

Rethinking DRAM Design and Organization for Energy-Constrained Multi-Cores

Aniruddha N. Udipi
University of Utah
Salt Lake City, UT
udipi@cs.utah.edu

Naveen Muralimanohar
Hewlett-Packard Laboratories
Palo Alto, CA
naveen.murali@hp.com

Niladrish Chatterjee
University of Utah
Salt Lake City, UT
nil@cs.utah.edu

Rajeev Balasubramonian
University of Utah
Salt Lake City, UT
rajeev@cs.utah.edu

Al Davis
University of Utah
Salt Lake City, UT
ald@cs.utah.edu

Norman P. Jouppi
Hewlett-Packard Laboratories
Palo Alto, CA
norm.jouppi@hp.com

ABSTRACT

DRAM vendors have traditionally optimized the cost-per-bit metric, often making design decisions that incur energy penalties. A prime example is the overfetch feature in DRAM, where a single request activates thousands of bitlines in many DRAM chips, only to return a single cache line to the CPU. The focus on cost-per-bit is questionable in modern-day servers where operating costs can easily exceed the purchase cost. Modern technology trends are also placing very different demands on the memory system: (i) queuing delays are a significant component of memory access time, (ii) there is a high energy premium for the level of reliability expected for business-critical computing, and (iii) the memory access stream emerging from multi-core systems exhibits limited locality. All of these trends necessitate an overhaul of DRAM architecture, even if it means a slight compromise in the cost-per-bit metric.

This paper examines three primary innovations. The first is a modification to DRAM chip microarchitecture that retains the traditional DDRx SDRAM interface. Selective Bitline Activation (SBA) waits for both RAS (row address) and CAS (column address) signals to arrive before activating exactly those bitlines that provide the requested cache line. SBA reduces energy consumption while incurring slight area and performance penalties. The second innovation, Single Subarray Access (SSA), 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. It requires a different interface to the memory controller, reduces dynamic and background energy (by about 6X), incurs a slight area penalty (4%), and can even lead to performance improvements (up to 10%) by reducing queuing delays. The third innovation further penalizes the cost-per-bit metric by adding a checksum feature to each cache line. This checksum error-detection feature can then be used to

build stronger RAID-like fault tolerance, including chipkill-level reliability. Such a technique is especially crucial for the SSA architecture where the entire cache line is localized to a single chip. This DRAM chip microarchitectural change leads to a dramatic reduction in the energy and storage overheads for reliability. The proposed architectures will also apply to other emerging memory technologies (such as resistive memories) and will be less disruptive to standards, interfaces, and the design flow if they can be incorporated into first-generation designs.

Categories and Subject Descriptors

B.3.1 [Memory Structures]: Semiconductor Memories—*Dynamic memory (DRAM)*; B.3.2 [Memory Structures]: Design Styles—*Primary memory*; B.8.1 [Performance and Reliability]: Reliability, Testing and Fault-Tolerance; C.5.5 [Computer System Implementation]: Servers

General Terms

Design, Performance, Reliability

Keywords

DRAM Architecture, Energy-efficiency, Locality, Chipkill, Subarrays

1. INTRODUCTION

The computing landscape is undergoing a major change, primarily enabled by ubiquitous wireless networks and the rapid increase in the usage of mobile devices which access the web-based information infrastructure. It is expected that most CPU-intensive computing may either happen in servers housed in large datacenters, *e.g.*, cloud computing and other web services, or in many-core high-performance computing (HPC) platforms in scientific labs. In both situations, it is expected that the memory system will be problematic in terms of performance, reliability, and power consumption.

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISCA'10, June 19–23, 2010, Saint-Malo, France.

Copyright 2010 ACM 978-1-4503-0053-7/10/06 ...\$10.00.

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. Consider the following technological trends that place very different demands on future DRAM architectures:

- *Energy*: While energy was never a first-order design constraint in prior DRAM systems, it has certainly emerged as the primary constraint today, especially in datacenters. Energy efficiency in datacenters has already been highlighted as a national priority [50]. Many studies attribute 25-40% of total datacenter power to the DRAM system [11, 33, 34, 37]. Modern DRAM architectures are ill-suited for energy-efficient operation because they are designed to fetch much more data than required. This *overfetch* wastes dynamic energy. Today’s DRAMs employ coarse-grained power-down tactics to reduce area and cost, but finer grained approaches can further reduce background energy.
- *Reduced locality*: Single-core workloads typically exhibit high locality. Consequently, current DRAMs 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. The high degree of multi-threading in future multi-cores [42] 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” that are comprised of many commodