concerns

Being the backbone of the future web-connected infrastructure, datacenters will necessarily have to be designed to be highly reliable, with low failure rates and high uptime. Users should be able to trust that the services they increasingly rely on will always be available to them [3]. Moreover, running these large server farms can cost service providers millions of dollars per year, and any downtime due to server failure has a large economic impact due to breached Service Level Agreement (SLA) contracts. Recent studies [17] have shown that DRAMs are often plagued with errors and can lead to significant server downtime in datacenters. In fact, DRAM errors are known to be the number one hardware reason for machines to enter repair at Google. Estimates from Google [17] indicate that repair and maintenance costs are about 1% of the capital costs every year. While the percentage seems small, consider the fact that a *single* large datacenter like the one being built in Utah by the National Security Agency (NSA) is expected to cost about $1 billion. At this scale, even small fractions translate to large amounts of money, underscoring the importance of reliability. Moreover, recent work [17] has shown that memory errors can cause security vulnerabilities. Increasing memory density will further increase the rate of errors, exacerbating this problem. The scale of these datacenters makes guaranteeing hardware reliability a challenging problem. For example, even if the Mean Time Before Failure (MTBF) of a part is, say, 1000 days, when 1000 such parts are operated together, a failure is likely every day.

Today's memory architectures depend on symbol-based Reed-Solomon codes [18] to provide very strong reliability guarantees, tolerating multiple random bit-errors in a single chip or even the failure of an entire chip, called *Chipkill*. To achieve this level of fault-tolerance, they spread data over several memory chips, thereby increasing access granularity. While this was acceptable so far, the increasing importance of energy consumption and parallelism makes this untenable, necessitating a fundamentally different approach to reliability.

1.1.5 Lower Relevance of DRAM Chip Area

Given that datacenters consume several billion kilowatt hours of energy every year [2], it has been shown that the 3-year operating energy costs of today's datacenters equal the capital acquisition costs [6]. This means that lowering Total Cost of Ownership (TCO) requires components that are not just cheap, but also efficient. DRAM vendors have long optimized the cost-per-bit metric, even at the cost of increased energy consumption. Memory systems of the future will have to reverse this trend, making it acceptable to incur a slightly higher cost-per-bit when purchasing DRAM as long as it leads to significantly lower energy footprints during operation.

1.1.6 Complexity

Memory systems are becoming increasingly complex. Tight power and performance margins in dense DRAM arrays necessitate as many as 20 different timing parameters to be considered while scheduling every single request. Increasing error rates require maintenance operations such as scrubbing in addition to regular refresh operations. Additionally, there appears to be a clear trend towards heterogeneous systems that incorporate both DRAM and Non-Volatile Memory (NVM) [19, 20]. In such systems, each kind of memory will have its own timing and maintenance requirements. Current memory systems require the memory controller to maintain complete control over all of these details. Going forward, such designs are likely to be highly inflexible and impractical, requiring a rethink of memory access mechanisms.

1.2 Dissertation Overview

It is evident from the preceding discussion that computing systems face disruptive trends on several fronts going forward. Processor architectures have, for the most part, recognized the impacts of these trends and made fundamental changes to their design. They took a "right-hand turn" around 2003 towards flatter power profiles, simpler cores, and more focus on parallelism [21]. Memory architectures, on the other hand, have been largely stagnant, with the same basic design and only minor tweaks from generation to generation. In order to provide the kind of energy, bandwidth, latency, complexity, reliability, and cost guarantees required for the future, the entire memory system needs to undergo a complete overhaul. This dissertation proposes innovations that can lead to order-of-magnitude improvements in memory figures of merit. An understanding of such disruptive innovations is vital before memory architectures can implement their own "right-hand turn".

1.2.1 Thesis Statement

Main memory systems are at an inflection point, with the convergence of several trends such as the increasing importance of energy consumption, reduced access stream locality, increasing failure rates, limited pin counts, increasing heterogeneity and complexity, and the diminished importance of cost-per-bit. In light of these trends, a major overhaul of every memory component is required to build energy-efficient, high-performance, low-complexity, reliable, and cost-effective memory systems for the future. The key to architecting the next generation of memory systems is a combination of the prudent incorporation of novel technologies, and a fundamental rethinking of certain conventional design decisions.

In this dissertation, we study every major element of the memory system – the memory chip, the processor-memory channel, the memory access mechanism, and memory reliability, and identify the key bottlenecks to efficiency. Based on this, we propose novel solutions that help build a memory architecture suited to meet the requirements of future large-scale computing systems.

1.2.2 Memory Chip Design

We start with the most basic component of the memory system, the chips themselves. Conventional DRAM chips are organized into large arrays, and cache lines are striped across multiple such arrays in multiple chips. This boosts storage density, reduces cost-per-bit, and increases the available pin-bandwidth for the transfer. This also leads to the *overfetch* of data, *i.e.,* every DRAM activation reads as much as 8 KB of data into a row buffer. If the access stream has locality, subsequent requests can be serviced quickly by the row buffer. The move to multicore has destroyed locality within memory access streams [22, 23], meaning that much of these data is never used. *The energy cost of this "overfetch" has been recognized as an impediment to realizing energy-efficient exascale DRAM systems [24, 25], and is the key bottleneck in memory chip design.*

In Chapter 3, we present the *Single Subarray Access (SSA)* memory chip architecture that attempts to solve the overfetch problem. It fundamentally re-organizes the layout of DRAM arrays and the mapping of data to these arrays so that an entire cache line is fetched from a single subarray on a single chip, thereby completely eliminating overfetch. It reduces DRAM chip dynamic energy consumption by 6X due to fewer bitline activations. It reduces DRAM chip static energy consumption by 5X due to finer activation footprints and the resultant increase in opportunities for fine-grained power-down modes. Finally, it can improve performance in some cases (over 50% on average) due to its close-page policy and also because it helps alleviate bank contention in some memory-sensitive applications. It achieves all these for about a 5% increase in area, which we believe is a worthwhile tradeoff in light of various trends discussed earlier. This work was published at ISCA'10 [22].

1.2.3 Memory Channel Design

With an efficient memory chip architecture in place, we look at the next bottleneck in the memory system, the processor-memory interconnect. The biggest impediment to building an efficient interconnect architecture is the fundamental nature of high-speed off-chip electrical interconnects. Neither pin count nor per-pin bandwidth is projected to scale, as per ITRS [26], meaning that we cannot arbitrarily increase the number of physical memory channels connected to a processor chip. Also, wide electrical buses with many DIMM drops cannot be driven at high frequencies, owing to both energy and signal integrity issues, fundamentally limiting their bandwidth [27]. Finally, there are limits on the bus length and the number of devices directly connected to the bus (DDR memory supported up to 8 DIMMs on a single channel; this went down to 4 with DDR2 and is expected to drop down to just 2 with DDR3 and beyond [15]), limiting the ability to expand capacity by simply adding more DIMMs. The obvious limitation that these constraints create is the lack of bandwidth that an application has access to. Additionally, they create a latency bottleneck due to increases in queuing delay [28, 29, 30, 31, 32, 33]. *Thus the diverging growth rates of core count and pin count is the key bottleneck in memory channel design.* Since this is a fundamental physical limitation, and not an engineering problem, future memory systems will need to exploit completely different basic technologies to improve efficiency.

In Chapter 4, we explore the use of integrated silicon photonics. This technology has thus far largely been studied in the context of on-chip communication. However, their properties promise to alleviate many of the problems with high-speed electrical pins and off-chip interconnects, and we are one of the few bodies of work to exploit them for memory access. For example, they can provide tremendous bandwidth through high frequency operation and Dense Wavelength Division Multiplexing (DWDM). More importantly, the photonic waveguide can directly couple to optical I/Os on-chip without conversion into the electrical domain, unconstrained by the overheads or limitations of electrical pins, helping break the "pin barrier". Their pitch is also small enough that multiple waveguides can be run in parallel to further increase bandwidth. While silicon photonics inherently have better properties than off-chip electrical interconnects, careful design is still required to fully exploit their potential while being energy-efficient. For instance, they have a nontrivial static energy component, unlike electrical interconnects, though they consume significantly lower dynamic energy. They also face power losses at various points along the light path. Based on detailed analysis, we conclude that photonics are effective primarily to improve socket-edge bandwidth and for use on heavily used links, but that within memory chips, efficient low-swing electrical wires continue to be the appropriate choice. Our final design provides at least 23% energy reduction compared to fully-optical extensions of state-of-the-art photonic DRAMs, with 4X capacity, and the potential for performance improvements due to increased available bank counts. Also, while novel technologies such as photonics make perfect sense from a purely technical perspective, commercial success is heavily dependent on how invasive the required changes are, and the impact on cost. This is especially true in the commodity memory industry. Our proposed design exploits a logic *interface die* on a 3D stack of memory dies that enables low-cost photonic access of unmodified commodity memory dies. Additionally, it enables the design of a single kind of memory die, which can then be used with either electrical or photonic off-chip interconnects, depending on the target market segment. Such a design removes some major hurdles in the road to successful commercial adoption. This work was published at ISCA'11 [34].

1.2.4 Memory Access Protocol Design

SSA chips and photonic channels address hardware problems with the memory system, but that is not all that needs to be overhauled – the memory access mechanism, the way the processor and memory communicate, needs to be changed as well. *The key bottleneck to scalability of the memory interface is the tight control held by the memory controller over every micro-operation of the memory system.* The tight integration of device characteristics, interconnect characteristics, and memory controller severely restricts interoperability. Explicit memory controller support is required for every kind of memory that may be present in the system, *i.e.*, DRAM, PCM, etc. Indeed, explicit support is required even for different kinds of DRAM, or if new features are added to a DRAM device. The memory controller needs to track dozens of timing constraints and maintenance requirements while scheduling every memory access. Such a paradigm is simply not scalable for future large-scale systems that are likely to contain potentially hundreds of banks in scores of DRAM ranks, aided by technologies such as silicon photonics.

In Chapter 5, we fundamentally rethink the access protocol, and present a novel packet-based interface between the processor and memory. We propose making the memory modules more autonomous by moving all low-level functionality to a logic die integrated into a 3D memory stack. The processor side memory controller deals with memory request re-ordering, fairness, quality-of-service, and other such functions that require knowledge of the applications and processor state. Device-level and memory technology specific functionalities such as handling timing constraints and performing maintenance operations are abstracted away. Such a design will help streamline the overall operation of the memory subsystem. It improves interoperability, allowing plug-and-play style operation of different kinds of memory modules without explicitly requiring memory controller redesign and support. It also offers memory module vendors more flexibility and the ability to innovate, allowing them to customize their DIMMs, unconstrained as long as their interface die supports the new features. While the change to the memory interface is significant, we believe that such a massive change is inevitable given the convergence of disruptive technologies such as silicon photonics, NVM, 3D, etc., and the pressing need for scalability. In fact, an analogy can be drawn between our proposal and the SATA interface for storage devices. Early storage controllers issued every microcommand required to read data out of hard disk drives, such as the angle of rotation required, track movement required, etc. This was eventually moved into the drives themselves, and the processor now only sees the higher-level SATA interface. This helped with the relatively painless adoption of flash-based SSDs, where an old hard disk drive could simply be swapped out for a new flash drive, despite a complete internal technology revamp and different operational requirements. This work was published at ISCA'11 [34].

1.2.5 Memory Reliability Mechanism Design

SSA chips, photonic channels, and a packet-based interface improve the efficiency of memory accesses. However, a memory system overhaul is incomplete if it does not guarantee that the data it supplies are free of errors, *i.e.,* if it does not consider support for reliability. Recent studies have shown that DRAMs are often plagued with errors and can lead to significant server downtime in datacenters [17]. A common expectation of business-critical server DRAM systems is that they are able to withstand a single DRAM chip failure, commonly referred to as *Chipkill* [35, 36]. Current chipkill-level mechanisms are based on Reed-Solomon codes and Galois Field arithmetic. *Due to the inherent nature of these codes, and the desire to keep storage overheads low, data and ECC codes are spread across a large number of chips in multiple ranks of the memory system. This is the key bottleneck to providing efficient reliability.* It increases energy consumption due to increased overfetch and increased access granularity, reduces performance due to reduced rank-level parallelism and forced prefetch, and restricts systems to the use of inefficient narrow-I/O x4 DRAMs. With the increasing importance of energy consumption, and decreasing impact of storage costs due to sharp drops in cost-per-bit of commodity DRAMs, we believe that a fundamentally different approach to reliability is called for, one that trades off a small amount of storage overhead for substantial energy and performance benefits.

In Chapter 6, we present LOT-ECC, such a localized and multitiered protection scheme. We separate error detection and error correction functionality, and employ simple checksum and parity codes effectively to provide strong fault-tolerance, while simultaneously simplifying implementation. Data and codes are localized to the same DRAM row to improve access efficiency. We use system firmware to store correction codes in DRAM data memory and modify the memory controller to handle data mapping, error detection, and correction. We thus build an effective fault-tolerance mechanism that provides strong reliability guarantees, reduces power consumption by 55%, reduces latency by 47%, and reduces circuit complexity, all while working with commodity DRAMs and operating systems. We also explore an alternative implementation of this idea, with hardware support from disruptive architectures such as the SSA architecture presented in Chapter 3. Reliability features can be incorporated into the architecture at design time (availability of variable burst lengths and row sizes, for example), simplifying data mapping and increasing flexibility. This work was published at ISCA'12 [37].

Through the substantial redesign of the memory chip, channel, access mechanism, and reliability design, we propose a large-scale memory system for the future that enables low energy consumption, high bandwidth, low latency, improved scalability, improved reliability, and reduced complexity in a cost-effective manner. While this dissertation focuses on DRAM as an evaluation vehicle, the proposed architectures will likely apply just as well to other emerging storage technologies, such as phase change memory (PCM) and spin torque transfer RAM (STT-RAM).

CHAPTER 2

BACKGROUND

2.1 Memory System Basics

We first describe the typical modern main memory architecture [38], focusing on the dominant DRAM architecture today: JEDEC-style DDRx SDRAM. An example is shown in Figure 2.1.

Modern processors [39, 40, 41] often integrate memory controllers on the processor die. Each memory controller is connected to one or two dedicated off-chip memory channels. For JEDEC standard DRAM, the channel typically has a 64-bit data bus, a 17-bit row/column address bus, and an 8-bit command bus [42]. Multiple dual in-line memory modules (*DIMMs*) can be accessed via a single memory channel and memory controller. Each DIMM typically comprises multiple *ranks*, each rank consisting of a set of DRAM chips. Exactly one *rank* is activated on every memory operation and this is the smallest number of chips that need to be activated to complete a read or write operation. Delays on the order of a few cycles are introduced when the memory controller switches between ranks to support electrical bus termination requirements. The proposed DRAM architecture is entirely focused on the DRAM chips, and has neither a positive or negative effect on rank issues. Figure 2.1 shows an example DIMM with 16 total DRAM chips forming two ranks.

Each DRAM chip has an intrinsic *word size* which corresponds to the number of data I/O pins on the chip. An *xN* DRAM chip has a word size of N, where N refers to the number of bits going in/out of the chip on each clock tick. For a 64-bit data bus and x8 chips, a rank would require 8 DRAM chips (Figure 2.1 only shows 8 x4 chips per rank to simplify the figure). If the DIMM supports ECC, the data bus expands to 72-bits and the rank would consist of 9 x8 DRAM chips. When a rank is selected, all DRAM chips in the rank receive address and command signals from the memory

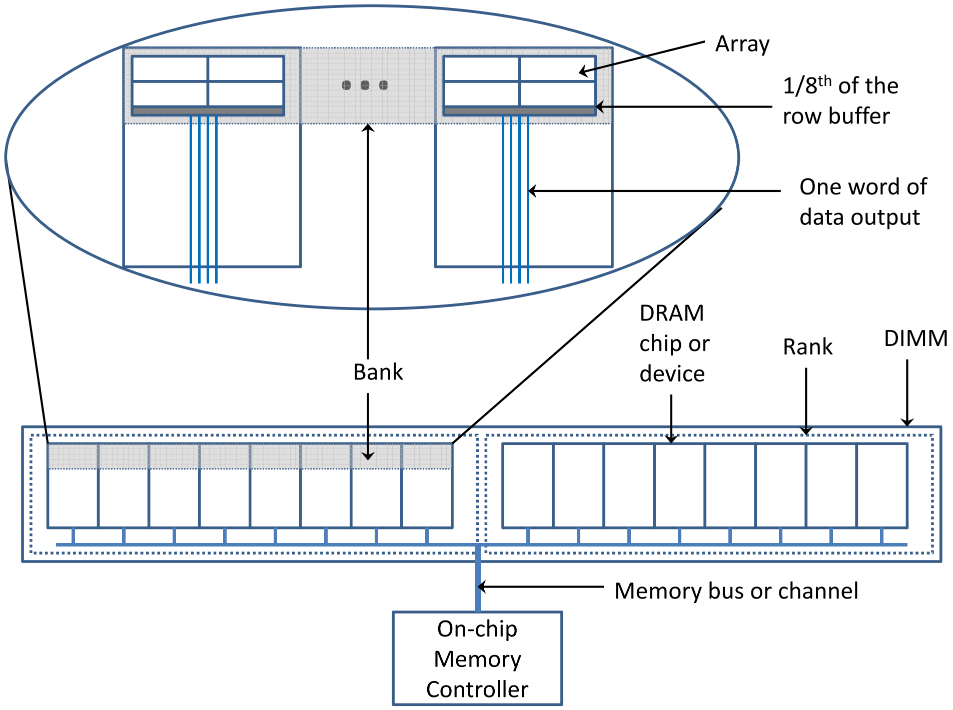


Figure 2.1. An example DDRx SDRAM architecture shown with one DIMM, two ranks, and eight x4 DRAM chips per rank

controller on the corresponding shared buses. Each DRAM chip is connected to a subset of the data bus; of the 64-bit data packet being communicated on the bus on a clock edge, each x8 chip reads/writes an 8-bit subset.

A rank is itself partitioned into multiple *banks*, typically 4-16. Each bank can be concurrently processing a different memory request, thus affording a limited amount of memory parallelism. Each bank is distributed across the DRAM chips in a rank; the portion of a bank in each chip will be referred to as a *sub-bank*. The organization of a sub-bank will be described in the next paragraph. When the memory controller issues a request for a cache line, all the DRAM chips in the rank are activated and each sub-bank contributes a portion of the requested cache line. By striping a cache line across multiple DRAM chips, the available pin and channel bandwidth for the cache line transfer can be enhanced. If the data bus width is 64 bits and a cache line is 64 bytes, the cache line transfer happens in a *burst* of 8 data transfers.

If a chip is an xN part, each sub-bank is itself partitioned into N *arrays* (see Figure 2.1). Each array contributes a single bit to the N-bit transfer on the data

I/O pins for that chip on a clock edge. An array has several rows and columns of single-bit DRAM cells. A cache line request starts with a *RAS* command that carries the subset of address bits that identify the bank and the row within that bank. Each array within that bank now reads out an entire row. The bits read out are saved in latches, referred to as the *row buffer*. The row is now considered *opened*. The *page size* or *row buffer size* is defined as the number of bits read out of all arrays involved in a bank access (usually 4-16 KB). Of these, only a cache line worth of data (identified by the *CAS* command and its associated subset of address bits) is communicated on the memory channel for each CPU request.

Each bank has its own row buffer, so there can potentially be 4-16 open rows at any time. The banks can be accessed in parallel, but the data transfers have to be serialized over the shared data bus. If the requested data are present in an open row (a *row buffer hit*), the memory controller is aware of this, and data can be returned much faster. If the requested data are not present in the bank's row buffer (a *row buffer miss*), the currently open row (if one exists) has to first be closed before opening the new row. To prevent the closing of the row from being on the critical path for the next row buffer miss, the controller may adopt a *close-page policy* that closes the row right after returning the requested cache line. Alternatively, an *open-page policy* keeps a row open until the bank receives a request for a different row.

As an example system, consider a 4 GB system, with two 2 GB ranks, each consisting of eight 256 MB x8, 4-bank devices, serving an L2 with a 64 byte cache line size. On every request from the L2 cache, each device has to provide 8 bytes of data. Each of the 4 banks in a 256 MB device is split into 8 arrays of 8 MB each. If there are 65,536 rows of 1024 columns of bits in each array, a row access brings down 1024 bits per array into the row buffer, giving a total row buffer size of 65,536 bits across 8 chips of 8 arrays each. The page size is therefore 65,536 bits (8 KBytes) and of these, only 64 Bytes are finally returned to the processor, with each of the 8 chips being responsible for 64 bits of the cache line. Such a baseline system usually significantly under-utilizes the bits it reads out (in the above example, only about 0.8% of the row buffer bits are utilized for a single cache line access) and ends up unnecessarily activating various circuits across the rank.

2.2 3D Stacked Memory

In an effort to increase density, reduce power, improve bandwidth, and provide small form factors, the memory industry has been exploring the vertical stacking of several memory dies, connected using Through Silicon Vias (TSVs). This trend is likely to see mainstream adoption, and is already on the technology roadmap of major memory vendors for imminent release [43, 44, 45]. Based on these projections, the traditional memory dies on a DIMM will likely be replaced by a set of 3D stacks of memory chips. Each DRAM die on each stack is typically logically independent, and can be addressed and accessed individually. In such systems, there is likely to be an *interface die* that is stacked with the memory dies and built in a CMOS logic process. This provides several distinct advantages over traditional organizations: (i) it allows us to exploit heterogeneity in the manufacturing process; logic circuits can now be incorporated with memory, without suffering from the inefficiencies of the DRAM process, (ii) it allows innovation in the memory without being invasive to the highly optimized array design of cost-sensitive commodity memory, (iii) it provides proximity to the memory for a small amount of logic circuitry; this can be exploited for light "Processing-In-Memory" style computation, and (iv) it allows the abstraction of several memory-specific properties behind a logical interface. In this dissertation, we will explore two applications of this novel organization – low-cost photonic access to commodity memory (Chapter 4), and a flexible and streamlined memory interface (Chapter 5).

2.3 Photonic Interconnect Basics

Integrated silicon photonics is an emerging technology that promises tremendous off-chip bandwidth within relatively small power envelopes. While several implementations of integrated silicon photonics are currently under study [46, 47, 48, 49, 50], we focus on *microring resonator*-based technology, which uses "rings" to modulate and demodulate light. We refer the reader to prior work [51, 48, 52] for more details, and subsequently we only discuss issues directly impacting the design of photonic memory channels, as discussed in Chapter 4.

2.3.1 Dense Wavelength Division Multiplexing

A key advantage of this technology is the ability to independently modulate, transmit, and detect light with different wavelengths on a single channel through Dense Wave Division Multiplexing (DWDM), providing high bandwidth density. This is achieved by using a multitude of *wavelength selective* ring resonators to form a *ring bank* or a *photonic stop*. Although the number of wavelengths that can be supported is limited by coupling losses between rings (which increase with tighter wavelength spacing), prior studies have shown that DWDM with up to 67 wavelengths is achievable with small rings [50]. Combined with fast modulators capable of operating at up to 5 GHz (10 Gbps with dual-rate), a single waveguide is capable of providing roughly 80 GB/s of bandwidth on a 64-wavelength bus.

2.3.2 Static Power

Photonic modulating rings are sized to resonate at a specific wavelength at fabrication time. A key requirement for the implementation of DWDM is to keep the various ring resonators perfectly tuned to their desired wavelength at all times. However, this wavelength is highly temperature dependent, and tends to drift during operation as temperatures vary. To compensate for this drift, the rings need to be constantly heated, a process called *trimming*. Without trimming, not only will those specific detuned wavelengths be affected, but those rings may drift close to adjacent wavelengths, potentially causing interference and data loss. Note that this trimming has to be on *continuously*, independent of usage of that wavelength or even that die. This introduces a constant, nontrivial static power overhead. Similarly, the receivers have to stay on for the entire duration of operation, adding yet another static power source. It is therefore important to keep utilization high in photonic systems.

2.3.3 Laser Power

The laser power is a function of total photonic loss on the entire *light-path* from source to receiver; some prespecified amount of photonic power, depending on the sensitivity of the receiver, has to finally reach the photodetector after all losses for reliable detection. Typical sources of loss include the on-chip silicon waveguide, off-chip fiber, couplers to go between on- and off-chip waveguides, the modulating rings, etc. With respect to the rings, it is important to note that at any given modulation point, *near rings* (idle rings tuned to the same wavelength or immediately adjacent wavelength at other modulation points on the light path) introduce two orders of magnitude more loss than *far rings* (all other rings). From the perspective of photonic loss, therefore, it is possible to have many hundreds of far rings coupled to a single waveguide, although such an organization may suffer from high trimming power overheads.

2.4 Chipkill-level Memory Reliability

A common expectation of business critical-server DRAM systems is that they are able to withstand the failure of an entire DRAM chip, and continue operation with no data loss [35, 36, 18]. Current commercial chipkill solutions employ Single Symbol Correct Double Symbol Detect (SSC-DSD) codes [53, 18, 54], which operate on a set of bits (a "symbol") rather than individual bits. All errors, of all lengths, within a single symbol can be corrected. There are two popular SSC-DSD codes, the eponymous 3-check-symbol and 4-check-symbol codes [18].

Three check symbols can protect up to $2^b - 1$ data symbols, where b is the width of the symbol. With x4 DRAMs, the symbol-width b is 4, the output of each chip; three ECC chips can therefore protect 15 data chips. Being non-power-of-two all around, this results in granularity mismatches and is inconvenient. The 4-check-symbol code is therefore preferred, which allows protection of more data symbols. 32 data symbols are protected by 4 ECC symbols, creating a 144-bit datapath from 36 total chips. This is typically implemented as two ECC DIMMs with 18 chips each, reading/writing two 64-byte cache lines at a time on a standard DDR3 channel with a burst length of 8.

The x4 chip, 4-check-symbol code-based designs suffer from several drawbacks, as described below, and summarized in Table 2.1. First, ECC codes are computed over large 144-bit data words. This activates a larger number of chips than absolutely required, increasing overfetch within DRAM chips [24, 25, 55, 22], and resulting in substantially increased energy consumption. Area, density, and cost constraints make overfetch inevitable to some extent within a rank of chips, but imposing additional inefficiency in order to provide fault tolerance should be avoided.

Table 2.1. Commercial chipkill implementations; burst length of 8

Design	**Bus-width**	**Granularity (Cache lines)**	**Storage overhead**	**Problems**
SSC-DSD x4, 4-check symbol code, commercially used	128b data + 16b ECC	2	12.5%	Causes overfetch, forces prefetching, reduces rank-level parallelism, uses GF arithmetic, restricted to x4
SSC-DSD x8, 3-check symbol code, Option 1	2040b data + 24b ECC	31	1.17%	Significant overfetch plus wasted bits due to nonpower of 2 data length, forced prefetching, nonstandard channel width, GF arithmetic
SSC-DSD x8, 3-check symbol code, Option 2	64b data + 24b ECC	1	37.5%	Significant storage overhead, nonstandard 88-bit channel, GF arithmetic
SSC-DSD x8, 3-check symbol code, Option 3	128b data + 24b ECC	2	18.75%	Significant overfetch and prefetching, reduced parallelism, nonstandard 152-bit channel, GF arithmetic

Second, the wide-word requirement results in increased access granularity as burst lengths increase. A 144-bit bus with the standard DDR3 burst length of 8 already reads/writes two 64-byte cache lines per access. This forced prefetch potentially wastes bandwidth and energy unless access locality is consistently high. Third, since a large number of chips is made busy on every access, there are fewer opportunities for rank-level/bank-level parallelism within a given amount of memory, potentially hurting performance. Bank contention will likely emerge as a major bottleneck if novel interconnect technologies such as silicon photonics [23, 34] substantially increase the available off-chip memory bandwidth, making parallelism more important. All of these problems are exacerbated by the fact that the structure of the ECC codes forces the use of narrow-I/O x4 DRAM chips [18]. This increases the number of DRAM chips needed to achieve a given data bus width, reducing space on the DIMM for more DRAM chips, decreasing the number of independent ranks available [38]. Additionally, for a given capacity, DIMMs with narrow chips consume more energy than those with wider I/O chips [56]. Attempts to reduce the access granularity or move to wide-I/O x8 or x16 DRAM chips results in a significant increase in storage overhead for the ECC codes [18]. Finally, symbol-based ECC computation and verification entails significant circuit complexity due to the involvement of Galois field arithmetic, particularly with wide symbols such as 8 or 16 bits [18, 57].

With x8 DRAMs, on the other hand, b is 8, allowing just three check symbols to protect as many as 255 data symbols. We consider three protection strategies; all considered options are summarized in Table 2.1. While it would be most efficient from a storage overhead perspective to use a configuration of 3 ECC chips + 255 data chips, the access granularity would be unacceptably large. Reducing access granularity to a single cache line would require 3 ECC chips + 8 data chips, but storage overhead rises to 37.5%. Reducing storage overhead to 18.75% through a 3 ECC + 16 data configuration ends up reading/writing two cache lines at a time, in addition to requiring a nonstandard 152-bit channel. The server industry has therefore stayed away from x8 DRAMs for chipkill-correct systems so far. Similar tradeoffs can be made with x16 or wider DRAMs, but at the cost of much sharper increases in either access granularity or storage overhead.

CHAPTER 3

ENERGY-EFFICIENT MEMORY CHIP DESIGN

3.1 Why We Need A Chip Overhaul

The memory wall is not new: long DRAM memory latencies have always been a problem. Given that little can be done about the latency problem, DRAM vendors have chosen to optimize their designs for improved bandwidth, increased density, and minimum cost-per-bit. With these objectives in mind, a few DRAM architectures, standards, and interfaces were instituted in the 1990s and have persisted since then. However, the objectives in datacenter servers and HPC platforms of the future will be very different than those that are reasonable for personal computers, such as desktop machines. As a result, traditional DRAM architectures are highly inefficient from a future system perspective, and are in need of a major revamp. For example, with energy consumption becoming such a big problem, the Total Cost of Ownership (TCO) is influenced far less by area and cost than before. It may therefore be acceptable to pay a small cost penalty, if operating power can be lowered. The design of DRAM devices specifically addressing these trends has, to the best of our knowledge, not been previously studied and is now more compelling than ever.

We attempt to fundamentally rethink DRAM microarchitecture and organization to achieve high-performance operation with extremely low energy footprints, all within acceptable area bounds. In this chapter, we propose two independent designs, both attempting to activate the minimum circuitry required to read a single cache line, rather than overfetch. We introduce and evaluate *Posted-RAS* in combination with a *Selective Bitline Activation (SBA)* scheme. This entails a relatively simple change to DRAM microarchitecture, with only a minor change to the DRAM interface, to provide significant dynamic energy savings. We then propose and evaluate a reorganization of DRAM chips and their interface, so that cache lines can be read via a *Single Subarray Access (SSA)* in a single DRAM chip. This approach trades off higher data transfer times for greater (dynamic and background) energy savings. While this study focuses on DRAM as an evaluation vehicle, the proposed architectures will likely apply just as well to other emerging storage technologies, such as phase change memory (PCM) and spin torque transfer RAM (STT-RAM).

3.2 Motivational Data

We start with a workload characterization on our simulation infrastructure (methodology details in Section 3.4.1). Figure 3.1 shows the trend of steeply dropping row-buffer hit rates as the number of threads simultaneously accessing memory goes up. We see average rates drop from over 60% for a 1 core system to 35% for a 16 core system. We also see that whenever a row is fetched into the row-buffer, the number of times it is used before being closed due to a conflict is often just one or two (Figure 3.2). This indicates that even on benchmarks with high locality and good average row buffer hit rates (for example, *cg*), a large number of pages still do not have much reuse in the row-buffer. These trends have also been observed in prior work on Micro-Pages [58]. This means that the energy costs of activating an entire 8 KB row is amortized over very few accesses, wasting significant energy.

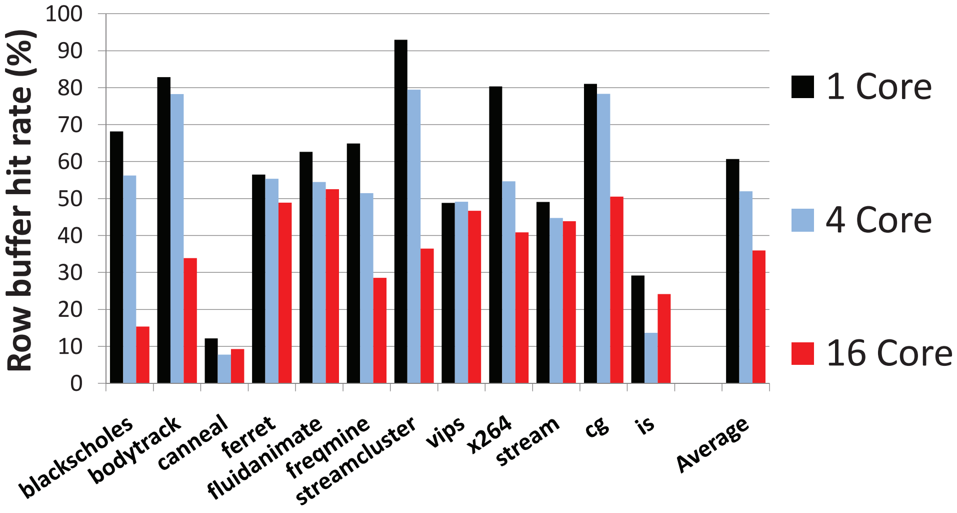


Figure 3.1. Row buffer hit rate trend

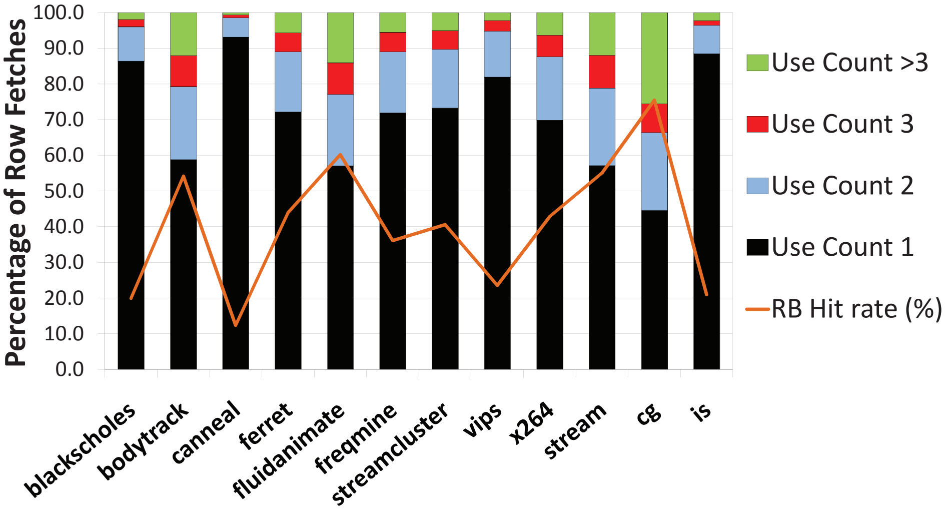


Figure 3.2. Row use count for 8 cores

3.3 Proposed Architecture

We start with the premise that the traditional row-buffer locality assumption is no longer valid, and try to find an energy-optimal DRAM design with minimal impacts on area and latency. Our first novel design (Selective Bitline Activation, Section 3.3.1) requires minor changes to DRAM chip microarchitecture, but is compatible with existing DRAM standards and interfaces. The second novel design (Single Subarray Access, Section 3.3.2) requires nontrivial changes to DRAM chip microarchitecture and its interface to the memory controller.

3.3.1 Selective Bitline Activation (SBA)

In an effort to mitigate the overfetch problem with minimal disruption to existing designs and standards, we propose the following two simple modifications: (i) we activate a much smaller segment of the wordline and (ii) we activate only those bitlines corresponding to the requested cache line. Note that we will still need a wire spanning the array to identify the exact segment of wordline that needs to be activated but this is very lightly loaded and therefore has low delay and energy. Thus, we are not changing the way data get laid out across DRAM chip arrays, but every access only brings down the relevant cache line into the row buffer. As a result, the notion of an open-page policy is now meaningless. After every access, the cache line is immediately written back. Most of the performance difference from this innovation is because of the shift to a close-page policy: for workloads with little locality, this can actually result in performance improvements as the page precharge after write-back is taken off the critical path of the subsequent row buffer miss. Next, we discuss the microarchitectural modifications in more detail.

Memory systems have traditionally multiplexed RAS and CAS commands on the same I/O lines due to pin count limitations. This situation is unlikely to change due to technological limitations [14] and is a hard constraint for DRAM optimization. In a traditional design, once the RAS arrives, enough information is available to activate the appropriate wordline within the array. The cells in that row place their data on the corresponding bitlines. Once the row's data is latched into the row buffer, the CAS signal is used to return some fraction of the many bits read from that array. In our proposed design, instead of letting the RAS immediately activate the entire row and all the bitlines, we wait until the CAS has arrived to begin the array access. The CAS bits identify the subset of the row that needs to be activated and the wordline is only driven in that section. Correspondingly, only those bitlines place data in the row buffer, saving the activation energy of the remaining bits. Therefore, we need the RAS and the CAS before starting the array access. Since the RAS arrives early, it must be stored in a register until the CAS arrives. We refer to this process as *Posted-RAS*[1]. Because we are now waiting for the CAS to begin the array access, some additional cycles (on the order of 10 CPU cycles) are added to the DRAM latency. We expect this impact (quantified in Section 3.4) to be relatively minor because of the hundreds of cycles already incurred on every DRAM access. Note again that this change is compatible with existing JEDEC standards: the memory controller issues the same set of commands, we simply save the RAS in a register until the CAS arrives before beginning the array access.

The selective bitline activation is made possible by only activating a small segment

[1] Many memory controllers introduce a gap between the issue of the RAS and CAS so that the CAS arrives just as the row buffer is being populated and the device's T_{rcd} constraint is satisfied [38]. Some memory systems send the CAS immediately after the RAS. The CAS is then saved in a register at the DRAM chip until the row buffer is ready. This is referred to as Posted-CAS [59]. We refer to our scheme as Posted-RAS because the RAS is saved in a register until the arrival of the CAS.

of the wordline. We employ hierarchical wordlines to facilitate this, at some area cost. Each wordline consists of a Main Wordline (MWL), typically run in first-level metal, controlling Sub-Wordlines (SWL), typically run in poly, which actually connect to the memory cells (see Figure 3.3). The MWL is loaded only by a few "AND" gates that enable the sub-wordlines, significantly reducing its capacitance, and therefore its delay. "Region Select (RX)" signals control activation of specific SWLs.

Hierarchical wordlines have been previously proposed for DRAMs [1] to reduce delay (rather than energy). Until now, other techniques (metal shunted wordlines [60], for instance) partially achieved what has been perceived as the advantage of hierarchical wordlines: significant reductions in wordline delay. In a shunted wordline, a metal wordline is stitched to the low-pitch poly wordline at regular intervals by metal-poly contacts. This reduces the wordline delay by limiting the high resistance poly to a small distance while saving area by having only a few metal-poly contacts. The increased area costs of hierarchical wordlines have therefore not been justifiable thus far. Now, with the increasing importance of energy considerations, we believe that using hierarchical wordlines is not only acceptable, but actually necessary. Note that wordlines do not contribute as much to overall DRAM energy, so this feature is important not for its wordline energy savings, but because it *enables* selective bitline activation. In our proposed design, a subset of the CAS address is used to trigger the RX signal, reducing the activation area and wordline/bitline energy. Note that since the MWL is not directly connected to the memory cells, the activation of the MWL

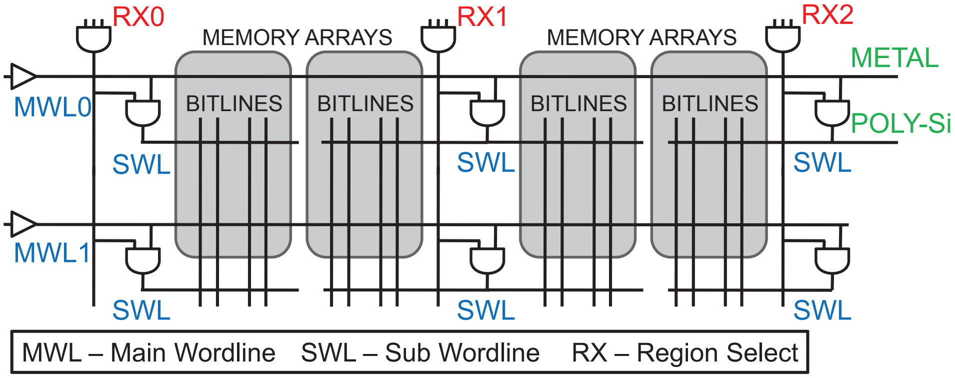


Figure 3.3. Hierarchical wordline with region select [1]

across the array does not result in destruction of data, since only the small subset of cells connected to the active SWL read their data out.

We incorporated an analytical model for hierarchical wordlines into CACTI 6.5 [61, 62] (more details in Section 3.4.1) to quantify the area overhead. For the specific DRAM part described in Section 3.4.1, we observed that an area overhead of 100% was incurred when enough SWLs were introduced to activate exactly one cache line in a bank. This is because of the high area overhead introduced by the AND gate and RX signals for a few memory cells. While this results in activating a minimum number of bitlines, the cost may be prohibitive. However, we can trade off energy for lower cost by not being as selective. If we were to instead read out 16 cache lines, the SWLs become 16 times longer. This still leads to high energy savings over the baseline, and a more acceptable area overhead of 12%. Most of our results in Section 3.4 pertain to this model. Even though we are reading out 16 cache lines, we continue to use the close-page policy.

In summary, the SBA mechanism (i) reduces bitline and wordline dynamic energy by reading out a limited number of cache lines from the arrays (to significantly reduce overfetch), (ii) impacts performance (negatively or positively) by using a close-page policy, (iii) negatively impacts performance by waiting for CAS before starting array access, (iv) increases area and cost by requiring hierarchical wordlines, and finally (v) does not impact the DRAM interface. As we will discuss subsequently, this mechanism does not impact any chipkill solutions for the DRAM system because the data organization across the chips has not been changed.

3.3.2 Single Subarray Access (SSA)

While the SBA design can eliminate overfetch, it is still an attempt to shoehorn in energy optimizations in a manner that conforms to modern-day DRAM interfaces and data layouts. Given that we have reached an inflection point, a major rethink of DRAM design is called for. An energy-efficient architecture will also be relevant for other emerging storage technologies. This subsection defines an energy-optimized architecture (SSA) that is not encumbered by existing standards.

Many features in current DRAMs have contributed to better locality handling and low cost-per-bit, but also to high energy overhead. Arrays are designed to be large structures so that the peripheral circuitry is better amortized. While DRAMs can allow low-power sleep modes for arrays, the large size of each array implies that the power-down granularity is rather coarse, offering fewer power-saving opportunities. Since each DRAM chip has limited pin bandwidth, a cache line is striped across all the DRAM chips on a DIMM to reduce the data transfer time (and also to improve reliability). As a result, a single access activates multiple chips, and multiple large arrays within each chip.

3.3.2.1 Overview

To overcome the above drawbacks and minimize energy, we move to an extreme model where an entire cache line is read out of a single small array in a single DRAM chip. This small array is henceforth referred to as a *"subarray"*. Figure 3.4 shows the entire memory channel and how various subcomponents are organized. The subarray is as wide as a cache line. Similar to SBA, we see a dramatic reduction in dynamic energy by only activating enough bitlines to read out a single cache line. Further, remaining inactive subarrays can be placed in low-power sleep modes, saving background energy. The area overhead of SSA is lower than that of SBA since we divide the DRAM array at a much coarser granularity.

If the DRAM chip is an x8 part, we either need to provide 8 wires from each

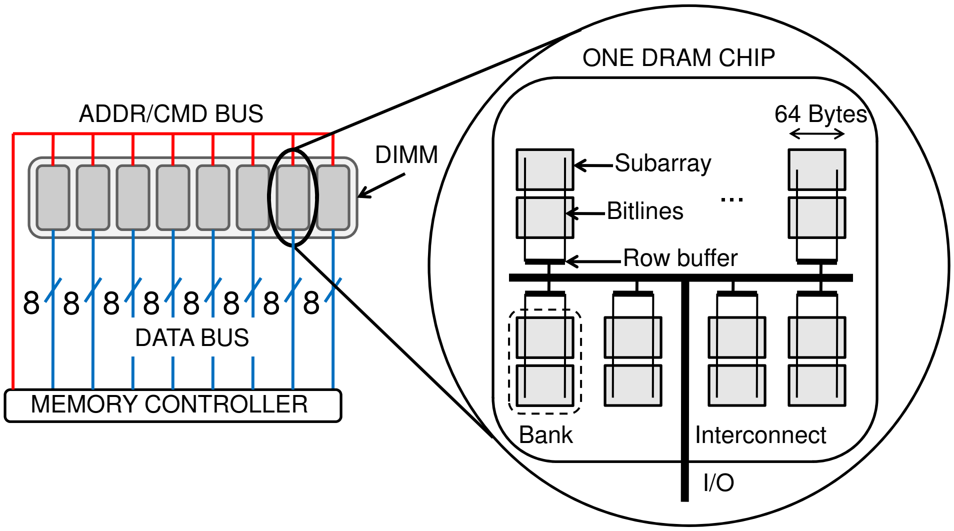


Figure 3.4. SSA DRAM architecture

subarray to the I/O pins or provide a single wire and serialize the transfer. We adopt the former option and as shown in Figure 3.4, the subarrays place their data on a shared 8-bit bus. In addition, since the entire cache line is being returned via the limited pins on a single chip, it takes many more cycles to effect the data transfer to the CPU. Thus, the new design clearly incurs a higher DRAM latency because of slow data transfer rates. It also only supports a close-page policy, which can impact performance either positively or negatively. On the other hand, the design has much higher concurrency, as each DRAM chip can be simultaneously servicing a different cache line. Since each chip can implement several independent subarrays, there can also be much higher intrachip or bank-level concurrency. We next examine our new design in greater detail.

3.3.2.2 Memory controller interface

Just as in the baseline, a single address/command bus is used to communicate with all DRAM chips on the DIMM. The address is provided in two transfers because of pin limitations on each DRAM chip. This is similar to RAS and CAS in a conventional DRAM, except that they need not be called as such (there is not a column-select in our design). The address bits from both transfers identify a unique subarray and row (cache line) within that subarray. Part of the address now identifies the DRAM chip that has the cache line (not required in conventional DRAM because all chips are activated). The entire address is required before the subarray can be identified or accessed. Similar to the SBA technique, a few more cycles are added to the DRAM access latency. An additional requirement is that every device has to be capable of latching commands as they are received to enable the command bus to then move on to operating a different device. This can easily be achieved by having a set of registers (each capable of signaling one device) connected to a demultiplexer which reads commands off the command bus and redirects them appropriately. The data bus is physically no different than the conventional design: for an xN DRAM chip, N data bits are communicated between the DRAM chip and the memory controller every bus cycle. Logically, the N bits from every DRAM chip on a DIMM rank were part of the same cache line in the conventional design; now they are completely independent and deal with different cache lines. Therefore, it is almost as if there are eight independent narrow channels to this DIMM, with the caveat that they all share a single address/command bus.

3.3.2.3 Subarray organization

The height of each subarray (*i.e.,* the number of cache lines in a given subarray) directly determines the delay/energy per access within the subarray. Many small subarrays also increase the potential for parallelism and low-power modes. However, a large number of subarrays implies a more complex on-die network and more energy and delay within this network. It also entails greater overhead from peripheral circuitry (decoders, drivers, senseamps, etc.) per subarray which directly impacts area and cost-per-bit. These are basic trade-offs considered during DRAM design and even incorporated into analytical cache models such as CACTI 6.5 [61, 62]. Figure 3.4 shows how a number of subarrays in a column share a row buffer that feeds the shared bus. The subarrays sharing a row buffer are referred to as a bank, and similar to the conventional model, a single bank can only be dealing with one access at a time. Our SSA implementation models hierarchical bitlines in which data read from a subarray are sent to the row buffer through second level bitlines. To distribute load and maximize concurrency, data are interleaved such that consecutive cache lines are first placed in different DRAM chips and then in different banks of the same chip. To limit the impact on area and interconnect overheads, if we assume the same number of banks per DRAM chip as the baseline, we still end up with a much higher number of total banks on the DIMM. This is because in the baseline organization, the physical banks on all the chips are simply parts of larger logical banks. In the SSA design, each physical bank is independent and a much higher degree of concurrency is offered. Our analysis with a heavily extended version of CACTI 6.5 showed that the area overhead of SSA is only 4%.

Since subarray widths are only 64 bytes, sequential refresh at this granularity will be more time-consuming. However, it is fairly easy to refresh multiple banks simultaneously, *i.e.*, they simply act as one large bank for refresh purposes. In addition, there exist simple techniques to perform refresh that keep the DRAM cell's access transistor on long enough to recharge the storage capacitor immediately after a destructive read, without involving the row-buffer [38].

3.3.2.4 Power-down modes

In the SSA architecture, a cache line request is serviced by a single bank in a single DRAM chip, and only a single subarray within that bank is activated. Since the activation "footprint" of the access is much smaller in the SSA design than in the baseline, there is the opportunity to power-down a large portion of the remaining area that may enjoy longer spells of inactivity. Datasheets from Micron [42] indicate that modern chips already support multiple power-down modes that disable various circuitry like the input and output buffers or even freeze the DLL. These modes do not destroy the data on the chip and the chip can be reactivated with a latency penalty proportional to the amount of circuitry that has been turned off and the depth of the power-down state. We adopt a simple strategy for power-down: if a subarray has been $Idle$ for I cycles, it goes into a power-down mode that consumes P times less background power than the active mode. When a request is later sent to this subarray, a W cycle latency penalty is incurred for wake-up. The results section quantifies the performance and power impact for various values of I, P, and W.

3.3.2.5 Impact summary

In summary, the proposed organization targets dynamic energy reduction by only activating a single chip and a single subarray (with short wordlines and exactly the required number of bitlines) when accessing a cache line. Area overhead is increased, compared to conventional DRAM, because each small subarray incurs the overhead of peripheral circuitry and because a slightly more complex on-die interconnect is required. Background energy can be reduced because a large fraction of the on-chip real estate is inactive at any point and can be placed in low-power modes. The interface between the memory controller and DRAM chips has been changed by effectively splitting the channel into multiple smaller width channels. Performance is impacted favorably by having many more banks per DIMM and higher concurrency. Similar to the baseline, if we assume that each chip has eight banks, the entire DIMM now has 64 banks. Performance may be impacted positively or negatively by adopting a close-page policy. Performance is negatively impacted because the cache line is returned to the memory controller via several serialized data transfers (an x8 part will take 64 transfers to return a 64 byte cache line). A negative impact is also incurred because the subarray access can begin only after the entire address is received.

We believe that SSA is superior to SBA, although it requires a larger redesign investment from the DRAM community. Firstly, in order to limit the area overhead of hierarchical wordlines, SBA is forced to fetch multiple cache lines, thus not completely eliminating overfetch. SSA therefore yields higher dynamic energy savings. By moving from large arrays in SBA to small subarrays in SSA, SSA also finds many more opportunities to place subarrays in low-power states and save leakage energy. In terms of performance, SSA is hurt by the long data transfer time, and will outdo SBA in workloads that have a high potential for bank-level concurrency.

3.4 Results

3.4.1 Methodology

We model a baseline, 8-core, out-of-order processor with private L1 caches and a shared L2 cache. We assume a main memory capacity of 4 GB organized as shown in Table 3.1. Our simulation infrastructure uses Virtutech's SIMICS [63] full-system simulator, with out-of-order timing supported by Simics' *'ooo-micro-arch'* module. The *'trans-staller'* module was heavily modified to accurately capture DRAM device

Table 3.1. General simulation parameters

Processor	8-Core OOO, 2GHz
L1 cache	Fully Private, 3 cycle 2-way, 32 KB each I and D
L2 cache	Fully shared, 10 cycle 8-way, 2 MB, 64B Cache lines
Row-buffer size	8 KB
DRAM Frequency	400 MHz
DRAM Part	256MB, x8
Chips per DIMM	16
Channels	1
Ranks	2
Banks	4
T-rcd, T-cas, T-rp	5 DRAM cyc

timing information including multiple channels, ranks, banks, and open rows in each bank. Both open- and close-row page management policies with first-come-first-serve (FCFS) and first-ready-first-come-first-serve (FR-FCFS) scheduling with appropriate queuing delays are accurately modeled. We also model overlapped processing of commands by the memory controller to hide precharge and activation delays when possible. We also include accurate bus models for data transfer between the memory controller and the DIMMs. Address mapping policies were adopted from the DRAMSim [64] framework and from [38]. DRAM timing information was obtained from Micron datasheets [42].

Area, latency, and energy numbers for DRAM banks were obtained from CACTI 6.5 [65], heavily modified to include accurate models for commodity DRAM, both for the baseline design and with hierarchical wordlines. By default, CACTI divides a large DRAM array into a number of mats with an H-tree to connect the mats. Such an organization incurs low latency but requires large area. However, traditional DRAM banks are heavily optimized for area to reduce cost and employ very large arrays with minimal peripheral circuitry overhead. Read or write operations are typically done using long multilevel hierarchical bitlines spanning the array instead of using an H-tree interconnect. We modified CACTI to reflect such a commodity DRAM implementation. Note that with a hierarchical bitline implementation, there is a potential opportunity to trade-off bitline energy for area by only using hierarchical wordlines at the higher-level bitline and leaving the first-level bitlines untouched. In this dissertation, we do not explore this trade-off. Instead, we focus on the maximum energy reduction possible. The DRAM energy parameters used in our evaluation are listed in Table 3.2. We evaluate our proposals on subsets of the multithreaded PARSEC [66], NAS [67] and STREAM [68] benchmark suites. We run every application for 2 million DRAM accesses (corresponding to many hundreds of millions of instructions) and report total energy consumption and IPC.

3.4.2 Results

We first discuss the energy advantage of the SBA and SSA schemes. We then evaluate the performance characteristics and area overheads of the proposed schemes

Table 3.2. Energy parameters

Component	**Dynamic Energy(nJ)**
Decoder + Wordline + Senseamps - Baseline	1.429
Decoder + Wordline + Senseamps - SBA	0.024
Decoder + Wordline + Senseamps - SSA	0.013
Bitlines - Baseline	19.282
Bitlines - SBA/SSA	0.151
Termination Resistors Baseline/SBA/SSA	7.323
Output Drivers	2.185
Global Interconnect Baseline/SBA/SSA	1.143
Low-power mode	**Background Power (mW)**
Active	104.5
Power-Down (3 mem. cyc)	19.0
Self Refresh (200 mem. cyc)	10.8

relative to the baseline organization.

3.4.2.1 Energy characteristics

Figure 3.5 shows the energy consumption of the close-page baseline, SBA, and SSA, normalized to the open-page baseline. The close-page baseline is clearly worse in terms of energy consumption than the open-page baseline simply due to the fact that even accesses that were potentially row-buffer hits (thus not incurring the energy of activating the entire row again) now need to go through the entire activate-read-precharge cycle. We see an average increase in energy consumption by 73% on average, with individual benchmark behavior varying based on their respective row-buffer hit rates. We see from Figure 3.6 (an average across all benchmarks) that in the baseline organizations (both open and close row), the total energy consumption in the device is dominated by energy in the bitlines. This is because every access to a new row results in a large number of bitlines getting activated *twice*, once to read data out of the cells into the row-buffer and once to precharge the array.

Moving to the SBA or SSA schemes eliminates a huge portion of this energy

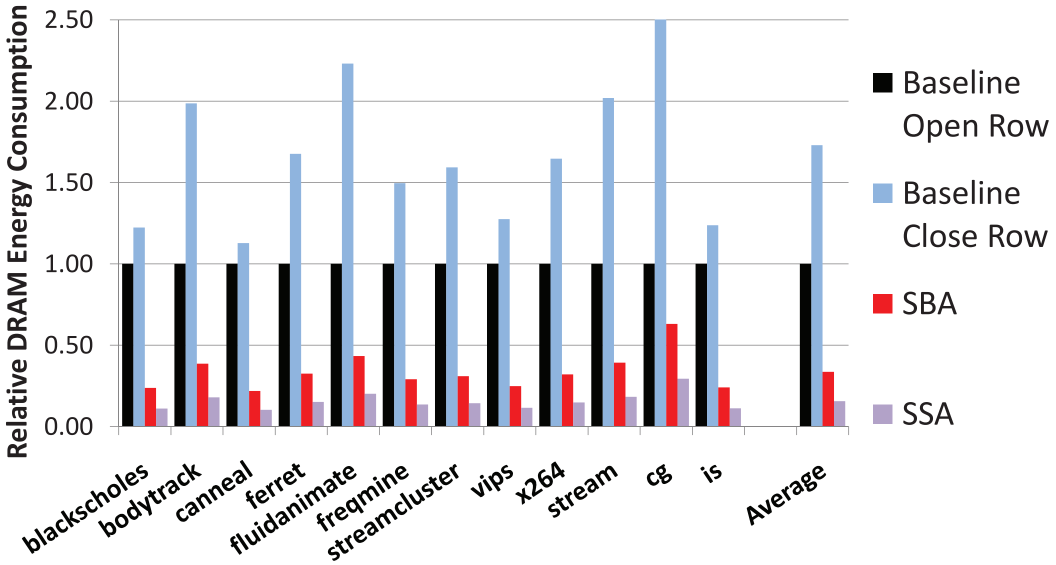


Figure 3.5. DRAM dynamic energy consumption

component. By waiting for the CAS signal and only activating/precharging the exact cache line that we need, bitline energy goes down by a factor of 128. This results in a dramatic energy reduction on every access. However, as discussed previously, prohibitive area overheads necessitate coarser grained selection in SBA, leading to slightly larger energy consumption compared to SSA. Compared to a baseline open-page system, we see average dynamic memory energy savings of *3X* in SBA and *over 6.4X* in SSA. Note that the proposed optimizations result in energy reduction only in the bitlines. The energy overhead due to other components such as decoder, predecoder, interbank bus, bus termination, etc. remains the same. Hence, their contribution to the total energy increases as bitline energy goes down. Localizing and managing DRAM accesses at a granularity as fine as a subarray allows more opportunity to put larger parts of the DRAM into low-power states. Current DRAM devices support multiple levels of power-down, with different levels of circuitry being turned off, and correspondingly larger wake-up penalties. We evaluate two simple low-power modes with P (Power savings factor) and W (Wakeup) values calculated based on numbers shown in Table 3.2, obtained from the Micron datasheet and power system calculator [42, 10]. In the deepest sleep mode, *Self Refresh*, P is 10 and W is 200 memory cycles. A less deep sleep mode is *power-down*, where P is 5.5, but W is

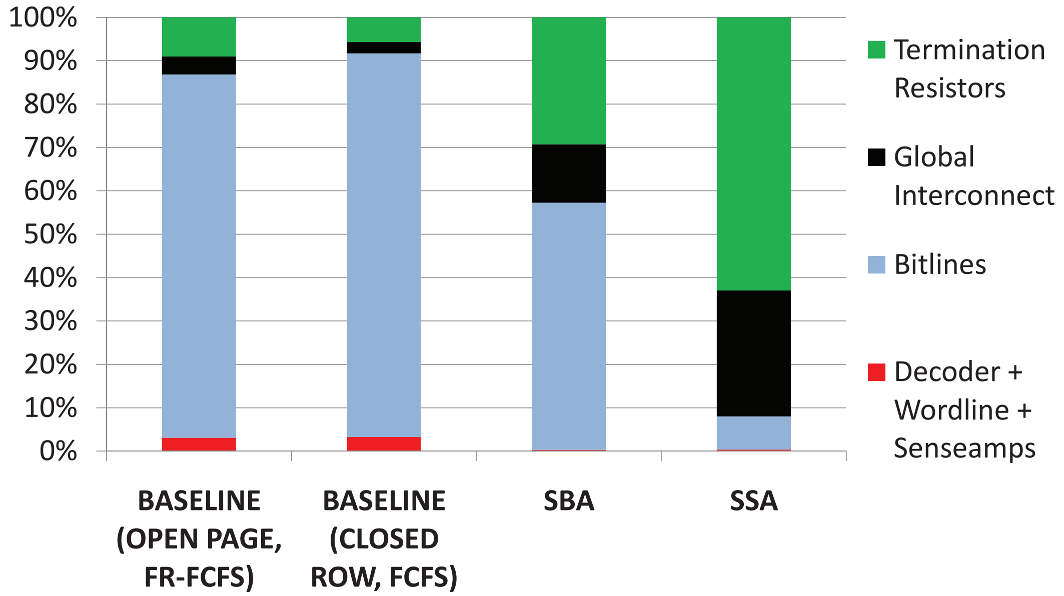


Figure 3.6. Contributors to DRAM dynamic energy

just 3 memory cycles. We vary I (Idle cycle threshold) as multiples of the wake-up time W. Figures 3.7 and 3.8 show the impact of these low-power states on performance and energy consumption in the SSA organization. We see that the more expensive *Self Refresh* low-power mode actually buys us much lower energy savings compared to the more efficient *power-down* mode. As we become less aggressive in transitioning to low-power states (increase I), the average memory latency penalty goes down, from just over 5% to just over 2% for the “power-down” mode. The percentage of time we can put subarrays in low-power mode correspondingly changes from almost 99% to about 86% with energy savings between 81% and 70%. The performance impacts are much larger for the expensive *Self Refresh* mode, going from over 400% at a very aggressive I to under 20% in the least aggressive case. Correspondingly, banks can be put in this state between 95% and 20% of the time, with energy savings ranging from 85% to 20%.

Naturally, these power-down modes can be applied to the baseline architecture as well. However, the granularity at which this can be done is much coarser, a DIMM bank at best. This means that there are fewer opportunities to move into low-power states. As a comparison, we study the application of the low-overhead “power-down”

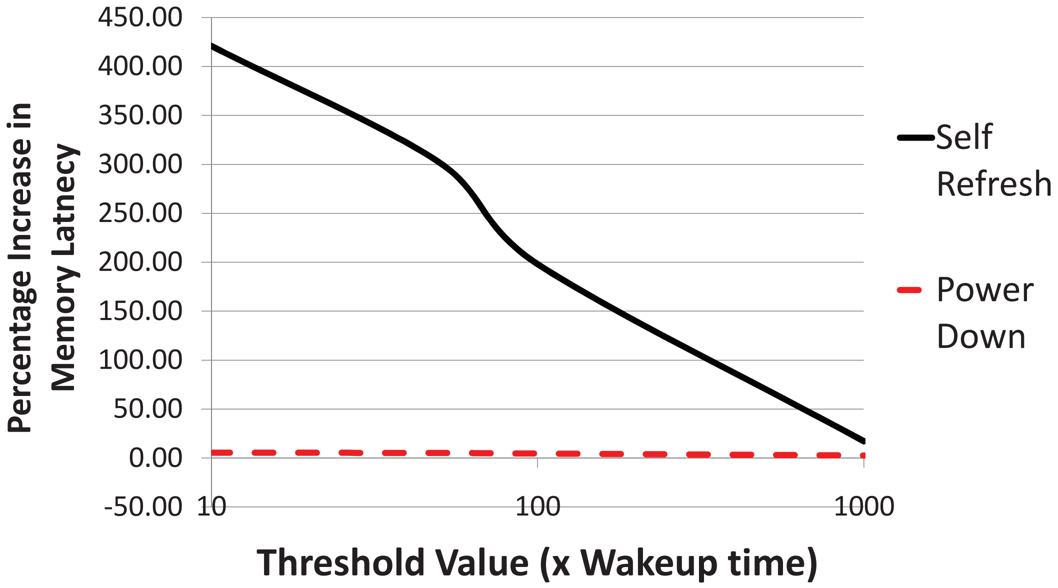


Figure 3.7. Memory latency impact of using low-power states

state to the baseline. We find that on average, even with an aggressive sleep threshold, banks can only be put in this mode about 80% of the time, while incurring a penalty of 16% in terms of added memory latency. Being less aggressive dramatically impacts the ability to power down the baseline, with banks going into sleep mode only 17% of the time with a minimal 3% latency penalty. As another comparison point, we consider the percentage of time subarrays or banks can be put in the deepest sleep Self Refresh mode in SSA vs. the baseline, for a constant 10% latency overhead. We find that subarrays in SSA can go into deep sleep nearly 18% of the time whereas banks in the baseline can only go into deep sleep about 5% of the time.

3.4.2.2 Performance characteristics

Employing either the SBA or SSA schemes impacts memory access latency (positively or negatively) as shown in Figure 3.9. Figure 3.10 then breaks this latency down into the average contributions of the various components. One of the primary factors affecting this latency is the page management policy. Moving to a close-page policy from an open-page baseline actually results in a drop in average memory latency by about 17% for a majority (10 of 12) of our benchmarks. This has favorable implications for SBA and SSA which must use a close-page policy. The remaining

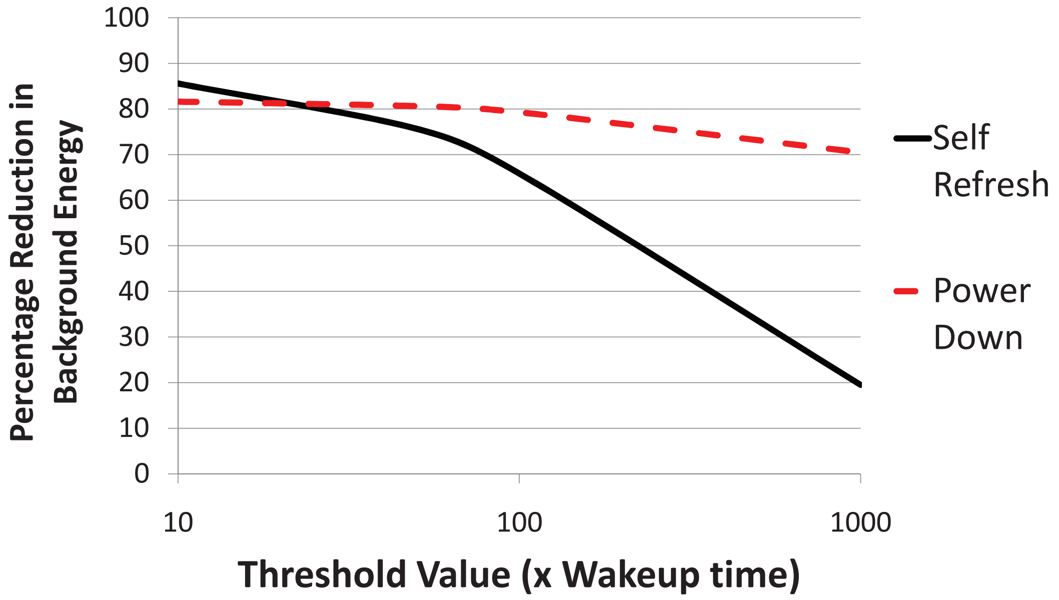


Figure 3.8. Energy reduction using low-power states

benchmarks see an increase in memory latency by about 28% on average when moving to close-page. Employing the "Posted-RAS" scheme in the SBA model causes an additional small latency of just over 10% on average (neglecting two outliers).

As seen in Figure 3.10, for these three models, the queuing delay is the dominant contributor to total memory access latency. Prior work [69] has also shown this to be true in many DRAM systems. We therefore see that the additional latency introduced by the "Posted-RAS" does not significantly change average memory access latency.

The SSA scheme, however, has an entirely different bottleneck. Every cache line return is now serialized over just 8 links to the memory controller. This data transfer delay now becomes the dominant factor in the total access time. However, this is offset to some extent by a large increase in parallelism in the system. Each of the 8 devices can now be servicing independent sets of requests, significantly reducing the queuing delay. As a result, we do not see a greatly increased memory latency. On half of our benchmarks, we see latency increases of just under 40%. The other benchmarks are actually able to exploit the parallelism much better, and this more than compensates for the serialization latency, with average access time going down by about 30%. These are also the applications with the highest memory latencies.

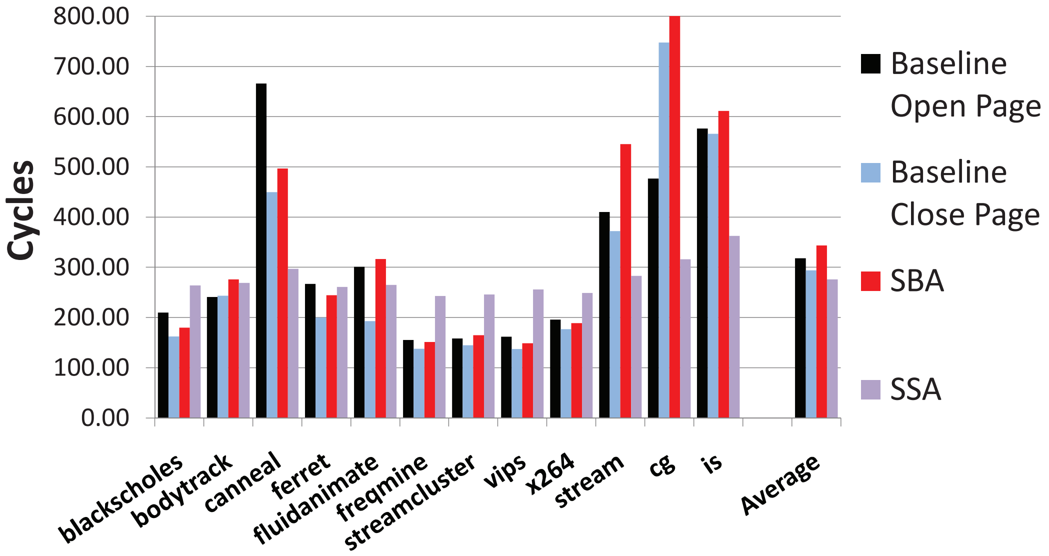


Figure 3.9. Average main memory latency

As a result, overall, SSA in fact outperforms all other models.

Figure 3.11 shows the relative IPCs of the various schemes under consideration. Like we saw for the memory latency numbers, a majority of our benchmarks perform better with a close-row policy than with an open-row policy. We see performance improvements of just under 10% on average (neglecting two outliers) for 9 of our 12 benchmarks. The other three suffered degradations of about 26% on average. These were the benchmarks with relatively higher last-level cache miss rates (on the order of 10 every 1000 instructions). Employing the “Posted-RAS” results in a marginal IPC degradation over the close-row baseline, about 4% on average, neglecting two outlier benchmarks.

The SSA scheme sees a performance degradation of 13% on average compared to the open-page baseline on the 6 benchmarks that saw a memory latency increase. The other 6 benchmarks with a decreased memory access latency see performance gains of 54% on average. These high numbers are observed because these applications are clearly limited by bank contention and SSA addresses this bottleneck. To summarize, in addition to significantly lowered DRAM access energies, SSA occasionally can boost performance, while yielding minor performance slowdowns for others. We expect SSA to yield even higher improvements in the future as ever more cores exert higher

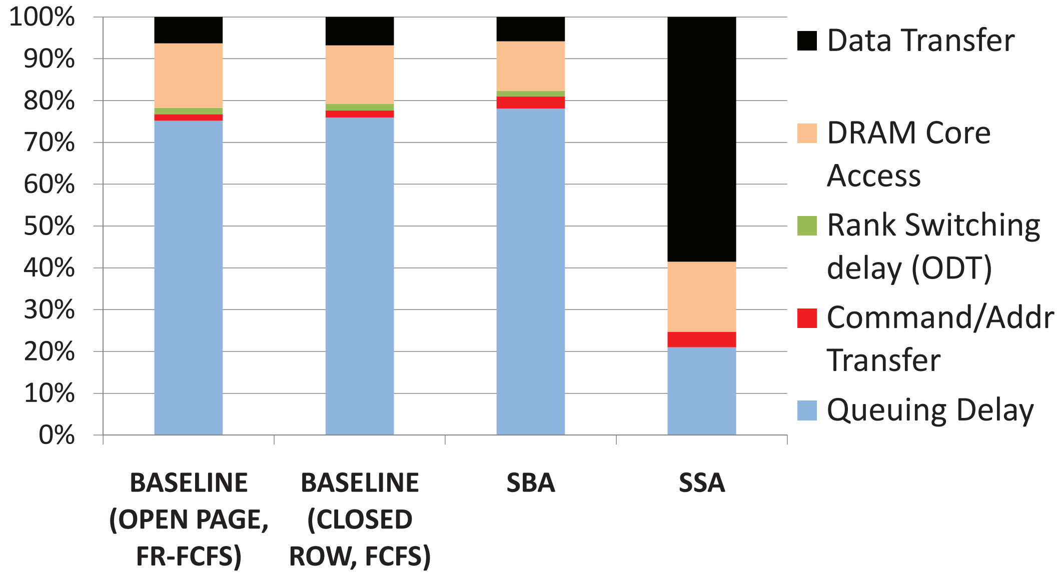


Figure 3.10. Contributors to total memory latency

queuing pressures on memory controllers.

3.5 Related Work

The significant contribution of DRAM to overall system power consumption has been documented in several studies [70, 71, 7]. A majority of techniques aimed at conserving DRAM energy try to transition inactive DRAM chips to low-power states [72] as effectively as possible to decrease the background power. Researchers have investigated prediction models for DRAM activity [73], adaptive memory controller policies [74], compiler-directed hardware-assisted data layout [75], management of DMA and CPU-generated request streams to increase DRAM idle periods [76, 77] as well as managing the virtual memory footprint and physical memory allocation schemes [78, 79, 80] to transition idle DRAM devices to low-power modes.

The theme of the other major volume of work aimed at DRAM power reduction has involved rank-subsetting. In addition to exploiting low-power states, these techniques attempt to reduce the dynamic energy component of an access. Zheng et al. suggest the subdivision of a conventional DRAM rank into mini-ranks [33] comprising a subset of DRAM devices. Ahn et al. [81, 82] propose a scheme where each DRAM device can be controlled individually via a demux register per channel that is responsible

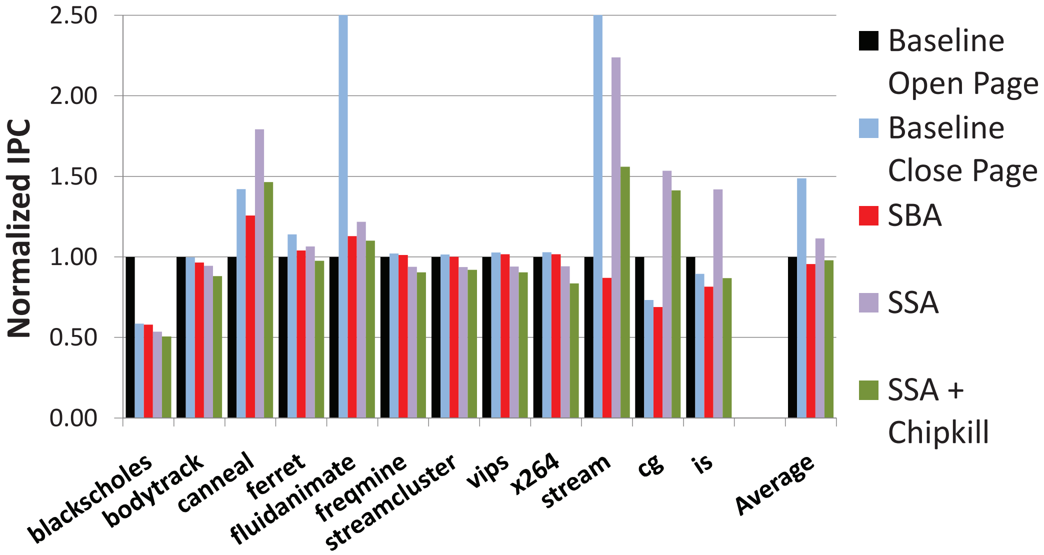


Figure 3.11. Normalized IPCs of various organizations

for routing all command signals to the appropriate chip. In their multicore DIMM proposal, multiple DRAM devices on a DIMM can be combined to form a Virtual Memory Device (VMD) and a cache line is supplied by one such VMD. They further extend their work with a comprehensive analytical model to estimate the implications of rank-subsetting on performance and power. They also identify the need to have mechanisms that would ensure chipkill level reliability and extend their designs with SCCDCD mechanisms. A similar approach was proposed by Ware et al. by employing high-speed signals to send chip selects separately to parts of a DIMM in order to achieve dual/quad threaded DIMMs [83]. On the other hand, Sudan et al. [58] attempt to improve row-buffer utilization by packing heavily used cache lines into “Micro-Pages”.

Other DRAM-related work includes design for 3-D architectures (Loh [84]), and design for systems with photonic interconnects (Vantrease et al. [51] and Beamer et al. [23]). Yoon and Erez [18] outline efficient chipkill-level reliability mechanisms for DRAM systems but work with existing microarchitectures and data layouts. However, to the best of our knowledge, our work is the first to attempt fundamental microarchitectural changes to the DRAM system specifically targeting reduced energy consumption.

3.6 Summary of Contributions

Memory power is becoming an increasingly large contributor to the overall power consumption of large-scale datacenters. A primary reason for this is the “overfetch” of data within memory chips, based on DRAM standards from the 1990s, focused on cost-per-bit, and until now held inviolable by both industrial and academic researchers. In recent years, however, the influence of operational costs (primarily energy) are beginning to rise in the Total Cost of Ownership (TCO) equation, signalling the need for novel architectures designed with modern constraints and requirements in mind, even at the cost of nontrivial initial design effort, and some minor increases in area/cost-per-bit. We propose novel techniques to eliminate overfetch in DRAM systems by activating only the necessary bitlines (SBA) and then going as far as to isolate an entire cache line to a single small subarray on a single DRAM chip (SSA). These techniques enable dramatic reductions in memory energy for both dynamic (6X) and background (5X) components. We also observe that fetching exactly a cache line with SSA can improve performance in some cases (over 50% on average) due to its close-page policy and also because it helps alleviate bank contention in some memory-sensitive applications. Similar architectures may also be adopted for emerging memory technologies such as PCM and STT-RAM, where they can be incorporated into first generation designs, mitigating some concerns regarding the initial design effort required of memory vendors.

CHAPTER 4

HIGH-BANDWIDTH MEMORY CHANNEL DESIGN

4.1 Why We Need Cost-Effective Photonic Memory Channels

An NSF-sponsored blue-ribbon panel [85] estimated that a single terascale node in a 2017 exascale system would have to support a memory capacity of 1 TB and a bandwidth of 1 TB/s. The biggest impediment to building an efficient memory channel that can support such requirements is the fundamental nature of high-speed off-chip electrical interconnects. Neither pin-count nor per-pin bandwidth is projected to scale [26], limiting the number of channels connected to the processor. Per-channel capacity is also limited, since high-speed buses can only support a small number of DIMMs due to signal integrity and power issues [15]. Overcoming these issues to build an electrical memory channel would either be impossible, or prohibitively expensive in terms of energy and cost.

Silicon photonics promise to alleviate many of these problems, and are well suited for the processor-memory channel. They can provide tremendous bandwidth through high-frequency operation and Dense Wavelength Division Multiplexing (DWDM). More importantly, the photonic waveguide can directly couple to optical I/Os on-chip without conversion into the electrical domain, unconstrained by the overheads or limitations of electrical pins. Their pitch is also small enough that multiple waveguides can be run in parallel to further increase bandwidth. While the device-level implementation issues are still an active topic of research worldwide, the technology has matured sufficiently in recent years to command serious interest in the architecture community with regard to potential applications and the accompanying design constraints.

Prior work in this area has advocated the use of specialized photonic memory dies, which will likely increase costs and be impractical for industry adoption due to their invasiveness. In this chapter, we propose a novel design that enables low-cost access to commodity memory dies. Also, careful consideration of various factors, including the presence of low-energy and low-swing interconnects for on-die communication, the presence of low-energy TSVs in a 3D stack, the trimming power reduction when photonic rings are aggregated, number of stacks on the channel, etc. is required to ensure an energy- and performance-optimal organization. We perform a detailed design space exploration to determine this organization.

4.2 Proposed Memory System Architecture

Assuming a theoretical bandwidth of 80 GB/s per waveguide and accounting for some inefficiency in achieved throughput, we provision 16 waveguides to reach the 1 TB/s goal. Each of these is considered an independent "channel", with a memory capacity of 64 GB to reach a total capacity of 1 TB. Each channel is physically organized as 32 dies of 2 GB each, with 16 independent banks per die. We assume a 3D stack depth of 8 DRAM dies [43], requiring four stacks per channel. We use the term DIMM to refer to these four stacks. For the rest of the discussion and evaluation, we will only focus on the design of a single channel.

We propose a novel memory architecture that uses photonics to break the pin barrier on a memory stack, but isolates all photonic components to a dedicated 3D stacked die called the *interface die.* The address and data waveguides only penetrate this die, where conversion between the photonic and electrical domains occurs; all intrastack communication is electrical, either on TSVs or low-swing wires. Read data, for instance, are read out of the array, travel laterally on their local die via conventional electrical interconnects to the nearest TSV, where they travel vertically to the interface die. Here, they are converted to the photonic domain, and shipped off to the processor. Write data, as expected, would follow the same path, but backwards. An example channel with two memory stacks is shown in Figure 4.1. Each memory channel consists of four logical buses - a 16-bit address/command bus, a 32-bit write-data bus, a 64-bit read-data bus, and a 3-bit acknowledgement bus.

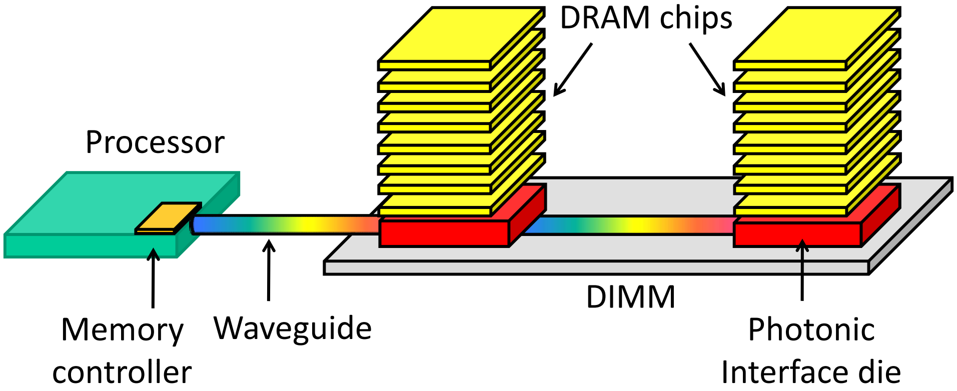


Figure 4.1. Proposed photonic memory system design

Since each waveguide can support up to 67 bits through DWDM, these four buses can be combined into two physical fibers, one outbound (address/command + write-data), and one inbound (read-data + acknowledgement). Note that the wavelengths used for read and acknowledgement are exclusive, and the buses are completely independent, though physically on the same fiber. Address and write are organized similarly.

4.2.1 Layout of Ring Resonators

We define a *photonic stop* as a set of wavelength-selective rings. The read-data bus, for example, has 64 rings per stop. A DRAM die can contain one or more photonic stops. PS denotes the number of photonic stops. Multiple stops would be distributed symmetrically on the DRAM die, similar to nodes in a hypothetical H-tree network. If two photonic stops are present, each services subarrays in half of the chip; if four stops are present, each services a quarter of the chip, and so forth. Communication between the DRAM arrays and the nearest photonic stop is fully electrical.

4.2.2 Address/Command Bus

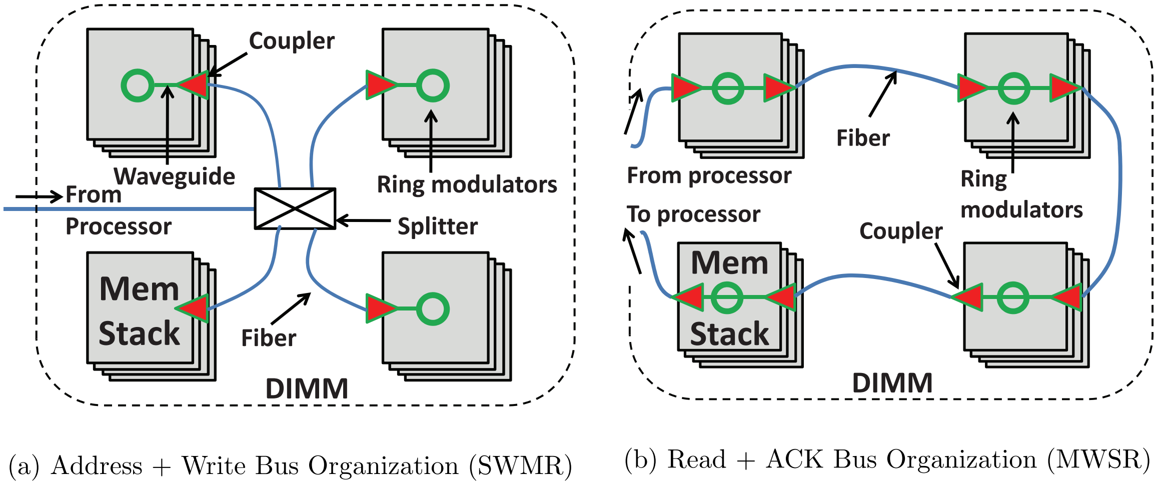
In the interest of simplicity, we assume that the address and associated command signals are broadcast to all stacks in the channel. Photonic broadcasts are complicated since every ring coupled to the waveguide has to be fine-tuned to not simply remove all light during demodulation, but only remove a fraction of the total optical power, allowing the rest of the photons to pass downstream. A simpler approach to building a single-writer, multiple-reader (SWMR) photonic interconnect is to use a Y-splitter to divide the incoming laser power among multiple fibers, as shown in Figure 4.2(a). While it introduces some insertion loss, only one splitter is used on each DIMM, making the overhead tolerable. Modulated light reaches every photonic stop in the channel, where it is converted to the electrical domain before being decoded. The traffic on the address/command bus is fixed for a given channel bandwidth (cache line requests per unit time). Accordingly, we provision a small 16-bit bus (fast and pipelined) to keep utilization high and energy consumption low.

4.2.3 Write-Data Bus

Since write traffic is typically lower than reads (about 25% of all traffic in our benchmarks), we provision only half as much peak bandwidth as for reads. With a separate dedicated data bus for writes, we avoid contention and turnaround issues with respect to read operations.

4.2.4 Read-Data Bus

The *read-data* bus is a multiple writer, single reader (MWSR) system, implemented with a daisy chain as shown in Figure 4.2(b). Light originates at the processor, travels on optical fiber between memory stacks and silicon waveguides on-chip, and arrives back at detectors on the processor. It passes by, and is coupled to, several



(a) Address + Write Bus Organization (SWMR)

(b) Read + ACK Bus Organization (MWSR)

Figure 4.2. Memory channel organization photonic stops in the channel along the way - these can be distributed among the many stacks on the channel. One of the photonic stops (as determined by some form of arbitration) modulates the light to carry the appropriate cache line data, and this is detected at the processor.

4.2.5 Acknowledgement Bus

ACK/NACK signals may be required for novel access protocols (Chapter 5), for which we provision a 3-bit MWSR bus.

4.2.6 Photonic Power Loss

As the number of DRAM stacks on the channel increases, or the number of rings on each stack increases, there is an increase in the total loss, increasing the input laser power required. With a small number of DRAM stacks per channel (a maximum of four in this work), however, total loss can be contained to about 13 dB, as shown in Table 4.1, based on the literature, and projections from device-physics work currently in progress at HP Labs. As we scale beyond this, solutions like the *guided photonic bus* have been shown to be effective in actively guiding power to only the target DRAM stack or die [23]. This will reduce the loss for a given source-destination pair in the system, providing further scalability in spite of the daisy-chained nature of the architecture.

Table 4.1. Photonic parameters (4-stack channel, 1 stop per stack)

Component	**Loss**	**Qty SWMR**	**Qty MWSR**
Si Waveguide	0.3 dB/cm	3.25	6.5
Fiber	0.4 dB/Km	0.01	0.01
Coupler	1.0 dB [86]	2	10
4-way splitter	8.0 dB	1	0
Near ring loss	0.17 dB	0	6
Far ring loss	0.001 dB	48	262
Total Loss	dB	11.03	13.24

Laser Efficiency 32% Receiver Sensitivity 1.08 A/W
Receiver Current Swing 20 μA

4.3 Methodology

We use an in-house cycle-accurate DRAM simulator for performance studies. We faithfully model the memory controller queue, address/command bus, data read bus, data write bus, bank contention, and refresh. We use a close-page row-buffer policy, as is the norm today for industry standard servers from all major vendors in an effort to improve performance in typical workload mixes with low locality [87, 88, 89]. We use a simple FCFS scheduling policy. We adopt address mapping policies from Jacob et al. [38]. On-chip latency numbers, and DRAM latency/area numbers were obtained from CACTI 6.5 [65]. Simulated system specifications are listed in Table 4.2. We use Simics [63] to execute the PARSEC [66] benchmarks ("sim-large" dataset) and Stream [68] on an 8 core machine with private 16 KB L1s per core and a shared 256 KB L2 cache. Every simulator run consists of 5 million DRAM accesses, and we report average memory stall cycles and energy consumption per cache line access. In order to analyze a futuristic memory system with a large physical address space and heavy traffic with high throughput, however, most of these benchmark workloads are unsuitable since they have small data sets and relatively low bandwidth requirements. As a result, most of our analysis uses a synthetic traffic generator with a tunable injection rate and tunable fraction of write operations feeding the DRAM simulator. Prior work [22, 23] has shown that memory access patterns are becoming increasingly random and will likely become more so with increasing socket, core, and thread counts. All of our synthetic analyses therefore use a random address pattern. A new request is injected into the channel once every C cycles on average, under a Bernoulli process to ensure that traffic is bursty [90]. We also artificially increased burstiness to stress-test the channel. With 64-byte cache lines, a 64-bit channel, and dual data rate, every data transfer takes 4 cycles. To prevent saturation, therefore, $C = 4$ is the maximum allowable request rate. This rate corresponds to the target of 1 TB/s described in the previous subsection. We show results for C ranging between 3 (saturation) and 10 cycles, stress-testing the channel to the maximum extent possible.

The energy consumption of the channel consists of 3 components - read-data bus, write-data bus, and address/command bus. We report total energy as the sum of the average read and write energy per cache line, weighted by the fraction of requests

Table 4.2. System specifications

Parameter	Value
Technology	Yr 2017, 22nm, 5 GHz
DRAM Chip	16 banks, 60 sq. mm
DRAM Capacity	32 Chips for a total of 64 GB per channel
Bank Access Latency	7 ns
On Chip Ntk Latency	5 ns
Queue Size	64 entries each for RD and WR

that are reads or writes, and the address/command bus energy. For most of our energy analysis, we aggressively assume the peak bus utilization of about 75% (50% for writes), just below saturation. This shows photonics in the best light possible, allowing maximum amortization of its static energy.

On-chip interconnect energy numbers are obtained from CACTI [65]. We use low-swing wire design points studied by prior work [91]. Low voltage swing alternatives represent a mechanism to vary the wire power/delay/area trade-off. There is little silicon overhead to using low-swing wires since there are no repeaters and the wires themselves occupy zero silicon area. A low-swing pair does require special transmitter and receiver circuits for signal generation and amplification but these are easily amortized over moderately long wires. Reducing the voltage swing on global wires can result in a linear reduction in power (we assume a voltage swing of 100 mV [92, 61]). In addition, assuming a separate voltage source for low-swing drivers will result in quadratic savings in power (we assume a supply voltage of 0.4 V [92, 61]). However, low-swing wires suffer significantly greater delays than repeated full-swing wires and thus cannot be used over very long distances. We model these effects in detail, and account for their energy and power as described in the CACTI technical report [93].

We use measured or estimated energy numbers for the photonic components based on published work [50, 94, 48, 51] and device-physics work currently in progress at HP Labs. For our initial graphs, we assume that the trimming power scales up linearly with the number of rings. In a subsequent graph, we also consider insight from a recent paper [95] that shows a reduction in trimming power when rings are aggregated. Laser power numbers are based on losses calculated in Table 4.1. These parameters are all listed in Table 4.3.

4.4 Energy Considerations

The proposed design helps reduce energy across several fronts, compared to state-of-the-art photonic DRAMs:

1. As described previously, photonic components consume nontrivial amounts of power for trimming and other analog operations, independent of bus activity. The key take-away for our design is that photonic resources cannot be over-provisioned without a significant energy impact. It is essential to provide the minimal photonic resources required to exploit the benefits of the technology, and run this at as high a utilization rate as possible to amortize the static energy costs over many useful transfers per unit time. On the other hand, electrical interconnects can be over-provisioned because their static energy consumption is relatively low. An efficient design should therefore ideally use photonics for heavily utilized, shared channels, while more localized communication with relatively low utilization is best handled electrically. The interface die helps do exactly this, providing substantial static energy savings by eliminating the need to have photonic rings on every die, while still breaking the pin barrier.
2. The use of low-swing wiring [92] for on-die traversal provides dual benefit. First, it reduces the dynamic energy consumption of electrical wires by 4-8X, saving

Table 4.3. Energy parameters

Component	**Power/Energy**
Read Laser	1.22 mW
Write Laser	0.49 mW
Addr/Cmd Laser	0.24 mW
Modulator	47 fJ/bit
Receiver	440 μW/bit
Thermal Tuning	250 μW/ring
Full Swing Wire	9.7E-14 J/mm
Low-Swing Wire + Senseamps	1.8E-14 J/mm
TSV	7.8E-14 J/bit [96]

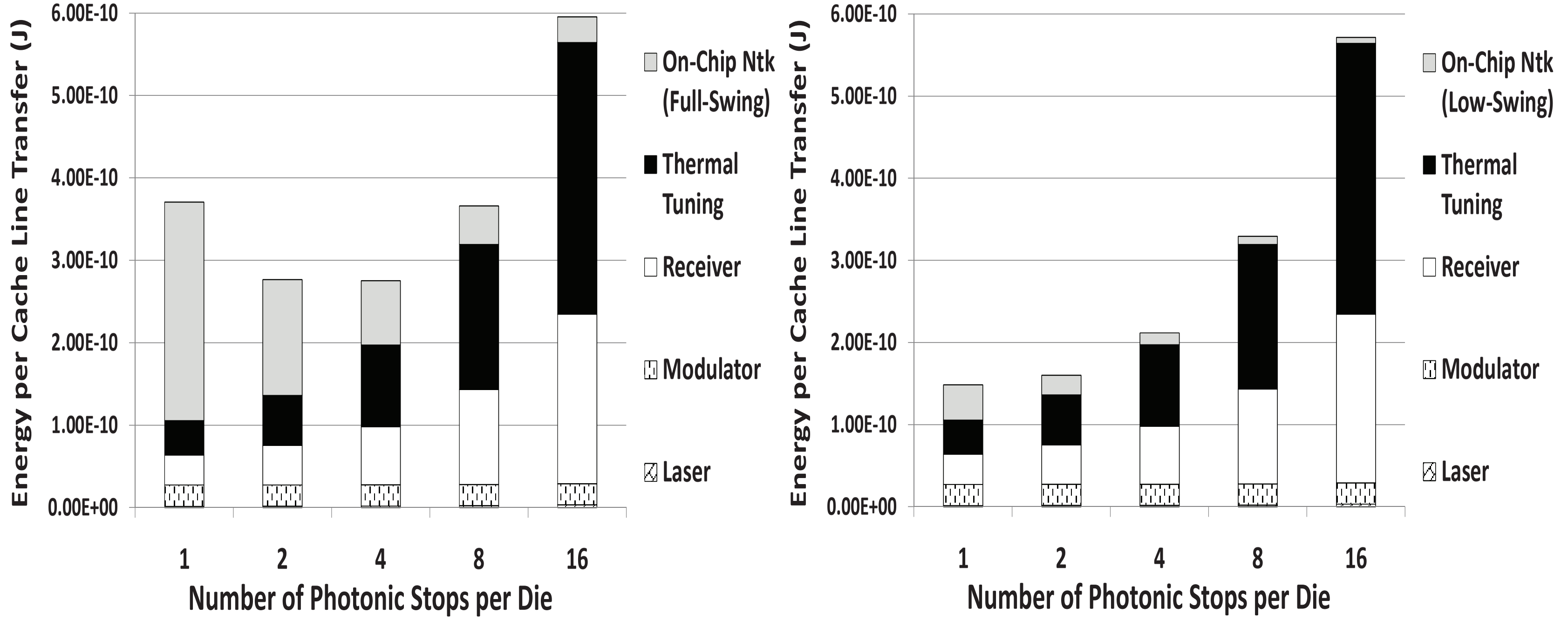
electrical communication energy. Second, the reduced electrical communication costs allow the optimal design to have fewer photonic stops overall, reducing the static energy costs of running these stops.

3. Having all photonic components on a single die reduces the number of fiber-waveguide couplers (which can be relatively lossy components) required by a factor of 8 (for an 8 memory die stack), helping reduce total power loss. Low-cost TSVs [96] are used instead for inter-die communication. This helps reduce the input laser power required.
4. Recent work [95] has shown that trimming power is dependent on the total area being heated, and aggregating rings on the interface die helps reduce some of this energy.

4.4.1 Prior Design

We start by evaluating a *single die* system, *i.e.*, the channel is attached to just one DRAM die, with no 3D stacking or daisy-chaining, similar to the state-of-the-art PIDRAM design of Beamer et al.[23]. The entire cache line is fetched from subarrays connected to one of the photonic stops. Figure 4.3(a) shows the average energy consumption per cache line transfer in such a system, with a varying number of photonic stops, PS.

We see two conflicting effects as PS increases. First, there is a reduction in the electrical energy spent in transferring data between the array and the photonic stop since the distance to the nearest stop is lowered. On the other hand, there are now many more rings on the die, and since only one stop can actively be transmitting data at any given time, there is increased static power overhead, increasing the average photonic energy per useful data transfer. These two factors balance out at 4 stops, a trend similar to the one described in [23]. Note that the actual balance point is dependent on several system parameter assumptions like the die area, static power overhead, etc. In general, however, this argues for specialized memory with photonics going well into the die, and rings spread throughout the die.



(a) With full-swing on-chip wires

(b) With low-swing on-chip wires

Figure 4.3. Determination of optimal PS in a *single-die* channel (1 DRAM die, no stacking or daisy-chaining)

4.4.2 Using Low-Swing Wiring

As Figure 4.3(b) shows, the trend starts to shift when the on-chip communication is made more efficient through the use of low-swing wires, since this emphasizes the overhead of increased photonic resources. In this configuration, a single-stop design is energy-optimal. In absolute terms, the best low-swing configuration (1 stop) has an energy consumption 46% lower than the best full-swing configuration (4 stops), clearly arguing for minimal photonic use with more efficient on-chip communication.

4.4.3 Basic 3D Extension

The analysis presented so far was for a single-die DRAM channel, but to allow capacity scaling, each channel will likely support more than a single die. A first-step towards scaling capacity is 3D stacking of DRAM dies [43, 44]. The straightforward approach to building photonic 3D DRAM would be to simply stack the photonic DRAM dies. The optical signal would now have to traverse dies at multiple levels, traveling on waveguides on each die, with some form of vertical coupling (perhaps free space optics) to go between dies. While the increased loss can be handled by hierarchical power guiding [23], there are now even more rings idling away while exactly one stop on exactly one of the dies in the stack is actually modulating light with useful data. For example, with an 8-deep DRAM stack, and 4 stops per die, there are a total of 2048 rings in the channel, with only 64 operational at any given time. This further emphasizes the importance of the photonic components' static power over the dynamic energy in electrical wires. The energy - PS curve shifts accordingly, and a system with a single photonic stop per die becomes energy-optimal, with *both* low-swing and full-swing electrical interconnect assumptions (we do not graph this due to space constraints). Note that this design still has 8X more rings (one stop on each of the eight dies) than can be usefully employed at a time, idling away static power due to under-utilization. The most efficient 3D configuration consumes 2.4X the energy of a single die channel, for 8X capacity gain. This is clearly inefficient, though still better than a single photonic die per channel.

4.4.4 Proposed Single-Stack Design

We next consider our novel model with a single stack of DRAM chips. Photonic components are only placed on the interface die and TSVs are used for inter-die communication. Now that we only have one photonic die in the DRAM channel, the decision on the number of photonic stops on the die is almost exactly the same as the *single-die* system discussed previously, except that the nonphotonic energy now includes TSVs (this vertical communication came virtually for free with photonics due to their distance-independent energy consumption). The energy-optimal design point continues to be either 4 photonic stops with full-swing wires or 1 stop with low-swing wires. This final design point (single photonic die, 1 stop per die, low-swing wires + TSVs) consumes 48% less energy than the best simplistic 3D design (8 photonic dies, 1 stop per die, low-swing wiring + free vertical communication), due to the reduced number of photonic stops, despite the additional TSV energy. There is no loss in performance between the fully-optical 3D stack and our proposed design since the TSVs can handle very high bandwidths [96] and the channel still gets its provisioned bandwidth. There is also no capacity loss, and no need for modification of the memory dies. This argues that despite photonic energy being virtually distance-independent, the static energy costs mean that photonic penetration should be minimized.

4.4.5 Proposed Daisy-Chained Multistack Design

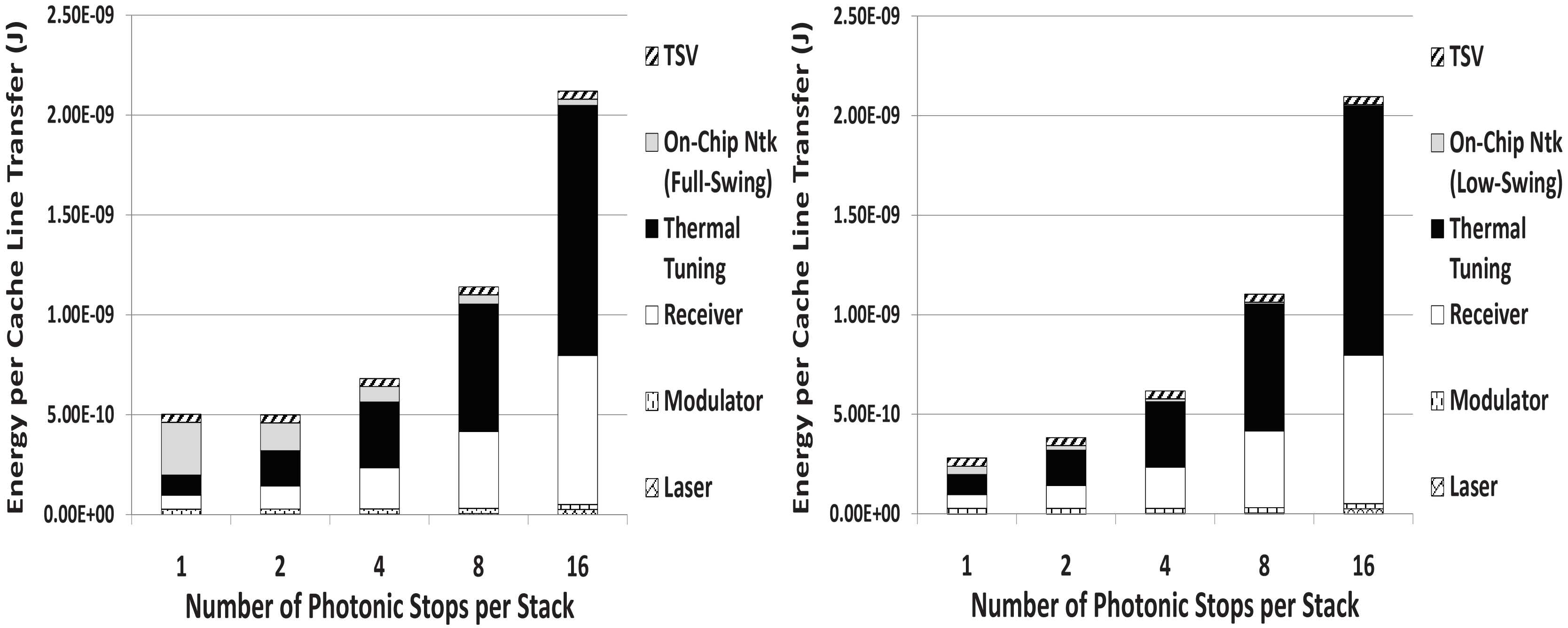
Finally, we explore a more realistic, expanded channel, with four stacks daisy-chained together on the same waveguide, each stack comprising 8 memory dies [43]. The operation of the channel is similar to the single stack design, except there are now more stops coupled to the waveguide. As before, only one of these can actively modulate the data waveguide at any given time, and this is handled by some form of arbitration in the access mechanism. Such a design provides two major benefits: first, there is increased capacity on a single channel, a critical metric in servers for increased scalability. Second, it provides a large number of independent banks that can operate in parallel to keep the shared photonic channel well utilized, improving the photonic energy characteristics, as well as improving overall performance. However, to enable a workload-independent comparison, we do not quantify this benefit, instead assuming a fixed utilization figure for all the designs under study.

We repeat our photonic stop study in a system with four DRAM stacks (a total of 32 dies) on a single waveguide, the likely use case for future optically connected memory. As seen in Figure 4.4, once again, the static energy of the photonic components dominates. It is clear that as the number of dies connected to the channel rises, it is more energy efficient to use a smaller number of photonic stops per stack. This supports our hypothesis that photonics should only be used to overcome the pin barrier and cheap electrical interconnects should be used within a stack. This design point (4 stacks, 1 stop per stack, with low-swing interconnects and TSVs between dies) still consumes 23% less energy than the best fully-optical 3D extension of state of the art photonic DRAMs (1 stack, 8 stops per stack, with low-swing interconnects and photonic vertical communication, no TSV energy), while providing 4X capacity, and the potential for improved performance and further reduced energy consumption due to the increased number of banks.

4.4.6 Sensitivity to Ring Trimming Power

Recent work has indicated that the trimming power may be less dependent on the absolute number of rings, and more correlated with the total die area, ambient temperature, and other factors [95]. However, that analysis was for large dies (300-400 mm^2) with several thousand rings spread across the die, unlike our 60 mm^2 DRAM dies with a few hundred rings at most, concentrated in “stops”. In light of these differences, we do not expect our trimming power to be independent of ring count, as claimed in that paper. We developed a simple thermal current flow model with rings laid out in a 2D array and a heater in the center of that area. In steady state, the power that the heater needs to supply equals the total heat that gets dissipated from the array, which is proportional to the *perimeter* of the array. There is negligible heat flow in the +Z and-Z directions since a thermal insulator is expected to be present to prevent heat loss. Based on this model, we arrive at an aggressive estimate for ring trimming power of about 50μW/ring for a 64-ring stop.

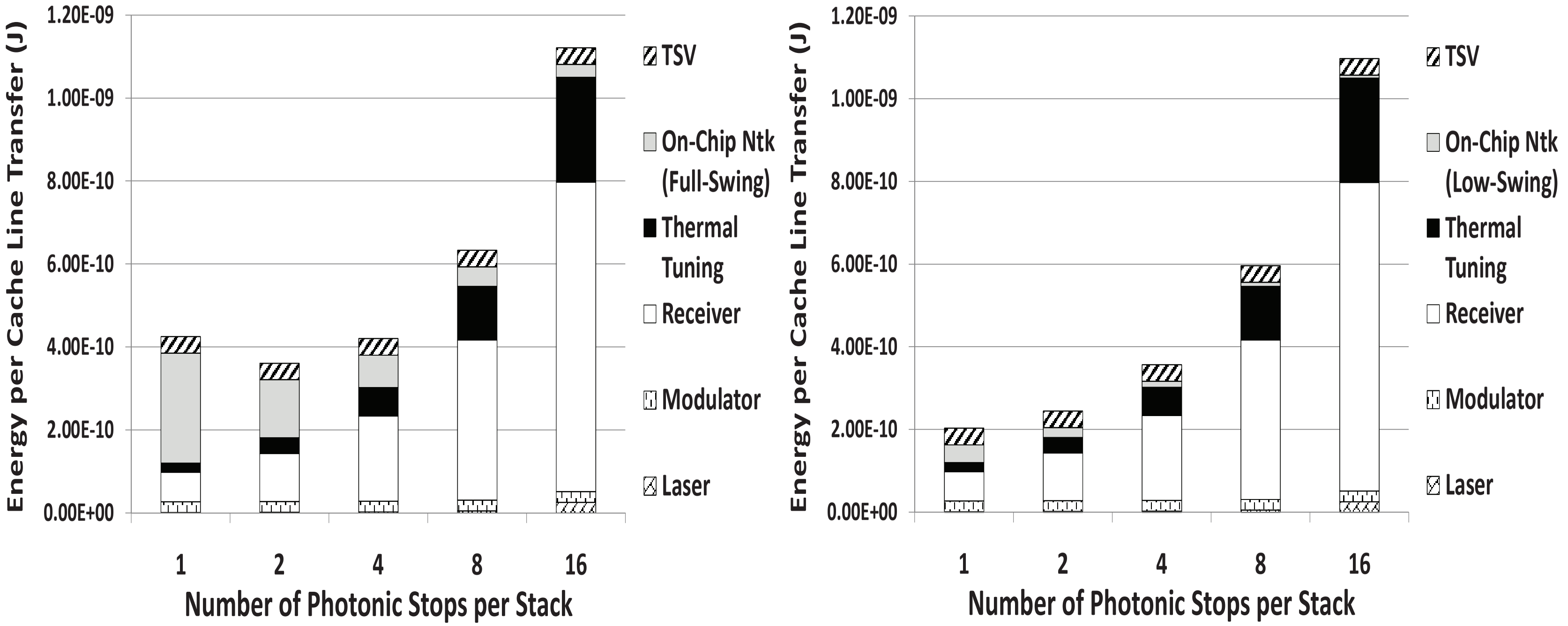
Figure 4.5 shows an energy analysis for our final multistack design point based on this estimate. As expected, reducing the trimming power de-emphasizes the photonic energy component, though other static components such as the analog receiver



(a) With full-swing on-chip wires

(b) With low-swing on-chip wires

Figure 4.4. Determination of optimal PS in a daisy-chained *four-stack (32 die)* channel



(a) With full-swing on-chip wires

(b) With low-swing on-chip wires

Figure 4.5. Energy analysis with aggressive trimming power assumption - 50μW/ring circuitry now begin to show up more dominantly. As a result, the optimal point may shift towards having more photonic stops. However, we continue to see that a single photonic stop is optimal when using low-swing wires.

4.4.7 Impact of Traffic

As we reduce injection rate from one in 4 cycles to one in 10, utilization falls from 75% for reads/50% for writes to 30% for reads/20% for writes. The static photonic energy has to be amortized over fewer useful transfers, increasing energy consumption per cache line by as much as 88%, arguing for a design that maximizes utilization.

4.5 Performance Considerations

Once we have defined the "photonic power budget", *i.e.,* the number of photonic stops on a memory stack, there are two ways to organize the rings within a photonic stop, as shown in Figures 4.6 and 4.7.

The *concentrated architecture* provides the entire bandwidth to a single request, significantly reducing serialization and queuing latencies. However, it may require a relatively large on-chip electrical traversal to go from the arrays to the stop, perhaps increasing energy consumption. The *distributed architecture* splits the total available bandwidth among several "mini-stops" (through mutually exclusive wavelengths), simultaneously servicing multiple requests. Since arrays are closer to photonic stops, electrical traversal is reduced, but individual accesses suffer from increased serialization and queueing delays. Going from 1 stop to 2, 4, and 8 "mini-stops", net energy reduced by 7%, 10%, and 12%. Corresponding latency increases were 10%, 30%, and 67%, which may also affect system-level power consumption. We believe that the modest energy savings is not worth the significant loss in performance.

Note that we could, instead, stripe a cache line across multiple subarrays so that each mini-stop and its corresponding subarrays can handle a portion of the cache line - there would then be no impact on performance in moving from a centralized to a distributed organization. However, assuming that the subarray size is fixed, the extent of overfetch increases as more subarrays are involved in servicing a cache line, so we do not consider this method of operation. Based on this analysis, we believe

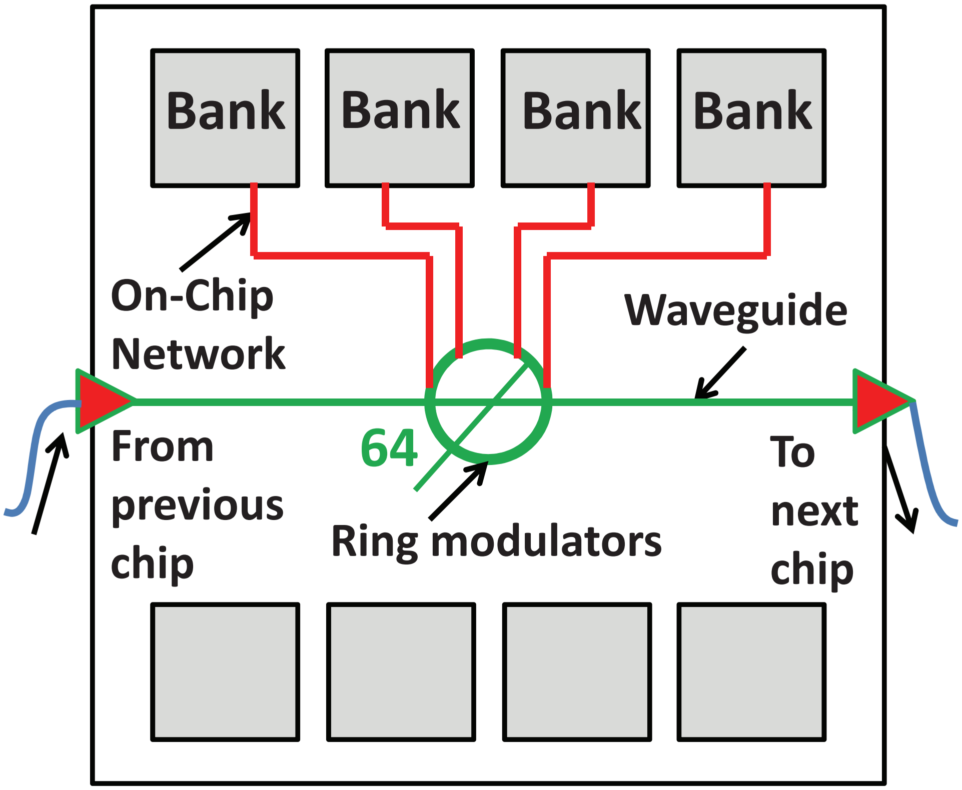


Figure 4.6. Rings concentrated in one stop

that rings should be concentrated into one centralized photonic stop when attempting to optimize both performance and energy.

4.6 Cost Considerations

The architecture described in this chapter has virtually no impact on the area and cost of the memory dies themselves, and enables efficient and relatively low-cost photonic access to commodity memory systems. First, since photonic components do not enter the memory dies themselves, there is no disruption to array density. Additionally, the same dies can be used with either large-scale photonic memory systems or small-scale single-user/handheld conventional electrical memory systems, eliminating the need for custom photonic dies and multiple product lines. Similarly, the interface die will only need minor modifications to be customized for use with different kinds of memory (DRAM, PCM, STT-RAM, etc.). This will help increase manufacturing volumes for the interface die. Finally, heavily optimized DRAM

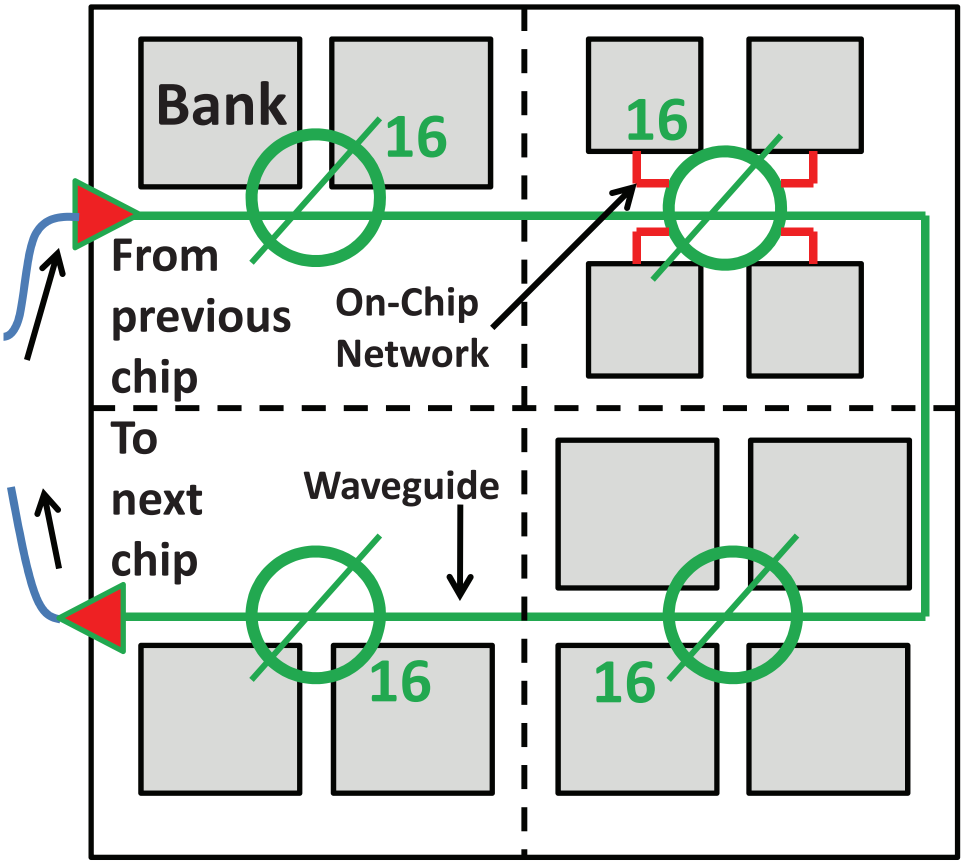


Figure 4.7. Rings distributed as "mini-stops"

processes need not be tweaked to accommodate the needs to photonic components. All of these are very important considerations for the cost-sensitive memory industry.

4.7 Impact on Thermals

We used Hotspot 5.0 [97] to perform thermal evaluation of our designs based on peak activity factors. The baseline 3D stack with 8 DRAM dies settled at a temperature of 320.15K. Adding a photonic interface die, including a layer of SiO_2 required to build optical components, resulted in a slight rise in temperature at the DRAM to 320.21K. The less than 0.1K increase in temperature will have practically no impact on refresh. Another interesting side-effect of the SiO_2 layer is that it helps thermally isolate the rings, reducing the amount of heat that needs to be supplied to keep the rings correctly tuned. Note that prior work on 3D DRAM architectures [84, 96, 98] has assumed DRAM stacked on top of processing cores, making thermals much more of a concern than in our study with only DRAM dies.

4.8 Related Work

Optically connected memory has only recently received the attention of the research community. Hadke et al. proposed the OCDIMM architecture [99], using photonic interconnects to replace the electrical channels in an FB-DIMM system, resulting in better performance, energy, and scalability characteristics than electrical interconnects. However, that design did not fully exploit two major advantages of silicon nanophotonics - large bandwidth density through DWDM and the ability to break the pin barrier. We address both these concerns in our design. Corona [51] considered the system level implications of photonic technology and briefly explored optically connected 3D memory stacks, but did not analyze in detail the performance or energy tradeoffs involved in such designs. Beamer et al. proposed the specially designed PIDRAM chip [23], with aggressive photonic penetration well into the DRAM chip. We considered this design in detail and provided comparisons in Section 4.4. Several papers have explored leveraging photonic interconnects, especially for on-chip communication [100, 101, 102, 103], which has an entirely different set of tradeoffs. 3D stacked DRAM designs have also been explored by several papers [84, 96, 98]. However, these study DRAM stacked directly on top of processing cores, thus dealing with much smaller scale memories, with a different set of constraints than ours, including capacity, thermals, etc.

4.9 Summary of Contributions

The primary limitation to scalability of the processor-memory channel is the poor scaling of off-chip electrical interconnects. Photonic technology, when used prudently, has the potential to address this, by providing fast, high-bandwidth communication within a reasonable power envelope. However, all prior work has integrated photonic components into individual memory chips, disrupting dense array layouts, requiring special optics-compliant dies and multiple product lines, and necessitating tweaks to the highly optimized DRAM manufacturing process. As a result, those innovations are unlikely to be adopted by the cost-sensitive memory industry. This chapter makes a fundamental contribution: it proposes the first architecture that allows photonic access with low-cost commodity memory. It shows that in the expected usage scenario for large-scale memory systems, the real strength of photonic interconnects lies only in breaking the pin barrier, and that further penetration into the DRAM die is actually detrimental to energy-efficiency. It proposes a DIMM design where memory chips and a separate logic interface die are organized in a 3D stack. It isolates all photonic rings to the interface die. Off-chip transmission between the processor and the interface die is fully photonic. The interface die performs opto-electrical conversion, and all intra-stack communication is purely electrical, on TSVs and low-swing wires. The chapter quantifies the extent of optical penetration that is optimal for a 3D stack of memory chips. The energy- and performance-optimal photonic design is strongly influenced by several factors, including the total number of memory dies on the channel, 3D stacking, channel organization, the on-die wiring technology, channel utilization and traffic characteristics, etc., several of which have never been considered before. Our final daisy-chained design provides at least 23% energy reduction compared to a fully photonic 3D stack, with 4X capacity, and potential for performance improvements and further energy reduction due to increased bank count, all without impacting the design of high-volume low-cost memory chips.

CHAPTER 5

STREAMLINED MEMORY ACCESS PROTOCOL DESIGN

5.1 Why We Need To Rethink The Memory Interface

We believe that a memory system overhaul should not be limited to the DRAM chip or the channel's physical interconnect, but should also include a redesigned interface protocol. Aided by rapidly advancing technologies such as silicon photonics and 3D stacking, future memory systems can potentially support very large capacities at theoretically very large peak bandwidths. The next big impediment to scalability of the memory interface then becomes the tight control held by the memory controller over every micro-operation of the memory system. With DRAM, for example, every memory access involves the issue of several commands such as ACT, CAS, PRE, WAKEUP, SLEEP, etc., while being cognizant of as many as 20 different timing constraints such as t_{FAW}, t_{RRD}, t_{WTR}, t_{RTP}, etc. for scheduling. Nonvolatile memory technologies such as PCM or STT-MRAM will come with their own maintenance requirements (wear-leveling, drift management, etc.) and specialized coding requirements. All these factors significantly increase complexity. In the current paradigm, all of these will have to be hard coded into the memory controller, requiring explicit processor support every time a new memory technology is introduced, or even when new value-add features are introduced into existing memories. It also creates increased pressure on the address/command bus, and will require large amounts of state to manage potentially hundreds of independent banks, with further complexity to handle heterogeneous systems with some combination of DRAM and nonvolatile memories.

A new interface is required to overcome these bottlenecks, and efficiently translate theoretical peak bandwidth to actual sustained throughput, and is a vital component in designing a scalable interface for terascale memory. The introduction of a 3D-stacked *interface die* on the memory stack opens up the interesting possibility of shifting some memory controller functionality to the interface die, streamlining the operation of the memory system. The on-processor memory controller can simply dispatch an address in a request packet and eventually receive a data response packet in a speculatively reserved time slot. Routine operations such as precharge, refresh, and scrub, low-level functions such as timing parameter control, and any other technology-specific functions such as special coding techniques can be moved to the memory modules. The processor-side memory controller may retain only a few important functions, such as exploiting workload information and state to handle request reordering or enforce scheduling policies for fairness, QoS, etc. In this chapter, we propose such an access mechanism and protocol, with a novel slot-based approach that offers low complexity and maximizes bandwidth utilization, for a minor increase in memory latency.

5.2 Interface Design

We start by making the memory more autonomous – the memory controller deals with it at a much higher level than existing protocols. We propose moving all low-level device management to the interface die in the memory stacks. Essentially, the *only* information the memory module requires is the address of the block being requested and the kind of operation - read or write. This is sent to the memory pool in a *request packet*, over the *address channel*. The memory modules then decode this packet, and the appropriate module fetches the requested block. This is returned to the memory controller via a *data packet* over the shared *data channel*. Using such a shared resource typically involves arbitration, leading to nontrivial performance penalties. In this particular case, however, we are presented with a special scenario where every transfer on the data bus occurs due to an explicit request by a single entity – the memory controller. Coupled with the fact that close-page memory organizations offer more deterministic access latencies, time slots on the data bus can be reserved apriori, obviating the need for the memory modules to go through an arbitration stage before every data transfer. Unlike conventional DDR systems, there is not a slew of commands from the controller to handle every micro-operation (bank activation, column select, precharge, etc.).

5.2.1 Basic Operation

As discussed previously, sustaining high bandwidth is a fundamental focus of the redesigned interface for both performance and power reasons. We therefore make the availability of data I/O at the processor-side memory controller on the return path the minimum unit of arbitration. Every incoming request (for one cache line) is placed in a queue of finite length on the processor chip and arbitrates for a *slot* to use the data I/O for the eventual response. The width of the time slot is determined by the size of the cache line and the width of the data path. With 64-byte cache lines and a 64-bit data channel, for example, one request can ideally be serviced every 4 cycles (assuming a fully pipelined data bus, with dual data rate transmission), yielding a slot that is 4 cycles wide. The position of the reserved slot is determined by the total "memory latency" (ML cycles, say) - comprising the command transfer, bank access, on-chip network access, and data return latency. Common timing constraints such as t_{RAS}, t_{RCD}, t_{CAS}, etc. are subsumed under ML and handled by the stack controller. The earliest available slot that is at least ML away is reserved for the request (see Figure 5.1). The request is then issued *just-in-time*, *i.e.*, the address transfer, bank access, on-chip network traversal, etc. all occur timed in such a way that the data can

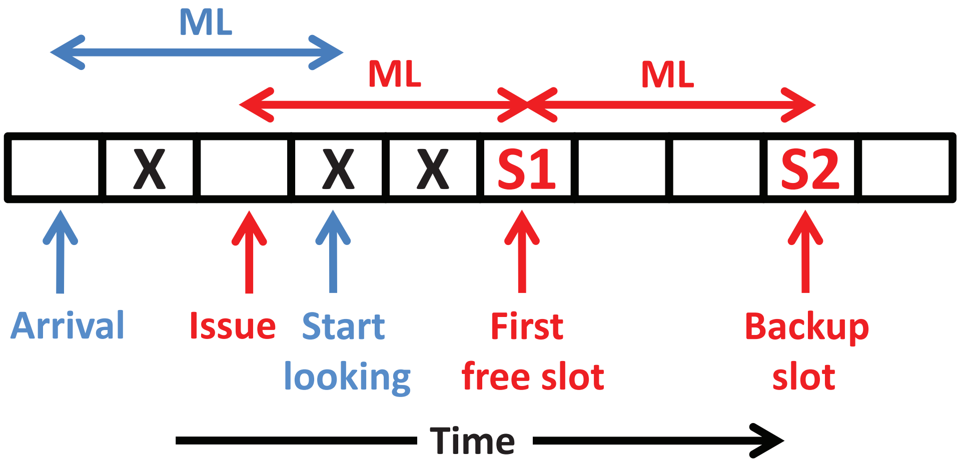


Figure 5.1. Proposed slot reservation scheme

use the processor I/O at exactly the reserved slot; the resources for these preceding operations are automatically reserved, eliminating the need for separate arbitration. There also is not a need for an accompanying tag to help match the response to the corresponding request. Similarly, the DIMM is also aware that it must provide its data response a fixed latency after it receives the request – so the request need not carry a deadline with it. The memory controller and the interface die can exchange information such as the value of ML for each memory module, regardless of the technology used, through the use of control registers during boot-time.

5.2.2 Handling Conflicts

Clearly, the above design works well in the expected usage scenario - when the latency within the memory is consistent and predictable, and where bank conflicts and timing constraint violations are uncommon because of the large number of banks. However, conflicts can occasionally happen. For example, in the interface just described, a new *slot* is alloted every 4 cycles. Even if two requests go to the same DIMM, they can still be completed with a 4-cycle offset because of bank-level parallelism on the DIMM. But if two requests go to the same bank, the second request cannot start until after the first has completed. There might also be other constraints that hold the access up, such as refresh, scrubbing, low-power wake-up, t_{FAW}, etc. These prevent the request from having its data ready at its expected deadline. The memory controller on the processor die could not have anticipated this because, by design, it no longer deals with the minutiae of per-bank state.

To handle such situations, we introduce the following novel mechanism. In addition to the data return slot described above (called Slot 1, say), each request speculatively reserves a second slot, called Slot 2 (see Figure 5.1). Slot 1 is chosen to be at least ML away in time from the cycle of issue, including the *uncontended* bank access time, exactly as before. If, for whatever reason, the bank was busy at that time, a negative acknowledgement (NACK) is returned by the memory module in Slot 1 (on the acknowledgement bus), and the actual data follows in Slot 2. This wastes Slot 1, reducing effective bandwidth, but this scenario is infrequent enough to have minimal impact. Note that Slot 2 has to be placed far enough away that if data is indeed returned in Slot 1 (the common case), the speculative reservation can be cleared in time for a subsequent request to use it. The gap between Slot 1 and Slot 2 has to therefore be at least ML. This results in some additional latency for Slot 2 returns, but ensures that it is not wasted in the common case. Again, Slot 2 need not be explicitly conveyed with the request packet, and is some previously agreed upon (through control registers) time from Slot 1. The scheme therefore still works without the overheads of tags.

5.2.3 Other Scenarios

A contrived example scenario that is yet to be considered is when a long series of requests are all targeted at the same bank. In this case, neither Slot 1 nor Slot 2 will be able to return data beyond a certain number of requests since accesses will essentially have to be spaced apart by a time equal to the bank access time (and not the 4 cycle slot). In such a situation, NACKs are returned in both Slot 1 and Slot 2, and the request has to be retried at a future time. The memory controller backs off for a fixed time, and then arbitrates for a slot and issues the request. This process can potentially be repeated multiple times, consuming resources in the form of queue occupation, wasted return slots, and address re-transmission energy, but is infrequent enough to not impact overall performance.

5.2.4 Handling Writes

As in most modern memory systems, we assume that writes are buffered at the memory controller, prioritizing read requests. When the write queue fills up to a predetermined level, a burst of write requests is issued. This helps avoid frequent “turnaround” delays in the on-stack interconnect (note that there is no bus turnaround since we use a dedicated write bus). It also reduces the frequency of nonuniformity in access time that might arise due to DRAM timing constraints if writes and reads are interleaved arbitrarily [38]. Write data are sent out over a relatively narrow data bus over multiple cycles. If the bank is free, the write operation is performed. If not, a NACK is sent back (on a prereserved slot on the ack-bus) and the request is retried, wasting energy, but such situations are, again, infrequent. Our evaluations show that writes had to be refused in only about 5% of requests, and are almost always completed with the first retry. Alternatively, such requests could be buffered on the memory module, and NACKed only if that buffer is full. We do not consider this optimization in this work.

5.2.5 Handling Heterogeneity

With the new interface, the memory controller only needs knowledge of the access time ML of the memory modules to reserve an appropriate slot, which can be saved in control registers on setup, much like existing DRAM systems. The rest of the operation is completely independent of the underlying memory technology, enabling the support of heterogeneous systems.

5.3 Impact and Results

We evaluate the novel access protocol by implementing it in the framework described in Section 4.3.

5.3.1 General Benefits

5.3.1.1 Supporting heterogeneity

It is likely that future memory systems will be a combination of different kinds of memory, including DRAM, PCM, STT-RAM, etc. [19, 20]. Each will have its own timing constraints; for instance, DRAM itself has as many as 18 parameters such as t_{FAW}, t_{RRD}, t_{WTR}, t_{RTP}, t_{WTW}, t_{RTRS}, etc., which have to be accounted for during scheduling. Each will also have its own maintenance requirements (wear-leveling, drift management [104], etc.) and specialized coding requirements [105]. All these factors significantly increase complexity. In the current paradigm, all of these will have to be hard coded into the memory controller, requiring explicit processor support every time a new memory technology is introduced, or even when new features are introduced. Putting all this functionality on the interface die allows different types of DIMMs to co-exist on the same memory channel, especially important when, for example, DRAM and PCM are used as different levels of the memory hierarchy.

5.3.1.2 Flexibility and interoperability

The current tight integration of device characteristics, interconnect characteristics, and memory controller operation severely restricts interoperability. The memory market today consists of scores of DIMM variants, with a range of speeds, protocols, reliability levels, buffering, and physical profiles, with little cross compatibility. This affects both users and vendors - users are faced with a bewildering array of purchase choices, and vendors have to deal with stock-keeping and compatibility validation issues for a large number of products. The use of a packet-based interface abstracts away these low-level details, and allows the DIMM and the processor to be more independent, with greater plug-and-play capabilities across different product lines, or even different memory technologies. It also opens the door to innovation and value-addition within the memory modules (in stark contrast to today's conventional DIMMs) while presenting a consistent interface to the rest of the system.

5.3.1.3 Creating new market models

In the current scheme of things, memory vendors are unable to show substantial differentiation since they are constrained by strict standards and require processor-side support for each new feature. Decoupling processor and memory design via an abstracted interface helps address this problem. The ability to add value will spur more innovation (processing-in-memory, nanostores, row buffer caches, etc.), and perhaps remove commercial hurdles in realizing technically sound ideas. By using the proposed 3D chip stack, much of this innovation can be localized to the interface die and the high-density memory arrays can be left untouched. Memory vendors can also combine their generic memory chips with different interface dies to create a variety of products.

5.3.2 Address/Command Bus Energy Savings

In current low-level DDR interfaces with a capacity of 64 GB and 64 byte cache lines, each transaction consists of 36 address bits, 6 command bits (RAS and CAS; we assume an implicit precharge), and 512 data bits, for a total of 554 bits. Aggressive power-down schemes have been shown to be both essential [7], and easily achievable [22] in large-scale systems to keep idle power in check. With a large number of independent banks/arrays (as in our target system), potentially *every* request would therefore also involve a *Wake-up*, and a *Sleep* command, adding a further 58 bits [1]. Refresh, scrubbing, or other maintenance commands also add traffic on the bus in conventional systems, but these are harder to quantify. Every request would therefore transmit 600+ bits on the channel. A self-managing packet-based system such as ours, on the other hand, would only need to transmit the address and data, totalling to about 550 bits. This translates to a nontrivial energy saving of about 10% in the bit transport energy on every transaction.

5.3.3 Memory Controller Complexity

Memory controller complexity is expected to grow substantially as the number of banks increases and heterogeneity becomes more common. In our proposed model, the memory controller simply buffers requests, tracks the next available data slot, and issues the request accordingly. Thus, the logic that handles timing constraints and maintenance operations can be taken out of the memory controller, saving valuable processor die area. This is significant since future multicore processors are expected to have multiple memory controllers, growing as the square root of the number of processing clusters [106].

The stack controllers on the interface dies in the memory must now implement this logic, introducing some overhead due to replication in multistack channels. However, the interface die is already being employed to simplify photonic integration and has sparse functionality on it, providing enough dead silicon to introduce some logic functionality without severely impacting area or cost.

5.3.4 Protocol Performance Characteristics

The new interface's lack of global system knowledge may sometimes lead to suboptimal scheduling decisions, resulting in a minor negative performance impact compared to a traditional low-level DDRx[2] standard interface. It is speculative and relies on a coarse grained approach to resource management, with a simple retry

[1] Assuming a power-down granularity of 1024-bytes, the total capacity consists of 64M power-down "domains", which need 26 address bits. Adding 3 command-bits, and multiplying by 2 for "sleep" and "wake-up", we arrive at 58-bits. This can vary based on sleep granularity.

[2] DDR here refers to the general family of conventional, low-level, JEDEC-based memory interfaces, and not necessarily a particular Dual Data Rate signalling standard.

mechanism as a safety net. Mis-scheduled requests may also suffer an added latency penalty beyond that required for resources to become free. For example, a request that could not be serviced in Slot-1, but could potentially have been serviced say 2 cycles later, is now made to wait until Slot-2. These effects may also have a combined impact on the bandwidth. The DDRx system on the other hand is oracular in nature and the memory controller is fully aware of the entire memory system. However, our results show that even with these disadvantages, the simple packet-based interface reaches performance levels very close to the oracular DDRx interface. Even under peak load, less than 5% of requests use Slot-2, and only about 0.5% need to back-off and retry, ensuring that the overall performance loss is small.

Figure 5.2 compares the packet interface and the oracular DDRx interface under varying load, for a high-performance system such as the one described in Chapter 4. There is practically no difference in achieved bandwidth (the line graphs show this as a fraction of total provisioned bandwidth), and even in the heavily loaded case, there is less than 0.01% degradation. The latency characteristics are dependent on the load.

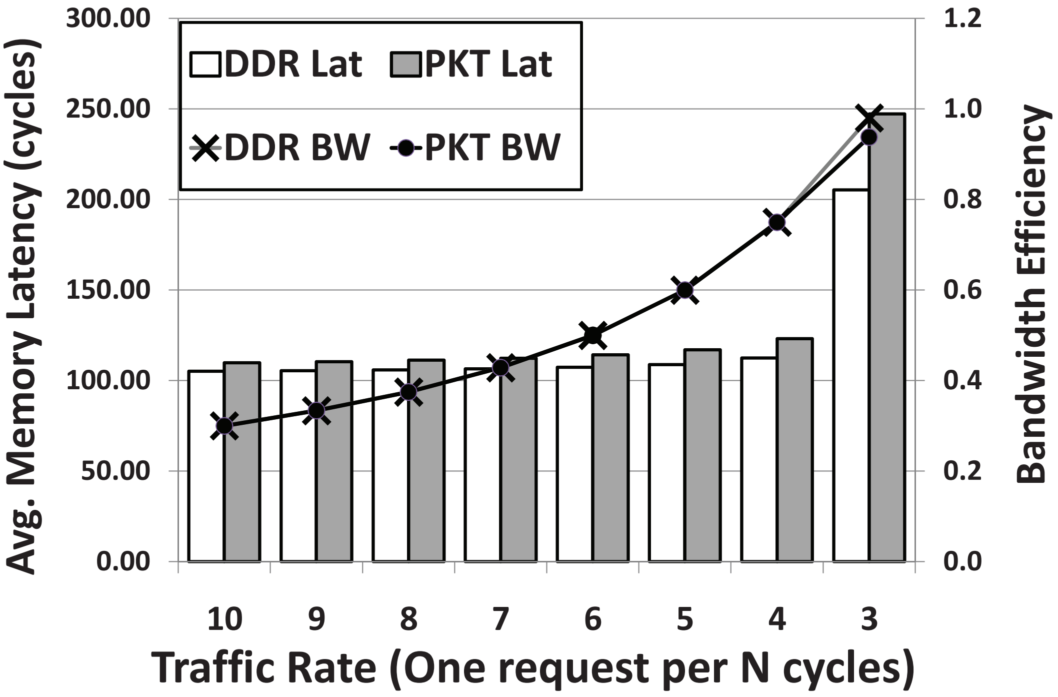


Figure 5.2. Performance under synthetic traffic stress test

At a traffic rate of 1 in 10 cycles, the latency penalty is just over 4%, rising to about 9% at a peak load of 1 in 4 cycles. With lower traffic, there are fewer possibilities of the conflicts that raise access latency in the packet-based system.

Both latency and bandwidth trends with real benchmarks are very similar to those with synthetic traffic, as shown in Figure 5.3. Since these applications have relatively low bandwidth requirements, we show results for a scaled down channel with a provisioned bandwidth of 10 GB/s (an 8-bit wide bus), and even this is not fully utilized.

The impact of changing the channel organization (number of stacks, number of dies per stack, etc.) is primarily due to the change in bank count, which determines how frequent conflicts are. Figure 5.4 shows the impact of increasing the number of banks per DRAM chip in the system. The bar graph shows the percentage increase in latency moving from an oracular DDRx protocol to the unscheduled packet-based protocol. As expected, increasing bank count reduces the negative performance impact. At peak load, the latency impact reduces from 9% with 16 banks to just 4.5% with 32 banks. At low loads, increasing bank count bridges the gap between the packet-based and

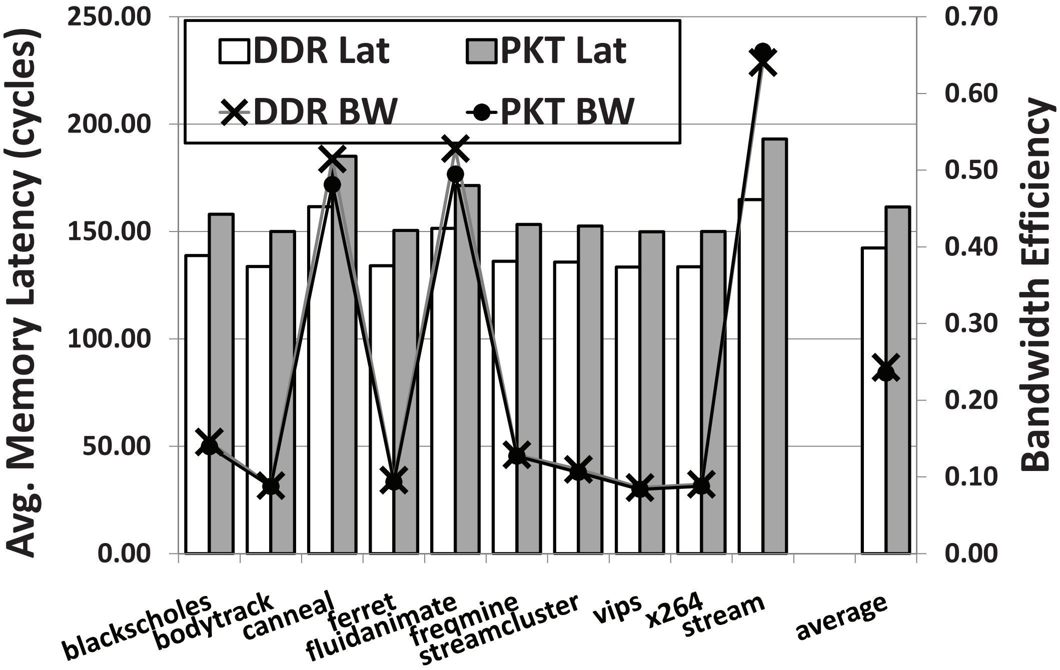


Figure 5.3. Performance of benchmark workloads, under reduced bandwidth

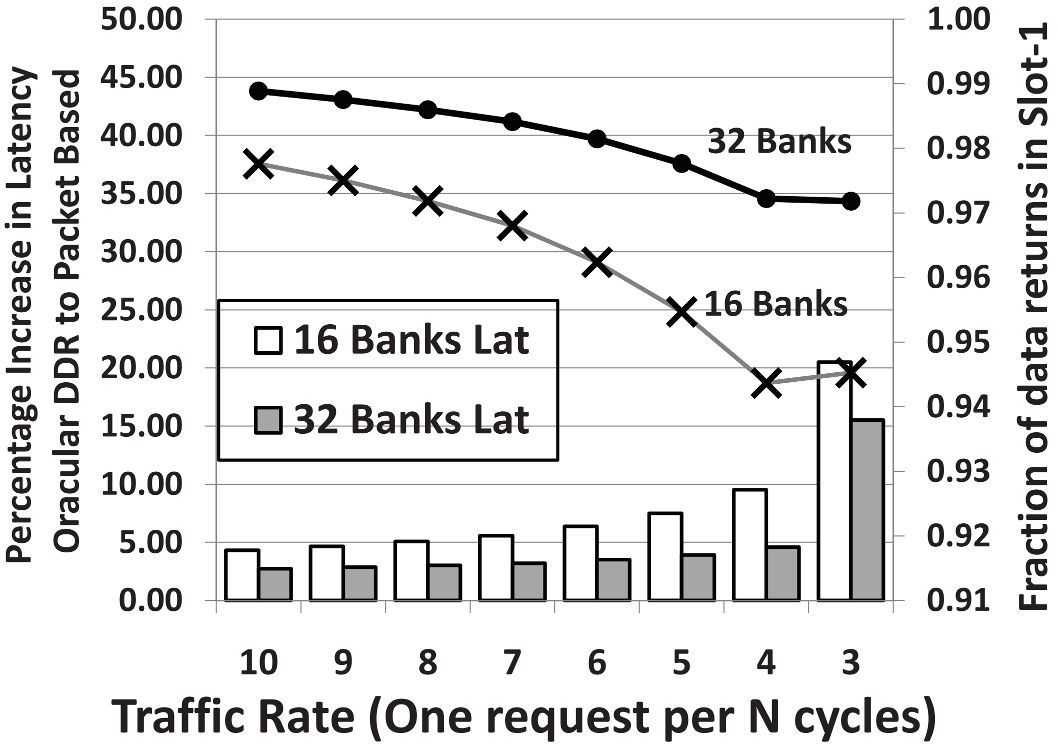


Figure 5.4. Impact of varying bank count

oracular implementations from 4.3% to just 2.7%. The line graphs correspondingly show the fraction of accesses that only use ‘Slot-1’ in the packet-protocol, *i.e.,* cases where there was no resource conflict. This metric improves from about 94% to 97% as the bank count is increased, even at peak load. Current DRAM parts already have 8 banks, and it is very likely that this will rise to 16 or 32 in the future, making the performance losses negligible. Another sensitivity analysis we carried out was to artificially make the traffic extra-bursty. While both DDR and the packet-based protocol suffered latency increases, the performance difference between the two only strayed marginally from the standard Bernoulli injection case.

5.4 Summary of Contributions

The primary limitation to scalability of the memory access mechanism is the low-level control paradigm of the aging DDR family interface, with the memory controller micromanaging every aspect of the system. We argue that the memory modules need to be made more autonomous, with low-level device management being handled locally, and the memory controller only performing essential tasks such as global scheduling, QoS enforcement, etc. We propose a novel access protocol where the processor-side memory controller simply issues a data request and speculatively reserves a slot on the bus for the data return, while by design being unaware of minutiae such as per-bank state. A low overhead dual-slot scheme handles relatively uncommon cases such as bank conflicts or refresh conflicts. Such a memory access system provides several benefits. First, it increases flexibility and interoperability by breaking the tight integration of device characteristics, interconnect characteristics, and the memory controller, enabling plug-and-play access by various kinds of DIMMs. Second, it allows the design of heterogeneous memory systems, carrying both DRAM and other kinds of nonvolatile memory on the same channel. Third, it reduces traffic on the address/command bus, relieving some pressure there to improve performance, while simultaneously saving approximately 10% in bit-transport energy per cache line request (address + commands + data). Finally, it opens the door to innovation and value-addition in the memory while presenting a consistent interface to the rest of the system.

CHAPTER 6

EFFICIENT MEMORY RELIABILITY MECHANISMS

6.1 Why We Need A Different Approach to Reliability

High-availability servers are expected to provide at least chipkill-level reliability – the ability to withstand the failure of an entire DRAM chip. Current commercial chipkill-level reliability mechanisms are based on conventional Reed-Solomon symbol-based ECC codes. As we described in detail in Chapter 2, they impose various restrictions on system configuration, and suffer from several problems. They activate a large number of chips on every memory access – this increases energy consumption due to severe activation overfetch, and reduces performance due to the reduction in rank-level parallelism. Additionally, they increase access granularity, resulting in forced prefetching and wasted memory bus bandwidth in the absence of sufficient access locality. They also restrict systems to use narrow-I/O x4 devices, which are known to be less energy-efficient than wide-I/O x8 DRAM devices. Finally, they increase circuit complexity due to the use of Galois-field arithmetic. These effects are likely to be exacerbated by various trends in memory system design such as increasing burst length, wider I/O DRAM chips, increasing contribution of memory to system-level power/performance, larger memory capacities, greater need for parallelism, and less reliable hardware. There is a need to fundamentally rethink memory error protection in a manner that is cognizant of these trends, and that improves efficiency while effectively addressing various DRAM failure modes.

Some recent papers [18, 22] have proposed solutions that are efficient, but require substantial modifications to some combination of the memory chips, channel, caches, OS, etc. Given the memory industry's hard constraints with respect to commodity parts in servers, however, it is essential to stay fully standards-compliant to even be considered for commercial adoption. We allow ourselves no leeway to modify the design of DRAM chips, DIMMs, or the JEDEC protocol in any way, including burst length, access granularity, etc. Similarly, the fault-tolerance mechanism should be completely transparent to the cache hierarchy, OS, and applications in order to be implementation friendly. The only permitted changes are in the memory controller and system firmware. Also, our interactions with the server industry indicate that memory capacity is itself affordable, as long as commodity components are used. Some of it can be traded off for RAS benefits, rather than attempt to change standards or modify a multitude of system components.

With these requirements and constraints in mind, in this chapter, we present LOT-ECC, a novel chipkill-level reliability mechanism that employs simple localized and multitiered checksum and parity fields with separate error detection and error correction functionality that significantly reduces energy consumption and average memory access latency compared to conventional chipkill designs, at the cost of a small increase in storage overhead.

6.2 Proposed Architecture: LOT-ECC

LOT-ECC takes a novel, fundamentally different approach to reliability. It splits protection into two distinct layers – one to detect errors and isolate their location, and the other to reconstruct the erroneous data. It maps data and the two layers of error protection such that they all fall in the same row-buffer, guaranteeing efficient row-buffer hits during reliability operations.

6.2.1 Layer 1 - Local Error Detection (LED)

The first layer of protection afforded by LOT-ECC is *local error detection (LED)*. The function of this code is to perform an immediate check following every read operation to verify data fidelity. Additionally, it needs to identify the exact location of the failure, at a chip-granularity within a rank. To ensure such chip-level detection (required for chipkill), the LED information itself needs to be maintained at the chip level – associated not with each cache line as a whole (as in symbol-based ECC codes), but with every cache line “segment”, the fraction of the line present in a single chip in the rank. In Figure 6.1, for example, cache line A is divided into segments A_0 through A_8, with the associated local error detection codes L_{A0} through L_{A8}.

6.2.1.1 Data layout

Consider a standard rank with 9 x8 DRAMs, and a burst length of 8, as shown in Figure 6.1. Instead of treating this as eight data chips and one ECC chip, we propose storing both data and LED information on *all nine chips*. Since we need 512 data bits (one cache line) in total, each chip will have to provide 57 bits towards the cache line. An x8 chip with a burst length of 8 supplies 64 bits per access, which are interpreted as 57 bits of data (A_0 in Figure 6.1, for example), and 7 bits of *LED* information for those 57 bits (L_{A0}). The physical data mapping policy ensures that LED bits and the data segments they protect are located on the same chip.

First, such a design slightly reduces the storage available to build an error detection code, from 12.5% in conventional systems to 12.3%. However, we expect the impact of this reduction to be close to zero. Second, one bit remains unused for every 576 bits, since 57*9 is 513, and we only need 512 to store the cache line. We utilize this *surplus bit* as part of the second layer of protection, details of which follow in Section 6.2.2.

6.2.1.2 Impact on memory reads and writes

There are no performance penalties on either reads or writes due to the LED code. Every cache line access also reads/writes its corresponding LED information. Since the LED is "self-contained", *i.e.*, it is constructed from bits belonging to exactly one cache line, no read-before-write is required – all bits required to build the code are already at the memory controller before a write.

6.2.1.3 LED checksum design

The choice of error detection code for the LED depends primarily on the expected failure modes. For our design, we believe that a simple one's complement addition checksum [107] will suffice. Further details on the operation of the checksum and its ability to handle common DRAM failure modes follow in Section 6.2.5.

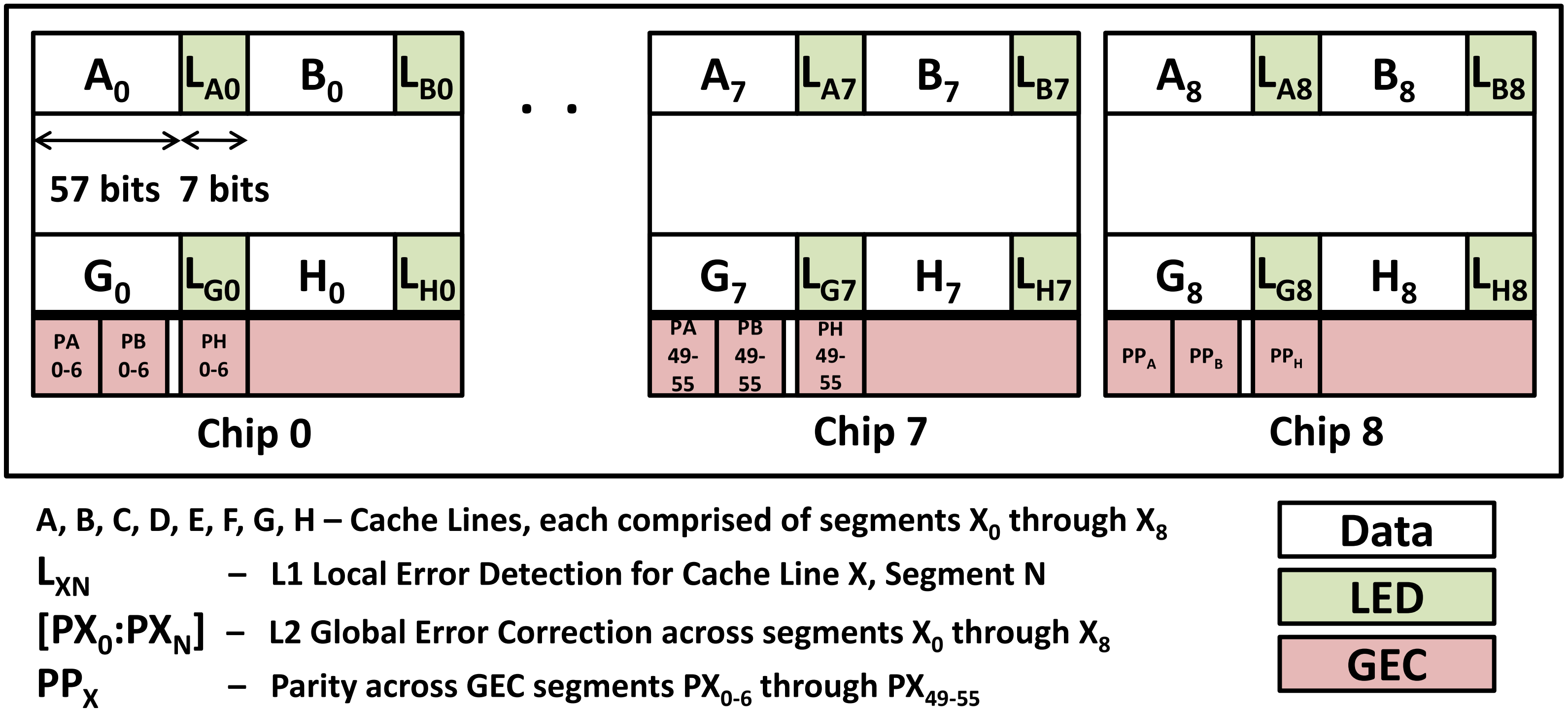

A0
LA0
B0
LB0
57 bits
7 bits
G0
LG0
H0
LH0
PA 0-6
PB 0-6
PH 0-6
Chip 0
A7
LA7
B7
LB7
G7
LG7
H7
LH7
PA 49-55
PB 49-55
PH 49-55
Chip 7
A8
LA8
B8
LB8
G8
LG8
H8
LH8
PPA
PPB
PPH
Chip 8
A, B, C, D, E, F, G, H – Cache Lines, each comprised of segments X0 through X8
LXN – L1 Local Error Detection for Cache Line X, Segment N
[PX0:PXN] – L2 Global Error Correction across segments X0 through X8
PPX – Parity across GEC segments PX0-6 through PX49-55
Data
LED
GEC

Figure 6.1. LOT-ECC shown with a single rank of nine x8 DRAM chips

6.2.2 Layer 2 - Global Error Correction (GEC)

The function of the Layer 2 Global Error Correction code is to aid in the recovery of lost data once the Layer 1 code detects an error and indicates its location. The Layer 2 GEC is itself decomposed into three tiers. The global error correction code is a 57-bit entity that is a column-wise XOR parity of nine cache line segments, each a 57-bit field from the data region. For cache line A, for example, its GEC parity PA is a XOR of data segments A_0, A_1, .., A_8. Data reconstruction from the GEC code is trivial (a simple XOR of the error-free segments and the GEC code) since the LED has already flagged the erroneous chip. Since there is not an additional dedicated ECC chip (we used up the one provided to store data + LED), the GEC code has to be stored in data memory itself. The memory is simply made to appear smaller than it physically is (by 12.5%) to the operating system via firmware modifications. The memory controller is also aware of this change, and maps data accordingly.

6.2.2.1 Data layout

In line with our objective to provide strong fault-tolerance of 1 dead chip in 9, and to minimize the number of chips touched on each access, we choose to place the GEC code in the same rank as its corresponding cache line, unlike prior proposals [18].

We set apart a specially-reserved region (shaded red in Figure 6.1) in each of the 9 chips in the rank for this purpose. This is a subset of cache lines in every DRAM page (row), although it is shown as a distinct set of rows in Figure 6.1 for clarity. This co-location ensures that any reads or writes to the GEC information will be guaranteed to produce a row-buffer hit when made in conjunction with the read or write to the actual data cache line, thus reducing its performance impact.

Figure 6.2 shows the way the GEC information is laid out in the reserved region, for an example cache line A. Similar to the data bits, the 57-bit parity PA is itself distributed among all 9 chips. The first seven bits of the PA field (PA_{0-6}) are stored in the first chip, the next seven bits (PA_{7-13}) are stored in the second chip, and so on. Bits PA_{49-55} are stored on the eighth chip. The last bit, PA_{56} is stored on the ninth chip, in the *surplus bit* borrowed from the Data+LED region (see Section 6.2.1).

The failure of a chip also results in the loss of the corresponding bits in the GEC region. The GEC code PA itself, therefore, is protected by an additional parity,

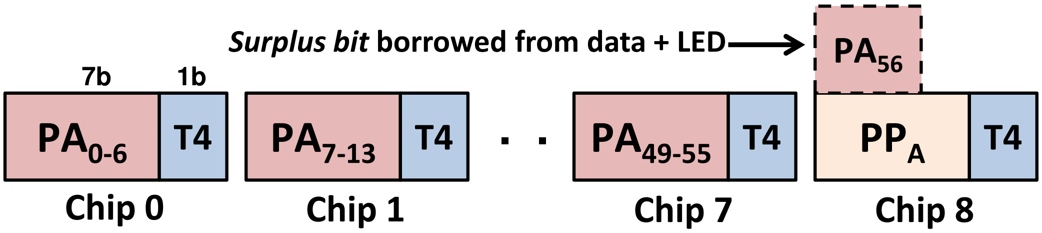


Figure 6.2. Data layout in the red-shaded GEC region

the third tier PP_A. PP_A is a 7-bit field, and is the XOR of eight other 7-bit fields, PA_{0-6}, PA_{7-13}, .., PA_{49-55}. The PP_A field is stored on the ninth chip. If an entire chip fails, the GEC is first recovered using this parity combined with uncorrupted GEC segments from the other chips (we know which chips are uncorrupted since the LED indicates the failed chip). The full GEC is then used to reconstruct the original data.

Next, consider the case where in addition to a fully failed chip, there is an error in a second chip – we still need to be able to detect, if not correct, such a failure under our fault model. If this second error is also a full-chip failure, it will be detected by the LED along with the initial data read, and flagged as an uncorrectable double-chip failure. On the other hand, if this second error occurs just in the GEC row of interest, it needs to be detected during the GEC phase.

For example, assume that the second chip has completely failed – we have now lost A_1, and PA_{7-13}. If, in addition, there is an error in the GEC region of the first chip, there is a possibility that one or more of the bits PA_{0-6} are corrupt. The reconstruction of lost bits PA_{7-13} from PP_A and PA_{0-6}, PA_{14-20}, PA_{21-27}, .., PA_{56} may itself be incorrect. To handle this problem, we use the remaining 9 bits (marked $T4$, for Tier-4, in Figure 6.2) to build an error *detection* code across GEC bits PA_0 through PA_{55}, and PP_A. Note that neither exact error location information nor correction capabilities are required at this stage since the reliability target is only to *detect* a second error, and not necessarily correct it. We can therefore build a code using various permutations of bits from the different chips to form each of the $T4$ bits. This should include multiple bits from the same chip, and bits from different columns across chips to maximize the probability of detection.

We will now work through examples of various failure possibilities, and illustrate LOT-ECC operation in each case. Again, consider a single cache line A. Recall that chips 0-7 (without loss of generality) contain 57 bits of data plus 7 bits of LED in the data region, and 7 bits of GEC parity plus 1 bit of T4 information in the GEC region. Chip-8 contains 56 bits of data plus 7 bits of LED plus one surplus bit in the data region, and 8 bits of parity (including the surplus bit borrowed from the data region) plus one bit of T4 information in the GEC region.

If one of the first eight chips, say the second chip, fails, 57 bits of data (A_1) are lost, in addition to GEC parity information PA_{7-13}. We first read A_0 - A_8, and the LED associated with A_1 (L_{A1}) indicates a chip error. We then read GEC segments PA_{0-6}, PA_{14-20}, PA_{21-27}, .., PA_{49-55}, PA_{56} and PP_A to recover the lost GEC bits PA_{7-13}, thereby reconstructing GEC parity PA. Combined with values A_0 and A_2 - A_7, data value A_1 can be reconstructed, thus recovering the entire original cache line. If, on the other hand, the ninth were to fail, only 56 bits of data are lost (A_8), in addition to PP_A, and the surplus bit PA_{56}. The lost 56 bits can be recovered simply from the 56 columns of parity stored in the first eight chips (PA_{0-55}), thus recovering the entire original cache line. The loss of surplus bit PA_{56} is immaterial. Across these cases, the fidelity of the GEC parity bits themselves is guaranteed by $T4$.

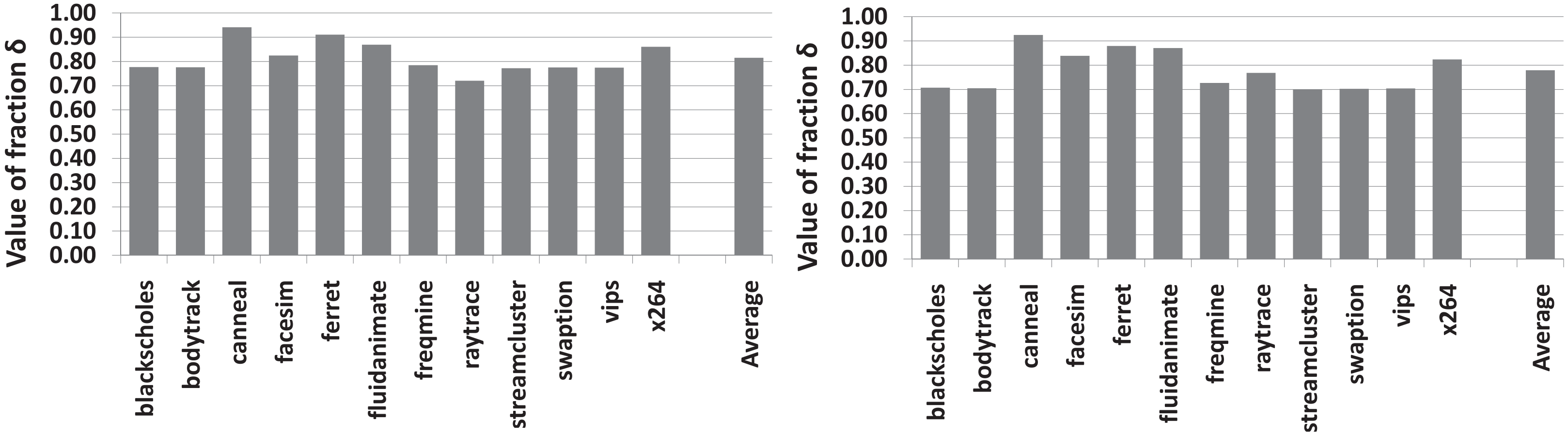
6.2.2.2 Impact on memory reads and writes

Read operations need not access GEC information unless an error is detected, which is a rare event. GEC therefore has no significant impact on reads. Write operations, on the other hand, need to update the GEC (which includes P_X, PP_X, and $T4$) when any data are modified. In a baseline implementation, each cache line write gets transformed into two writes – one to the data location (for a full 576 bits of data+LED+surplus bit) and another to its corresponding GEC location (72-bits). Although only 72 bits of GEC+T4 code need to be updated per write, we are forced by the DDR3 protocol to complete a burst of 8 per access (an entire 72-byte "cache line" size of data). We could thus potentially combine as many as 8 different GEC updates into a single write command, reducing some of its performance impact. This is low-overhead since writes are already buffered and streamed out intermittently from the memory controller, and additional logic can easily be implemented at this stage to coalesce as many GEC writes as possible. Performance impact is further minimized since the data mapping ensures that the GEC write is always a row-buffer hit once the data line is written. This trick is similar to Loh and Hill's optimization to store tag and data for a cache line in the same DRAM row in a 3D-stacked architecture to guarantee row-buffer hits [108]. Additionally, note that there is not a need for a read-before-write of the data cache lines themselves as in some earlier proposals [22], since all bits contributing to the GEC code are from a single cache line, already available at the controller. This further minimizes performance impact. If complete coalescing is not possible (based on the addresses being written to), data masking [109] can be employed to only write the appropriate bits into memory. Note that the complete burst of 8 has to performed nonetheless – some pieces of data are just masked out while actually writing to DRAM.

With all these considerations, every write is transformed into $1+\delta$ writes, for some fraction $\delta < 1$ dependent on the access characteristics of the application. Note that $\delta = 1$ in a noncoalesced baseline LOT-ECC implementation, and 0.125 in an oracular design since eight GEC words fit in a single "cache line", and could potentially be coalesced into a single write. The values of δ for the PARSEC [66] benchmark workloads are shown in Figure 6.3 (details on methodology follow in Section 6.4). Note that for the purposes of this chapter, we employ a very rudimentary coalescing scheme where the "window" into which we look to find suitable writes to coalesce is simply the set of writes that are sent out by default each time the bus is turned around for a "write burst". There is certainly a possibility of more involved schemes that buffer GEC writes for longer periods (even if the corresponding data writes have already been sent out) in an effort to increase the coalescing window, and further decrease the value of δ. We leave such optimizations to future work.

6.2.3 Storage Costs of LOT-ECC

For each 64-byte cache line in a rank with nine x8 chips, LOT-ECC requires the following ECC bits: (1) 63 bits of LED information, at 7 bits per chip, (2) 57 bits of GEC parity, spread across the nine chips, (3) 7 bits of third-level parity, PP_X,



(a) Xeon 7500 based system

(b) Xeon 5500 based system

Figure 6.3. Quantification of GEC write overheads after coalescing and (4) 9 bits of $T4$ protection,1 bit per chip. This adds up to a total of 136 bits, a storage overhead of 26.5%. Out of this 26.5%, 12.5% is provided by the 9th chip added on to standard ECC DIMMs, and the other 14% is stored in data memory in the GEC region.

6.2.4 Extending LOT-ECC to Wide-I/O DRAMs

Wider-I/O DRAM parts are favored over narrower DRAMs since they are typically more power efficient [18] and enable greater rank-level parallelism. To enable their widespread adoption, there is a need to develop efficient reliability mechanisms that work with such configurations. However, there are several unknowns regarding the use of x16 and x32 DRAMs in future commodity DIMMs – data bus width increases, the provisioning of additional chips for ECC support, and burst length, for example. We therefore present a brief discussion of LOT-ECC implementation under a few possible scenarios, and propose the novel concept of *heterogeneous DIMMs* to address some concerns with these options. For clarity, we only quantify overheads with x16 DRAMs; similar ideas can be extended to x32 DRAMs and beyond. The options are summarized in Table 6.1.

6.2.4.1 Option 1 - wide memory channels

If the natural progression of memory system design is to simply increase the channel width as wider-I/O DRAMs become common, LOT-ECC can be implemented with little modification. For instance, consider a rank of nine x16 DRAMs. The 128 bits supplied by an x16 DRAM in a burst of 8 would simply be interpreted as 114 data bits and 14 checksum LED bits, the same storage overhead as with x8 DRAMs. GEC operation remains unchanged. While there is necessarily an increase in access granularity and overfetch (independent of LOT-ECC), storage overhead remains constant at approximately 25% (LED + GEC).

6.2.4.2 Option 2 - increasing storage overhead

If access granularity is fixed at exactly one cache line (*i.e.*, a 64-bit bus), the minimum rank size with x16 chips is 5 chips (4 data plus 1 ECC). Each chip provides 128 bits per burst of 8, interpreted as 103 data bits (since 103 * 4 chips = 512-bit

Table 6.1. Implementing LOT-ECC with x16 DRAM parts

Design	Access Granularity (Cache lines)	Storage Overhead	Customization Required
Option 1	2	25.0%	None
Option 2	1	50.0%	None
Option 3	10	37.5%	None
Heterogeneous DIMMs	1	37.5%	Commodity x16 & x8 DRAMs on a custom DIMM

cache line). This leaves 25 bits per chip to store the LED code, which provides very strong error protection, but is likely overkill and wasteful of storage area (the overhead is 24%). GEC overhead increases as well, since the global parity is a 103-bit entity computed over four 103-bit data segments, a storage overhead of 25%. The total overhead is approximately 50%.

6.2.4.3 Option 3 - optimizing storage overhead

If storage overhead is an important consideration, it can be fixed at about 12.5%, paying for it through an increase in access granularity. With x16 chips and a 5-chip rank, for example, 9 reads can be issued consecutively, reading out a total of 80 bits per cycle * burst of 8 cycles * 9 accesses = 5,760 bits. This results in a very large access granularity of 10 cache lines (5120 bits) plus their LED codes, with a storage overhead of 12.5%. The GEC overhead remains approximately 25% (1 in 4 chips), similar to Option 2, for an overall ECC storage overhead of 37.5%.

6.2.4.4 Heterogeneous DRAMs within a rank

If neither access granularity nor storage overhead can be compromised, but there is freedom to implement a custom DIMM, we propose a novel third option – the use of heterogeneous DRAMs within a single DIMM rank. In this case, minimum access granularity can be maintained while still retaining a 12.5% storage overhead. With x16 parts, for instance, a minimum-sized rank would be four x16 DRAMs plus one x8 DRAM (note that the DRAMs are still commodity, just not the DIMM), providing a DIMM width of 72 bits. With a burst length of 8, each x16 DRAM supplies 128 bits and the x8 DRAM supplies 64 bits. These should be interpreted as (114 data + 14 LED) and (56 data + 8 LED), respectively. There is no change to GEC overhead or operation.

6.2.4.5 Conclusion

It is clear that there are several knobs available to turn – the storage overhead, the importance of access granularity (typically a function of access locality in the workload), the willingness to build heterogeneous DIMMs – as wide I/O parts such as x16 or x32 become mainstream due to their reduced power consumption. LOT-ECC is flexible enough to be effective in designs with varying combinations of these knobs.

6.2.5 DRAM Failure Modes and Checksum Operation

The target of the LED code (Section 6.2.1) is to *detect* errors in 57 bits of data through the use of a 7-bit checksum. The 57 bits of data are divided into 8 blocks of 7 bits each. The last bit is padded with 6 zeros to make it a 7-bit value. A 7-bit checksum is produced by performing an integer one's complement addition of these nine blocks. The carry-out information of the MSB is preserved via being wrapped around and added back to the LSB, improving error detection capability.

Prior work has shown that DRAM errors occur in five major ways – single-bit error, double-bit error, row-failure, column-failure, and full chip-failure [110, 111, 112, 113]. We now consider each of these in turn, and show how the proposed LED checksum handles error detection.

6.2.5.1 Single-bit error

This is among the most common failure modes in DRAM, and can be either due to a soft-error or a hard-error. Typical occurrence probability is of the order of 12.6 FIT[1] due to hard errors and several orders of magnitude higher for soft-errors [110, 113]. Any checksum is guaranteed to capture this kind of error; it is therefore not a cause for concern.

6.2.5.2 Double-bit error

These are defects that span two *nearest neighbor* memory cells and are *not* the coincidence of two independent single bit errors [110]. The probability of such an occurrence is extremely low, estimated at just 0.7 FIT per DRAM, two orders of magnitude lower than single-bit errors. In any case, adjacent bits will fall into adjacent columns for addition during the checksum generation process, only flipping a single bit in each column. This will alter the checksum value, causing the error to be detected.

6.2.5.3 Row-failure

These are caused by a failure of one of the chip's row drivers, causing all cells in one row of the affected chip to fail. We employ bit inversion in the checksum computation so that stuck-at faults are detected [22]. With this in place, the checksum captures an entire row incorrectly being either all 0 or all 1. The row-error probability has been estimated to be about 6.3 FIT [110].

6.2.5.4 Column-failure

The most common cause of this kind of error is a single data-pin getting stuck at 0 or 1. It might also be caused by a failure of one of the column senseamps or column decoders, with similar effects as a failed pin. The error probability for this kind of error is about 4.1 FIT [110]. With the LED scheme we described, since each block is 7 bits, and each pin outputs 8 bits per access, there is a "wrap-around" that occurs naturally. This means that each of the 8 potentially erroneous bits coming from the failed pin falls into a distinct column (with the exception of the last bit). Column-failures, therefore, are easily captured by the LED checksum.

[1] 1 FIT, or Failure In Time, is when 1 failure is likely in 1 billion hours of operation.

6.2.5.5 Chip-failure

This is the second most common cause of failure, with an estimated FIT of 13.7 per DRAM [110]. For a given cache line, handling a chip error is conceptually equivalent to handling a row failure, and the same arguments with regard to error detection hold good. The difference lies primarily in posterror service operation. With a row error, there is a possibility of remapping the data from the failed row (upon reconstruction) into one of several *spare rows* typically provisioned on DRAM chips, if the memory controller has this functionality (this is often called “page-remapping”). With an entire chip failure, the DIMM is immediately flagged for replacement.

6.2.5.6 Multiple random bit errors

Addition-based checksums are susceptible to silent data corruption if an even number of bits in a given checksum-column flip. In our LED checksum, Column 0, for example, consists of bits 0, 7, 14, 21, 28, 35, 42, 49, and 57 from a single row. Since there is 6-bit gap between the closest 2 bits, it is highly unlikely that the same failure “cause” (other than a row or column failure), such as a soft-error strike, can result in such bit flips. This means that the two have to be *independent* single-bit events. Since each error is already rare in general, the probability of a *combination* of errors affecting the same data word in the same locality is extremely low [111]. To quantify, consider the 12.6 FIT number for a single-bit hard error. If a single DRAM has 16 M cache line segments, the probability of an error in a given segment during a billion hour period is 12.6 / 16 M, which is 7.51E-7 – this is the *segment FIT*. The probability of two independent errors affecting the same segment is therefore 5.64E-13 per billion hours. With each segment being 64 bits, the probability that the two erroneous bits are among the 9 bits 0, 7, 14, 21, 28, 35, 42, 49, and 57 is (9C2)/(64C2). Since errors can occur in any of the 7 checksum-columns, we multiply this value by 7, giving 0.125. The overall probability of a 2 bits flipping in a way that is undetected is, therefore, 5.64E-13 * 0.125, which is just 7E-14 FIT, a negligibly small risk. Even if the FIT of a single-bit error is orders of magnitude larger than the assumed 12.6, the likelihood of silent data corruption would be extremely small. The odds of 4 or 6 or 8 independent errors all affecting the same row are even smaller.

This also holds true for other combinations such as partial row failures or multiple column decoder/senseamp failures on the same data word. We believe that these highly improbable events have a negligible impact on overall reliability guarantees.

6.2.5.7 Discussion

Existing commercial chipkill mechanisms formally guarantee catching *any and all* possible combinations of bit flips within a chip. However, field studies indicate that DRAM errors typically fall under a very small number of failure modes with well-understood root causes. Current mechanisms therefore pay very significant power, performance, and flexibility costs just to protect against obscure bit-flip combinations that occur very rarely, if ever.

In contrast, the philosophy behind LOT-ECC is to be cognizant of the fundamental sources of error and their relative probabilities, and tailor protection accordingly. It captures all of the commonly occurring failure modes, while simultaneously providing very substantial benefits in terms of power, performance, flexibility, and complexity. Additionally, LOT-ECC enables recovery from even the failure of 1 DRAM chip out of every 9 in the system, compared to 1 out of 36 in current commercial mechanisms.

Given the significant power/performance overheads of fault-tolerance, the burden of memory reliability will likely not be foisted solely upon ultra-robust ECC codes, especially considering trends such as rising burst lengths, wider chip I/Os, and increasing system capacities. Current high-availability servers already implement RAS features such as patrol scrubbing [114], spare-row mapping, bit steering, memory page retire [110], etc. These will likely handle a majority of errors before they can worsen into undetectable or uncorrectable errors. With proactive scrubbing, for example, the chances are extremely remote that three bit flips in specific locations in a single row will occur timed in a way so as to stay undetected for long enough that a fourth corruption can occur and cause a silent data corruption with LOT-ECC's LED checksum. We believe that such multipronged approaches will be far more beneficial from a full-system perspective than brute force catch-all techniques that attempt to single-handedly guarantee high levels of reliability.

6.3 Related Work

6.3.1 Virtualized ECC

Virtualized ECC (VECC) [18] is a recent academic proposal which attempts to provide chipkill-level reliability while exploiting the benefits of x8 and x16 DRAMs, without requiring nonstandard DIMMs. It separates error detection and error correction into two tiers, similar also to other prior work [115, 116, 22]. Further, it proposes storing some ECC information (the second tier T2EC, responsible for data correction if an error is detected) in data memory, similar to an earlier proposal for last level caches [115]. However, VECC still suffers from some significant drawbacks. It continues to use conventional symbol-based ECC codes, inheriting all of their constraints with respect to DIMM/rank organization and access granularity. First, it requires 144-bit channels, which was not a problem in the original paper since it evaluated a DDR2 channel with a burst length of 4 – every access is only 144*4 bits, a single 64-byte cache line. However, most servers today use DDR3 channels, with a minimum burst length of 8, forcing prefetch. Half-burst accesses (length = 4) are possible in DDR3, but this simply masks out the last four bursts – no transfer occurs, and half the bandwidth is wasted. Second, 18 x8 chips are made busy on each access (twice the minimum required), increasing overfetch and reducing rank-level/bank-level parallelism. Third, these problems are exacerbated for memory write operations. Specifically, the data mapping adopted by VECC leaves open the possibility of the T2EC being placed in a failed DRAM chip if a single rank is used. It therefore forces the T2EC code to be placed in a different rank (also 18 x8 chips), thus raising the number of chips touched when writing to memory to 36. It also means that we can only tolerate failure in at most 1 in 36 DRAM chips. Finally, VECC requires modifications to several components of the system, including the operating system, memory management unit, caches, etc., making it difficult to implement.

6.3.2 Other Related Work

Memory reliability has received increased attention from the architecture community in recent years. Schroeder et al. studied DRAM failures in large-scale datacenters at Google, and highlighted their importance going forward [17]. The Rochester Memory Hardware Error Project [111] characterized not only failures at a platform level, but right down to individual bits. They conclude that the most common failure modes are single-bit errors, row errors, column errors, and entire chip failures. Papers on novel memory architectures such as Mini-Rank [33] and MC-DIMM [82] have included discussions on their impact on ECC and/or Chipkill, but have not proposed any specific fundamental changes in this regard. The MC-DIMM paper also mentions the use of heterogeneity within a DIMM to provide ECC. A qualitative comparison of LOT-ECC with commercial and academic implementations is presented in Table 6.2, and a quantitative discussion follows in the subsequent section.

6.4 Benefits and Results

6.4.1 Methodology

6.4.1.1 Performance studies

To study the performance characteristics of our designs, we use a detailed in-house DRAM simulator. We accurately model refresh overheads, address/command bus, data bus, bank/rank/channel contention, and the memory controller queue. We also model a separate write-queue that holds write operations (and the associated GEC writes) as they come in to the memory controller, and issues them in bursts to amortize bus-turnaround overheads. We use close-page row-buffer management with a simple FCFS policy, since the access pattern has little impact on the operation of LOT-ECC. The one exception is that for write operations, the data line and GEC lines (coalesced or otherwise) are guaranteed to be co-located in a single row buffer, making an open-page policy beneficial. Close-page policies are the norm today for industry standard servers from all major vendors in an effort to improve performance in typical workloads with relatively low locality [87, 88, 89]. We adopt the address mapping policy from Jacob et al. [38]. DRAM timing parameters were obtained from Micron DDR3 datasheets. We employ a synthetic traffic generator with a tunable rate of traffic injection, and a tunable fraction of write operations for a majority of our experiments. We use the PARSEC [66] benchmarks to evaluate the GEC coalescing scheme.

Table 6.2. Commercial, academic, and proposed chipkill implementations; burst length of 8

Design	**Bus-width**	**Granularity (Cache lines)**	**Storage overhead**	**Problems**
SSC-DSD x4, 4-check symbol code, commercially used	128b data + 16b ECC	2	12.5%	Causes overfetch, forces prefetching, reduces rank-level parallelism, uses GF arithmetic, restricted to x4
VECC x8, 3-check symbol code, 1 symbol virtualized	128b data + 16b ECC	2Rd/4Wr	18.75%	Significant overfetch and prefetching, even worse for writes, performance impact due to extra writes to virtualized symbol, reduced parallelism, GF arithmetic, extensive changes required to OS, TLBs, caches
LOT-ECC x8, checksum+parity	64b data + 8b ECC	1	26.5%	Increased storage overhead, small performance impact due to extra writes to global parity

6.4.1.2 Energy computation

To accurately compute memory power consumption, we directly use Micron's DDR3 power calculator spreadsheet [10]. The spreadsheets require inputs regarding bus utilization, bank utilization, etc., which we obtained from our performance simulation. The calculator accounts for activation power, read/write power, termination power, and background power. It also incorporates support for low-power modes, which we assume are oracularly applied whenever all banks are idle.

6.4.1.3 Target system configurations

We evaluate our designs on memory systems based on two contemporary high-end server processors from Intel. The same basic controller/channel/DIMM organization and reliability mechanisms are used by all major server vendors. First, we consider Xeon 7500-based systems [117, 118, 119], with a typical configuration shown in Table 6.3. One design characteristic that had a significant impact on the design of reliable memory is the mandatory implementation of a *Lockstep Mode* in all Xeon 7500 based servers. This gangs two x72 ECC-DIMMs and channels together to form a 144 bit bus, essential to provide chipkill-level reliability, due to the inherent nature of the ECC codes used. This results either in wasted bandwidth (if you only read 72 bytes per 144 byte burst of 8, and mask out the other half), or forced prefetching. With LOT-ECC, this requirement can be relaxed, allowing substantial power and performance gains.

Second, we consider Xeon 5500-based systems [120, 121], with a typical configuration as shown in Table 6.3. Since these processors support an odd-number of channels (three, in this case), the effect of lockstep mode is even more pronounced. While the processor does allow operation in nonlockstep mode for noncritical applications, chipkill still requires this mode to be turned on, where one of the three channels is simply left empty, and the other two are ganged together. This results in a significant performance hit due to the reduction of peak bandwidth. Note that design decisions leading to an odd number of channels are a result of complex interplay between pin availability, expected bandwidth demand based on target workloads and processor horsepower, re-use of designs for varying market segments (desktops, laptops, and servers), etc. It is therefore not impossible to believe that such designs may continue

Table 6.3. Main memory configuration

Parameter	Value
Protocol	DDR3
Channels	8
DIMMs/channel	2
Ranks/DIMM	2 for x4; 4 for x8
Banks/Rank	8
Capacity	128 GB

Xeon 7500-based systems

Parameter	Value
Protocol	DDR3
Channels	3
DIMMs/channel	3
Ranks/DIMM	2 for x4; 4 for x8
Banks/Rank	8
Capacity	72 GB

Xeon 5500-based systems

to exist in future commercial processors, despite the large impact on power and performance during high-reliability operation.

6.4.1.4 Reliability models evaluated

Our baseline model is a modern commercial Single Symbol Correct Double Symbol Detect (SSC-DSD) design, as described in Section 3.2. This is currently supported only with x4 DRAMs. Additionally, it requires the use of lockstep-mode operation. Our second baseline is VECC [18], which enables the use of x8 DRAMs in lockstep mode, while suffering a small write-penalty. We obtain the T2EC cache miss rate, *i.e.,* the fraction of writes that require an additional "redundant information write", from the original paper, and include it in our simulator – the VECC model is therefore an optimized baseline, accounting for T2EC caching. We also optimistically assume that all the hardware/software support mechanisms required to make VECC work happen oracularly with no performance or power impact. We compare these baseline designs against our proposed LOT-ECC design, which allows us to upgrade to x8 DIMMs, *without* the use of lockstep mode. In this mode, *every* write is transformed into two writes – one data and one GEC. We then show the benefits of coalescing GEC writes to reduce their performance impact. Finally, we show an oracular design with perfect coalescing, where only one in eight writes suffers from the overhead of an additional GEC write.

6.4.2 Power Savings

LOT-ECC provides substantial power savings compared to traditional chipkill mechanisms, through a reduction of both dynamic and static power.

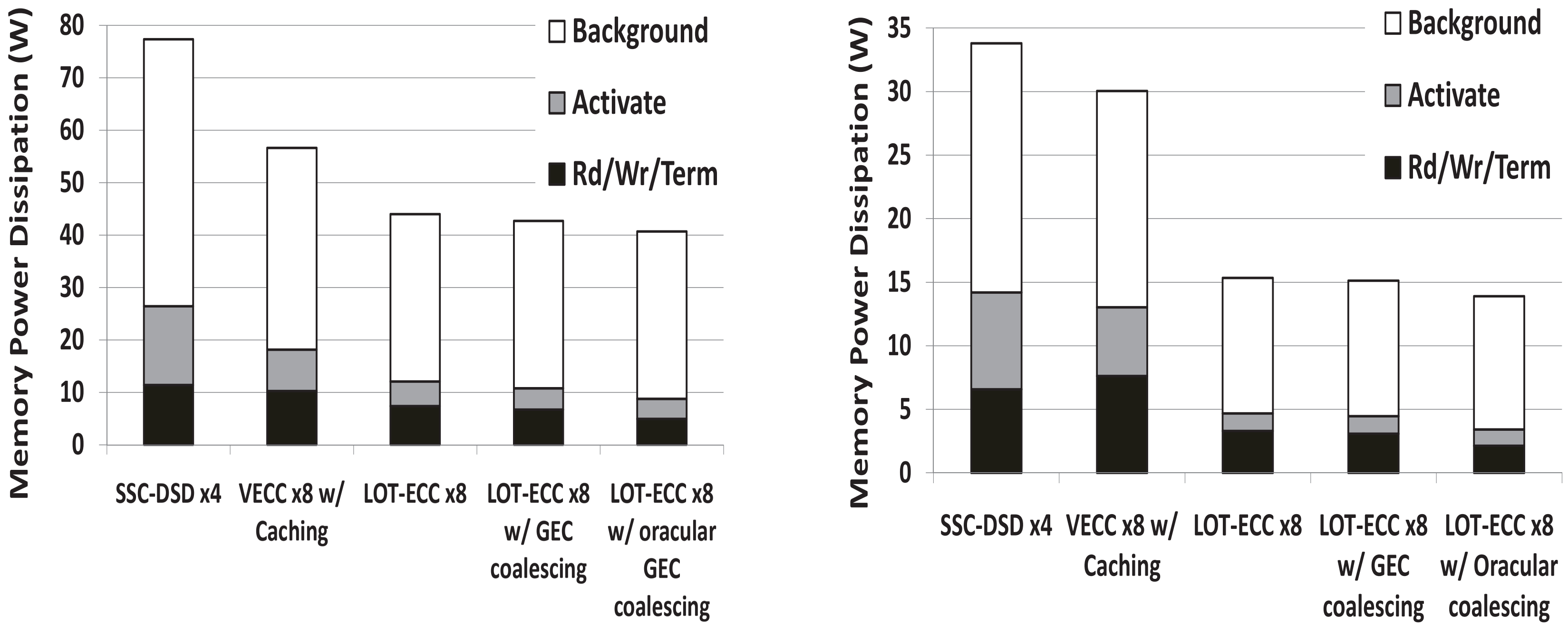
6.4.2.1 Dynamic power

LOT-ECC activates the absolute minimum number of chips required to service a request – just nine x8 chips, for example, reading/writing exactly one 64-byte cache line in a standard 8-burst access. Current chipkill solutions, on the other hand, cause *forced prefetching*, increasing dynamic power consumption. VECC [18] activates at least eighteen x8 chips per read and thirty-six x8 chips per write (data + T2EC), accessing multiple cache lines per standard 8-burst access. Commercial solutions touch thirty-six x8 chips both to read and write [53]. There is also a reduction in activation power since the size of the row buffer per chip is constant, but fewer chips are being activated. Activation power also reduces going from x4 chips to x8 chips, since fewer chips make up a rank, and is only possible with LOT-ECC.

6.4.2.2 Background power

LOT-ECC reduces the footprint of each activation, allowing unused rank/banks to transition into low-power modes. For our evaluation, we only employ the shallow low-power modes that can be entered into and exited from quickly.

The combined reduction in power is shown in Figure 6.4. In the Xeon 7500-based system, LOT-ECC reduces power consumption by 43% compared to the SSC-DSD baseline. Reducing the number of writes to GEC locations through GEC coalescing increases the savings to 45%, based on average δ values from Figure 6.3. With an oracular coalescer, savings can potentially increase to 47%. The trends are similar with 5500-based systems, although the power savings are even higher since the memory system is more heavily utilized due to the smaller number of channels. Power savings with respect to the SSC-DSD baseline are 54.6% with unoptimized x8 LOT-ECC, 55.3% with GEC coalescing, and 59% with oracular GEC coalescing. VECC's energy efficiency lies between those of SSC-DSD and LOT-ECC; we confirm that VECC's power savings compared to the SSC-DSD baseline are in line with numbers reported in the original paper, although the impact is slightly less in our 5500-based target



(a) Xeon 7500-based system

(b) Xeon 5500-based system

Figure 6.4. Overall power savings achieved using LOT-ECC due to elevated utilization. Compared to this baseline, LOT-ECC with coalescing reduces power consumption by 24.6% in 7500-based systems, and 49.7% in 5500-based systems. With both systems, the headroom for energy reduction via coalescing is small since a large fraction of the write energy is in the activation, which we avoid by guaranteeing a row-buffer hit.

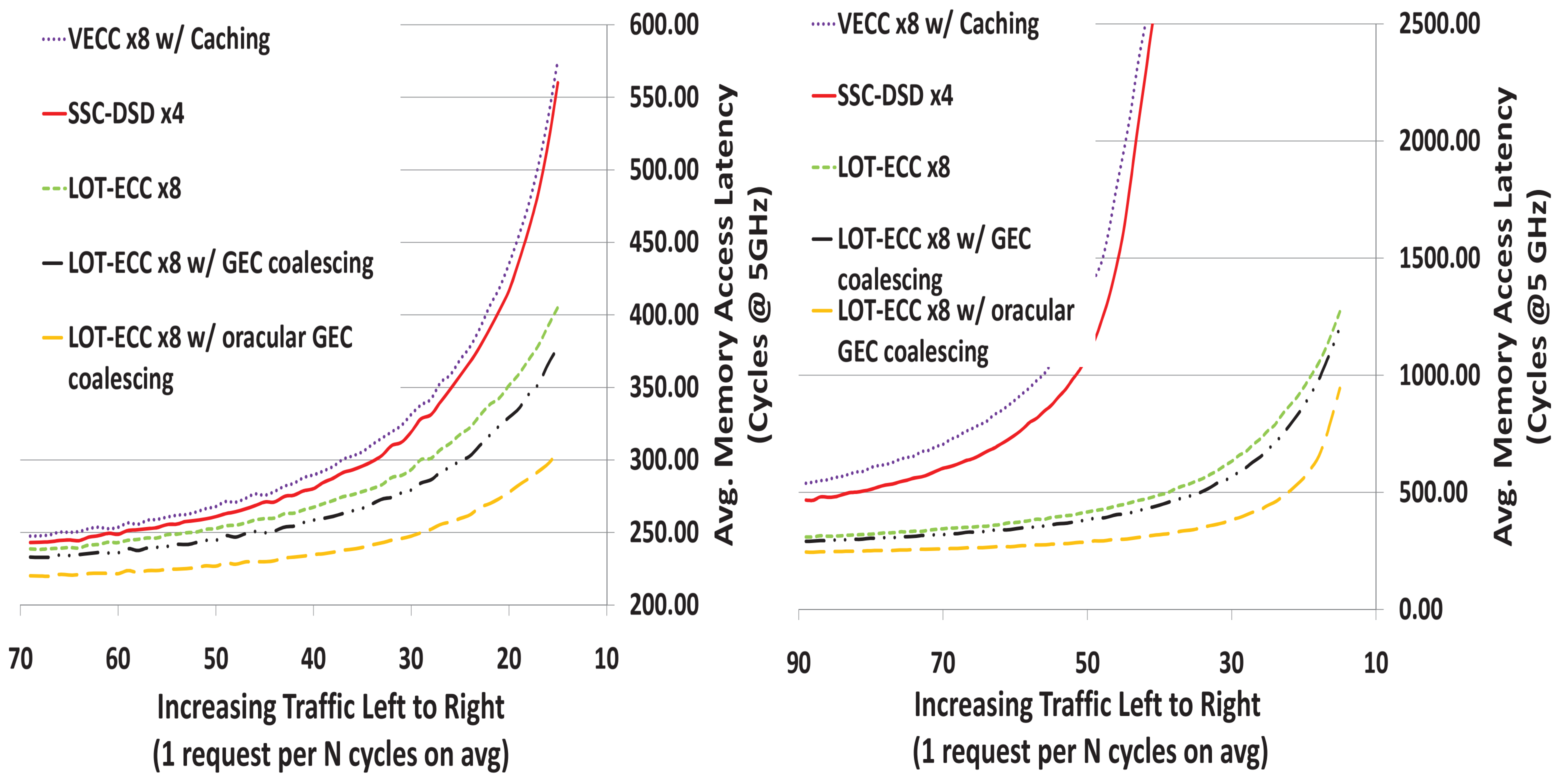
6.4.3 Performance Gains

In addition to the large energy advantage, reducing access granularity also has a small positive effect on performance. For a given total number of chips in the system, there is increased rank-level and bank-level parallelism. This reduces bank conflicts and reduces overall average memory access latency. A fraction of this gain is lost due to the the extra writes to GEC lines required along with the regular writes. Despite this overhead, LOT-ECC still comes out ahead by a small amount, even without coalescing. Figure 6.5 shows the average memory access latency.

With the 7500-based system, with traffic of 1 request per 40 cycles (just under saturation), a baseline implementation of LOT-ECC x8 provides 4.6% latency reduction compared to the SSC-DSD baseline. GEC coalescing further enhances the improvement to 7.7%, with an oracular policy providing a maximum of 16.2% reduction in latency. Latency reductions are much more substantial in the 5500-based system, since LOT-ECC relaxes the lockstep-mode constraint, providing substantially increased channel bandwidth. At a traffic rate of 70 (SSC-DSD in lockstep mode saturates soon after), latency reductions are 42.9% with LOT-ECC x8, 46.9% with GEC coalescing, and an oracular maximum of 57.3%. VECC actually performs slightly worse than the SSC-DSD baseline (confirming the results in the original paper [18]). Compared to VECC, LOT-ECC with coalescing reduces average memory latency by almost 11% in 7500-based systems, and 54% in 5500-based systems. Additionally, LOT-ECC allows the channel to operate without saturating even under substantially higher traffic rates, as much as 1 request every 40 cycles.

6.4.4 Positive Impact on System Design

LOT-ECC imposes few restrictions on the choice of DRAM parts, DIMM layout, DDR protocol, burst length, etc. In fact, with larger DRAM parts such as x16



(a) Xeon 7500 based system

(b) Xeon 5500 based system

Figure 6.5. Overall latency reductions achieved using LOT-ECC; note the Y-axis in Figure(a) or x32, the size of the data segment on each chip increases, and it is often more efficient to build strong error detection codes over larger data words. It is as yet unclear what DIMM configurations the memory industry will adopt with x16 or x32 chips, and LOT-ECC can work independent of these choices. Additionally, LOT-ECC only requires support from the memory controller and to a smaller extent from the firmware. These are relatively more amenable to change [122], and require the participation of fewer design teams, removing a few hurdles for commercial adoption.

Another consideration is that Galois field arithmetic over 16-bit or 32-bit symbols (required with the switch to x16 or x32 parts) can get complicated to implement [18, 57], increasing complexity, latency, and energy consumption. LOT-ECC utilizes simple additive checksums and parity calculations to provide strong fault tolerance, reducing the required design effort, and saving a small amount of power at the memory controller. LOT-ECC also requires minimal changes to the system in general. Absolutely no modification is required on the DRAM parts or channel. The memory controller reads, writes, and interprets bits from a commodity DDR3 channel as either data, LED, or GEC. System firmware changes are required only to reduce the amount of memory space visible to the OS, so that it can handle memory allocation accordingly.

6.4.5 Storage Overhead

LOT-ECC provides excellent benefits in terms of power, performance, complexity, and ease of system design. The price it pays to achieve these is a small increase in storage overhead. Conventional codes spread data across a large number of DRAM chips, keeping ECC overhead at 12.5%. LOT-ECC utilizes this space to store the LED information, and requires a further 12.5% storage overhead for the GEC code. However, since all memory used is strictly cheap commodity memory, this will likely be a cost that server vendors will be willing to pay [122]. This cost is also likely to be smaller than the cost of purchasing a faster server CPU, or paying for twice the energy consumption in the memory system, reducing overall TCO.

6.5 Reliability in the SSA Architecture

Since the SSA architecture (Chapter 3) proposes fairly deep changes to the DRAM itself, an additional change can be considered wherein each row also has additional storage space for a local checksum (similar to LED, Section 6.2.1), avoiding the need for a custom data layout or the use of data memory to store ECC codes. Arbitrarily wide DRAMs can be used, since each chip inherently supports the local checksum. A longer burst length (again, we assume we have the freedom to make these changes) can be used to read the LED out with every data read, depending on the cache line size and the I/O width. An example implementation is shown in Figure 6.6.

6.6 Summary of Contributions

Memory system reliability is a serious and growing concern in modern servers and blades. The power, performance, storage, and complexity overheads of providing strong levels of fault-tolerance are already very significant. Various trends in memory system design such as increasing burst length, wider I/O DRAM chips, increasing contribution of memory to system-level power/performance, larger memory capacities, greater need for parallelism, and less reliable hardware will exacerbate this problem. We present LOT-ECC, a localized and multitiered protection scheme that takes a completely different approach to reliability in order to solve these problems. We separate error detection and error correction functionality, and employ simple checksum

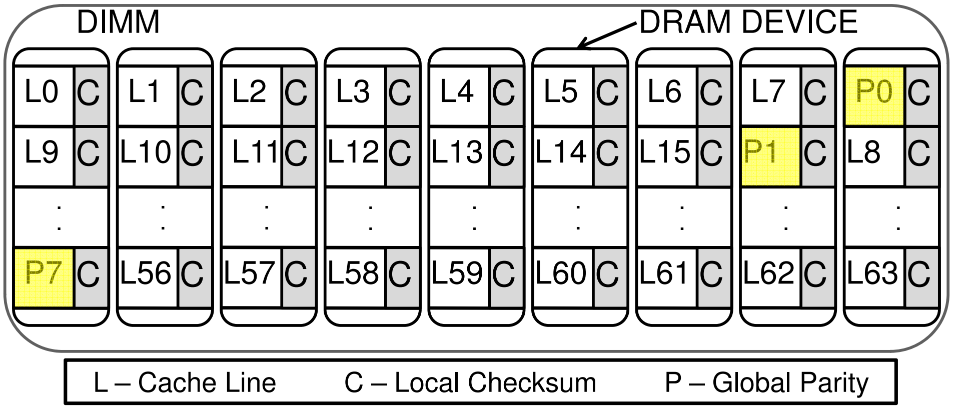


Figure 6.6. Chipkill support in SSA (only shown for 64 cache lines)

and parity codes effectively to provide strong fault-tolerance, while simultaneously simplifying implementation. Data and codes are localized to the same DRAM row to improve access efficiency. We use system firmware to store correction codes in DRAM data memory and modify the memory controller to handle data mapping, error detection, and correction.

LOT-ECC works with just a single rank of nine x8 chips, improving access granularity, energy consumption, and performance. The design reduces energy consumption by up to 55% and average memory access latency by up to 47%. Also, both commercial designs and VECC define chipkill as the failure of 1 chip out of 36, whereas LOT-ECC supports 1 dead chip in 9, a significant boost in reliability guarantee. We also propose the novel concept of a heterogeneous DIMM that contains combinations of x8, x16, and x32 DRAM parts, enabling the extension of LOT-ECC to wide-I/O DRAMs. LOT-ECC achieves these benefits while being transparent to everything but the memory controller and the firmware, making it less invasive, and more implementation friendly. It pays for all these advantages through a small increase in storage overhead, considered an acceptable cost in the memory industry as long as commodity DRAMs are used.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Contributions and Impact

In this dissertation, we presented a completely overhauled memory system architecture geared for future large-scale computing systems. We started with the most basic component of the system, the memory chips themselves. We identified energy-inefficiency caused by significant overfetch within DRAM arrays and row buffers as the key problem. We rethought this locality-centric design decision, and fundamentally re-organized the layout of the DRAM arrays and the mapping of data to these arrays. The new architecture eliminated overfetch and increased parallelism, making it better suited for future workloads. This helped reduce memory chip dynamic energy by 6X on average, memory chip static energy by 5X on average, and in some applications, helped improve performance by 54% on average. With a better chip in place, we turned to the processor-memory interconnect. We realized that the problem here is much more fundamental than an engineering design decision – it is a physics problem, caused by the poor scaling of electrical pins and off-chip interconnects. We therefore explored the application of integrated silicon photonics to build the memory channel. We proposed a novel architecture that exploited 3D-stacking technology to isolate all photonic components to a separate interface die, leaving the tightly-optimized commodity DRAM chips untouched. This will help keep costs low, and promote easier commercial acceptance. We also performed a careful design space exploration, and arrived at a design point that consumes at least 23% less energy than 3D extensions of state-of-the-art designs, enables 4X capacity, and has the potential for performance improvements due to improved bank parallelism. With such a scalable interconnect available, the next efficiency bottleneck becomes the processor-memory communication interface. The fundamental problem here is the tight integration between the memory modules, memory interconnect, and the memory controller on the processor, which severely limits interoperability and flexibility. While this worked for small systems in the past, it is simply not feasible for future systems with large capacities, hundreds of banks, and heterogeneous memory technologies sharing an interface. We proposed changing this paradigm, making the memory modules more autonomous, and employing a packet-based interface for processor-memory communication, where the processor issues a request, and reserves time on the channel for data return through a simple dual-slot scheme. Such a system significantly streamlines the operation of the memory. Finally, we looked at reliability in the memory system. The basic problem with today's reliability mechanisms is the inordinate focus on storage overhead and 100% error coverage, resulting in large ECC words, with large energy/performance overheads imposed on *every* access in order to capture a few rare error patterns. While such techniques made sense when capacity was precious and energy not a huge concern, these trends have recently reversed. In light of this, we proposed a novel multitier, localized fault-tolerance mechanism that separates error detection and error correction, employs simple parities and checksums, improves row-buffer utilization, and enables the use of more efficient x8 DRAMs. The new mechanism reduces power consumption by 55%, and latency by 47%, for a small 14% increase in storage overhead. Most importantly, it achieves all these benefits using strictly commodity memory systems, only requiring modifications to the memory controller and system firmware. We also explore an alternative implementation that employs the noncommodity disruptive SSA DRAM design from Chapter 3 to simplify data layouts and increase flexibility.

The dissertation presents various innovations that, together, constitute a sweeping overhaul of the memory system, in light of various emerging trends in computing. Several of these innovations are synergistic, and provide additive improvements in various memory figures of merit. A few examples are listed below.

- The innovations in Chapters 4 and 5 complement each other nicely – the "stack-controller" that assumes some memory controller functionality and the microrings used to provide photonic access share the same logic *interface die* in a 3D stack of DRAMs, thereby improving area utilization on this die and reducing cost.

- The innovations in Chapters 3 and 4 also complement each other well. Implementation of the photonic interconnect is simplified when every access occurs to a single chip, as in the SSA architecture.
- Similarly, the innovations in Chapter 3 aid the implementation of those in Chapter 6. The LOT-ECC mechanism can exploit hardware features such as dedicated per-row checksum storage and variable burst lengths in the SSA architecture to improve flexibility and simplify data layout.
- The energy benefits provided by the ideas presented in Chapters 3, 4, and 6 are additive. Each targets one of the three primary sources of energy consumption in memory systems – the DRAM arrays, the off-chip channel drivers, and inefficient reliability mechanisms.
- Similarly, the performance benefits provided by ideas in Chapters 3, 4, and 6 are additive. Each targets one of the three primary performance bottlenecks to memory performance – bank conflicts, pin bandwidth constraints, and inefficient reliability mechanisms.

The end result is a system that is more energy-efficient, has better performance in terms of latency and bandwidth, is simpler and more flexible, supports heterogeneity, and is reliable, all while being cost-effective. The work is a combination of different kinds of innovations. Some are very conscious of cost constraints and risk-averseness in the memory industry, and are designed to provide significant advantages while strictly conforming to various industry standards. They are less invasive, and are well-suited for relatively direct commercial adoption. Others are much more aggressive – they throw light on very fundamental problems in the memory system, and propose major changes that likely face a much steeper road to commercial adoption, but show very clearly the benefits that can be reaped by moving in such directions. In either case, we believe that the work has the potential to very substantially impact the design of the next generation of memory systems. While this study focuses on DRAM as an evaluation vehicle, the proposed architectures will apply just as well to other emerging storage technologies, such as phase change memory (PCM) and spin torque transfer RAM (STT-RAM). In fact, in that context, these innovations can be integrated into first-generation designs, enabling easier adoption by eliminating the cost of re-engineering existing systems.

7.2 Future Work

We believe that the general area of memory system architecture offers rich opportunities for future research. The various innovations proposed in this dissertation provide an efficient baseline for next-generation memory. Some of our ideas can be extended to increase their scope and usefulness. For example, the concept of autonomous memory modules can evolve into a virtualized memory system for servers, much like compute nodes today. There is also plenty of room to adapt our ideas for other kinds of applications or other kinds of computing systems. For example, we conducted this study in the context of large-scale datacenters running general-purpose workloads, but did not explore memory for visual computing, or for mobile devices. We elaborate below.

7.2.1 Autonomy and Virtualization in the Server Memory Hierarchy

Memory systems in large servers are becoming increasingly complex. There is non-determinism and heterogeneity at various levels – NUMA between DRAM attached to different sockets, different technologies at different levels in the hierarchy (DRAM, PCRAM, Memristors, etc.), variable write-cycles to achieve accuracy in flash and resistive memories, open/close page accesses, etc. Coupled with diverse workload requirements and time-varying loads, it will be increasingly difficult for the entire system to be managed and utilized effectively by a centralized controller. Finding ways to virtualize and abstract away a lot of these details will be an important research problem. For example, when a new virtual machine is fired up, it should be able to request the hypervisor for a certain amount of memory with certain characteristics (bandwidth, average latency, error-resiliency, write endurance), and use this space without needing to worry about any further operational details.

7.2.2 Memory Architectures for Visual Applications

Consumers are increasingly demanding high-definition video and immersive 3D graphics across applications, ranging from movies to gaming to virtual worlds, often wirelessly from the cloud. One use-case that was unthinkable a few years ago is fully cloud-based gaming (services provided by OnLive and Microsoft, for example) – all graphics processing and rendering happens in the cloud, and a simple compressed video stream is all that is sent to the client device. Servers for such applications have insatiable bandwidth appetites, while also requiring short response times to maintain video quality and avoid perceived lag for players. Supporting these needs will require innovations at every level of the memory system, including novel interconnect and device technologies, tailor-made scheduling mechanisms, etc. The design space becomes larger and even more interesting due to the several forms of heterogeneity present in such systems. For example, different users may play the same video or game on a small smartphone or a mid-sized tablet or a large full-HD TV screen. Each of these requires different bandwidth allocations, has a different tolerance to, say, dropped frames, and contributes differently to the overall service level of the server. There is also heterogeneity in data "importance" from a reliability perspective – while you may not care about a few dropped video frames, you probably do care about users' personal data, account information, gameplay information (levels, currency), etc. Exploiting this heterogeneity to provide efficient congestion control, reliability/availability guarantees, and quality of service makes for an interesting research problem.

7.2.3 Memory Optimizations for Handheld Devices

While much attention has been paid to server design for large datacenters, recent estimates indicate that 60 handheld devices go live for every datacenter server installed. These devices are evolving into novel form factors, with conventional laptop designs merging with tablet/smartphone designs in radically new ways. The resulting device is expected to provide features from both worlds – decent single-threaded performance for office applications and such, good graphics and a high-resolution display, and extended battery life, all within a small physical footprint. This will require a rethinking of various component architectures, the memory system being a prime candidate for innovation. For example, there will be a need for greater flexibility and dynamic reconfigurability in the memory system, making the study of aggressive low-power modes, dynamic voltage and frequency scaling, page size selection, data mapping, row-buffer management policies, and other bandwidth-latency-power-complexity tradeoff dials important research problems. Memory accesses are also faster with mobile-DRAM in general, as are page fills from "storage", thanks to plenty of non-volatile memory in close proximity to the DRAM. Combined with technologies such as 3D stacking, there is a vast design space to study and figure out exactly what the memory system should look like.

REFERENCES

[1] K. Itoh, *VLSI Memory Chip Design.* Springer, 2001.

[2] U.S. Environmental Protection Agency - Energy Star Program, *Report To Congress on Server and Data Center Energy Efficiency - Public Law 109-431*, 2007.

[3] L. Barroso and U. Holzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines.* Morgan & Claypool, 2009.

[4] P. Bose, “The Risk of Underestimating the Power of Communication,” in *NSF Workshop on Emerging Technologies for Interconnects (WETI)*, 2012.

[5] K. Lim *et al.*, “Disaggregated Memory for Expansion and Sharing in Blade Servers,” in *Proceedings of ISCA*, 2009.

[6] K. Lim *et al.*, “Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments,” in *Proceedings of ISCA*, 2008.

[7] D. Meisner, B. Gold, and T. Wenisch, “PowerNap: Eliminating Server Idle Power,” in *Proceedings of ASPLOS*, 2009.

[8] “SeaMicro Servers.” `http://www.seamicro.com`.

[9] “Calxeda Servers.” `http://www.calxeda.com`.

[10] “Micron System Power Calculator.” `http://goo.gl/4dzK6`.

[11] “Cortex-A5 Processor.” `http://www.arm.com/products/processors/cortex-a/cortex-a5.php`.

[12] J. Shin *et al.*, “A 40nm 16-Core 128-Thread CMT SPARC SoC Processor,” in *Proceedings of ISSCC*, 2010.

[13] J. Ousterhout *et al.*, “The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM,” *SIGOPS Operating Systems Review*, vol. 43(4), 2009.

[14] ITRS, “International Technology Roadmap for Semiconductors, 2007 Edition.”

[15] “Fully-Buffered DIMM Technology in HP ProLiant Servers - Technology Brief.” http://www.hp.com.

[16] U. Nawathe *et al.*, “An 8-Core 64-Thread 64b Power-Efficient SPARC SoC,” in *Proceedings of ISSCC*, 2007.

[17] B. Schroeder *et al.*, “DRAM Errors in the Wild: A Large-Scale Field Study,” in *Proceedings of SIGMETRICS*, 2009.

[18] D. Yoon and M. Erez, “Virtualized and Flexible ECC for Main Memory,” in *Proceedings of ASPLOS*, 2010.

[19] J. Condit *et al.*, “Better I/O Through Byte-Addressable, Persistent Memory,” in *Proceedings of SOSP*, 2009.

[20] M. Qureshi, V. Srinivasan, and J. Rivers, “Scalable High Performance Main Memory System Using Phase-Change Memory Technology,” in *Proceedings of ISCA*, 2009.

[21] Herb Sutter, “The Free Lunch Is Over – A Fundamental Turn Toward Concurrency in Software.” `http://www.gotw.ca/publications/concurrency-ddj.htm`.

[22] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. Jouppi, “Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores,” in *Proceedings of ISCA*, 2010.

[23] S. Beamer *et al.*, “Re-Architecting DRAM Memory Systems with Monolithically Integrated Silicon Photonics,” in *Proceedings of ISCA*, 2010.

[24] J. Torrellas, “Architectures for Extreme-Scale Computing,” *IEEE Computer*, November 2009.

[25] P. Kogge(Editor), *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems.* Defense Advanced Research Projects Agency (DARPA), 2008.

[26] ITRS, “International Technology Roadmap for Semiconductors, 2008 Update.”

[27] W. Dally and J. Poulton, *Digital System Engineering.* Cambridge, UK: Cambridge University Press, 1998.

[28] B. Rogers *et al.*, “Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling,” in *Proceedings of ISCA*, 2009.

[29] O. Mutlu and T. Moscibroda, “Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors,” in *Proceedings of MICRO*, 2007.

[30] O. Mutlu and T. Moscibroda, “Parallelism-Aware Batch Scheduling - Enhancing Both Performance and Fairness of Shared DRAM Systems,” in *Proceedings of ISCA*, 2008.

[31] C. Lee, O. Mutlu, V. Narasiman, and Y. Patt, “Prefetch-Aware DRAM Controllers,” in *Proceedings of MICRO*, 2008.

[32] E. Ipek, O. Mutlu, J. Martinez, and R. Caruana, “Self Optimizing Memory Controllers: A Reinforcement Learning Approach,” in *Proceedings of ISCA*, 2008.

[33] H. Zheng *et al.*, “Mini-Rank: Adaptive DRAM Architecture For Improving Memory Power Efficiency,” in *Proceedings of MICRO*, 2008.

[34] A. N. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. Jouppi, “Combining Memory and a Controller with Photonics through 3D-Stacking to Enable Scalable and Energy-Efficient Systems,” in *Proceedings of ISCA*, 2011.

[35] T. J. Dell, “A Whitepaper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory,” tech. rep., IBM Microelectronics Division, 1997.

[36] D. Locklear, “Chipkill Correct Memory Architecture,” tech. rep., Dell, 2000.

[37] A. N. Udipi, N. Muralimanohar, R. Balasubramonian, A. Davis, and N. Jouppi, “LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems,” in *Proceedings of ISCA*, 2012.

[38] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems - Cache, DRAM, Disk.* Elsevier, 2008.

[39] R. Swinburne, “Intel Core i7 - Nehalem Architecture Dive.” http://www.bit-tech.net/hardware/2008/11/03/intel-core-i7-nehalem-architecture-dive/.

[40] V. Romanchenko, “Quad-Core Opteron: Architecture and Roadmaps.” http://www.digital-daily.com/cpu/quad_core_opteron.

[41] D. Wentzlaff *et al.*, “On-Chip Interconnection Architecture of the Tile Processor,” in *IEEE Micro*, vol. 22, 2007.

[42] Micron Technology Inc., *Micron DDR2 SDRAM Part MT47H256M8*, 2006.

[43] U. Kang *et al.*, “8Gb 3D DDR DRAM Using Through-Silicon-Via Technology,” in *Proceedings of ISSCC*, 2009.

[44] Elpida Memory, Inc., “News Release: Elpida Completes Development of Cu-TSV (Through Silicon Via) Multi-Layer 8-Gigabit DRAM.” http://www.elpida.com/pdfs/pr/2009-08-27e.pdf.

[45] T. Pawlowski, “Hybrid Memory Cube (HMC),” in *HotChips*, 2011.

[46] R. G. Beausoleil *et al.*, “Nanoelectronic and Nanophotonic Interconnect,” *Proceedings of IEEE*, 2008.

[47] R. Barbieri *et al.*, “Design and Construction of the High-Speed Optoelectronic Memory System Demonstrator,” *Applied Opt.*, 2008.

[48] D. A. B. Miller, “Device Requirements for Optical Interconnects to Silicon Chips,” *Proceedings of IEEE Special Issue on Silicon Photonics*, 2009.

[49] N. Streibl *et al.*, “Digital Optics,” *Proceedings of IEEE*, 1989.

[50] Q. Xu *et al.*, “Micrometre-Scale Silicon Electro-Optic Modulator,” *Nature*, vol. 435, pp. 325–327, May 2005.

[51] D. Vantrease *et al.*, “Corona: System Implications of Emerging Nanophotonic Technology,” in *Proceedings of ISCA*, 2008.

[52] J. Ahn et al., “Devices and architectures for photonic chip-scale integration,” *Applied Physics A: Materials Science and Processing*, vol. 95, 2009.

[53] AMD Inc., “BIOS and Kernel Developer’s Guide for AMD NPT Family 0Fh Processors.”

[54] C. L. Chen, “Symbol Error Correcting Codes for Memory Applications,” in *Proceedings of FTCS*, 1996.

[55] E. Cooper-Balis and B. Jacob, “Fine-Grained Activation for Power Reduction in DRAM,” *IEEE Micro*, May/June 2010.

[56] S. Ankireddi and T. Chen, “Challenges in Thermal Management of Memory Modules.” `http://goo.gl/ZUn77`.

[57] S. Paul *et al.*, “Reliability-Driven ECC Allocation for Multiple Bit Error Resilience in Processor Cache,” *IEEE Transactions on Computers*, 2011.

[58] K. Sudan, N. Chatterjee, D. Nellans, M. Awasthi, R. Balasubramonian, and A. Davis, “Micro-Pages: Increasing DRAM Efficiency with Locality-Aware Data Placement,” in *Proceedings of ASPLOS-XV*, 2010.

[59] O. La, “SDRAM having posted CAS function of JEDEC standard,” 2002. United States Patent, Number 6483769.

[60] M. Kumanoya *et al.*, “An Optimized Design for High-Performance Megabit DRAMs,” *Electronics and Communications in Japan*, vol. 72(8), 2007.

[61] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, “Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0,” in *Proceedings of MICRO*, 2007.

[62] S. Thoziyoor, N. Muralimanohar, and N. Jouppi, “CACTI 5.0,” tech. rep., HP Laboratories, 2007.

[63] “Wind River Simics Full System Simulator.” `http://www.windriver.com/products/simics/`.

[64] D. Wang *et al.*, “DRAMsim: A Memory-System Simulator,” in *SIGARCH Computer Architecture News*, September 2005.

[65] “CACTI: An Integrated Cache and Memory Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model.” http://www.hpl.hp.com/research/cacti/.

[66] C. Bienia *et al.*, “The PARSEC Benchmark Suite: Characterization and Architectural Implications,” tech. rep., Princeton University, 2008.

[67] D. Bailey *et al.*, “The NAS Parallel Benchmarks,” *International Journal of Supercomputer Applications*, vol. 5, pp. 63–73, Fall 1991.

[68] "STREAM - Sustainable Memory Bandwidth in High Performance Computers." http://www.cs.virginia.edu/stream/.

[69] V. Cuppu and B. Jacob, "Concurrency, Latency, or System Overhead: Which Has the Largest Impact on Uniprocessor DRAM-System Performance," in *Proceedings of ISCA*, 2001.

[70] C. Lefurgy *et al.*, "Energy management for commercial servers.," *IEEE Computer*, vol. 36, no. 2, pp. 39–48, 2003.

[71] L. Barroso, "The Price of Performance," *Queue*, vol. 3, no. 7, pp. 48–53, 2005.

[72] A. Lebeck, X. Fan, H. Zeng, and C. Ellis, "Power Aware Page Allocation," in *Proceedings of ASPLOS*, 2000.

[73] X. Fan, H. Zeng, and C. Ellis, "Memory Controller Policies for DRAM Power Management," in *Proceedings of ISLPED*, 2001.

[74] I. Hur and C. Lin, "A Comprehensive Approach to DRAM Power Management," in *Proceedings of HPCA*, 2008.

[75] V. Delaluz *et al.*, "DRAM Energy Management Using Software and Hardware Directed Power Mode Control," in *Proceedings of HPCA*, 2001.

[76] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini, "DMA-Aware Memory Energy Management," in *Proceedings of HPCA*, 2006.

[77] H. Huang, K. Shin, C. Lefurgy, and T. Keller, "Improving Energy Efficiency by Making DRAM Less Randomly Accessed," in *Proceedings of ISLPED*, 2005.

[78] V. Delaluz *et al.*, "Scheduler-based DRAM Energy Management," in *Proceedings of DAC*, 2002.

[79] P. Burns *et al.*, "Dynamic Tracking of Page Miss Ratio Curve for Memory Management," in *Proceedings of ASPLOS*, 2004.

[80] H. Huang, P. Pillai, and K. G. Shin, "Design And Implementation Of Power-Aware Virtual Memory," in *Proceedings Of The Annual Conference On Usenix Annual Technical Conference*, 2003.

[81] J. Ahn, J. Leverich, R. S. Schreiber, and N. Jouppi, "Multicore DIMM: an Energy Efficient Memory Module with Independently Controlled DRAMs," *IEEE Computer Architecture Letters*, vol. vol.7(1), 2008.

[82] J. Ahn *et al.*, "Future Scaling of Processor-Memory Interfaces," in *Proceedings of SC*, 2009.

[83] F. A. Ware and C. Hampel, "Improving Power and Data Efficiency with Threaded Memory Modules," in *Proceedings of ICCD*, 2006.

[84] G. Loh, "3D-Stacked Memory Architectures for Multi-Core Processors," in *Proceedings of ISCA*, 2008.

[85] D. E. Atkins *et al.*, "Report of the NSF Blue-Ribbon Advisory Panel on Cyberinfrastructure," tech. rep., National Science Foundation, 2003.

[86] Raymond G. Beausoleil, HP Labs, *Personal Correspondence*, 2010.

[87] C. Natarajan *et al.*, "A Study of Performance Impact of Memory Controller Features in Multi-Processor Environment," in *Proceedings of WMPI*, 2004.

[88] N. Aggarwal *et al.*, "Power Efficient DRAM Speculation," in *Proceedings of HPCA*, 2008.

[89] John Carter, IBM Power Aware Systems, *Personal Correspondence*, 2011.

[90] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, 1st ed., 2003.

[91] A. N. Udipi, N. Muralimanohar, and R. Balasubramonian, "Non-Uniform Power Access in Large Caches with Low-Swing Wires," in *Proceedings of HiPC*, 2009.

[92] R. Ho, *On-Chip Wires: Scaling and Efficiency.* PhD thesis, Stanford University, August 2003.

[93] N. Muralimanohar *et al.*, "CACTI 6.0: A Tool to Understand Large Caches," tech. rep., University of Utah, 2007.

[94] K. Fang *et al.*, "Mode-locked Silicon Evanescent Lasers," *Optics Express*, September 2007.

[95] C. Nitta, M. Farrens, and V. Akella, "Addressing System-Level Trimming Issues in On-Chip Nanophotonic Networks," in *Proceedings of HPCA*, 2011.

[96] D. H. Woo *et al.*, "An Optimized 3D-Stacked Memory Architecture by Exploiting Excessive, High-Density TSV Bandwidth," in *Proceedings of HPCA*, 2010.

[97] K. Skadron *et al.*, "Temperature-Aware Microarchitecture," in *Proceedings of ISCA*, 2003.

[98] B. Black *et al.*, "Die Stacking (3D) Microarchitecture," in *Proceedings of MICRO*, December 2006.

[99] A. Hadke *et al.*, "OCDIMM: Scaling the DRAM Memory Wall Using WDM based Optical Interconnects," in *Proceedings of HOTI*, 2008.

[100] N. Kirman *et al.*, "Leveraging Optical Technology in Future Bus-Based Chip Multiprocessors," in *Proceedings of MICRO*, 2006.

[101] M. J. Cianchetti, J. C. Kerekes, and D. H. Albonesi, "Phastlane: A Rapid Transit Optical Routing Network," in *Proceedings of ISCA*, 2009.

[102] N. Kirman and J. F. Martinez, "An Efficient All-Optical On-Chip Interconnect Based on Oblivious Routing," in *Proceedings of ASPLOS*, 2010.

[103] J. Xue *et al.*, “An Intra-Chip Free-Space Optical Interconnect,” in *Proceedings of ISCA*, 2010.

[104] M. Awasthi *et al.*, “Handling PCM Resistance Drift with Device, Circuit, Architecture, and System Solutions,” in *Non-Volatile Memories Workshop*, 2011.

[105] E. Ordentlich *et al.*, “Coding for Limiting Current in Memristor Crossbar Memories,” in *Non-Volatile Memories Workshop*, 2011.

[106] S. Li *et al.*, “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures,” in *Proceedings of MICRO*, 2009.

[107] T. Maxino and P. Koopman, “The Effectiveness of Checksums for Embedded Control Networks,” *IEEE Transactions on Dependable and Secure Computing*, 2009.

[108] G. Loh and M. Hill, “Efficiently Enabling Conventional Block Sizes for Very Large Die-stacked DRAM Caches,” in *In Proceedings of MICRO*, 2011.

[109] JEDEC, *JESD79-3D: DDR3 SDRAM Standard*, 2009.

[110] C. Slayman *et al.*, “Impact of Error Correction Code and Dynamic Memory Reconfiguration on High-Reliability/Low-Cost Server Memory,” in *Integrated Reliability Workshop Final Report*, 2006.

[111] X. Li *et al.*, “A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility,” in *Proceedings of USENIX*, 2010.

[112] M. Blaum *et al.*, “The Reliability of Single-Error Protected Computer Memories,” *IEEE Transactions on Computers*, 1988.

[113] S. Li *et al.*, “System Implications of Memory Reliability in Exascale Computing,” in *Proceedings of SC*, 2011.

[114] “Advanced Memory Protection for HP ProLiant 300 Series G4 Servers.” `http://goo.gl/M2Mqa`.

[115] D. Yoon and M. Erez, “Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches,” in *Proceedings of ISCA*, 2009.

[116] N. N. Sadler and D. J. Sorin, “Choosing an Error Protection Scheme for a Microprocessor’s L1 Data Cache,” in *Proceedings of ICCD*, 2006.

[117] “Dell PowerEdge 11th Generation Servers: R810, R910, and M910.” `http://goo.gl/30QkU`.

[118] “HP ProLiant DL580 G7 Server Technology.” `http://goo.gl/aOQ3L`.

[119] “Oracle’s Sun Fire X4800 Server Architecture.” `http://goo.gl/h0efI`.

[120] “Dell Help Me Choose: Memory.” `http://goo.gl/paF1c`.

[121] "IBM System x3550 M3 Product Guide." `http://goo.gl/yFPLS`.

[122] HP Industry Standard Server Group and HP Business Critical Server Group, *Personal Correspondence*, 2011.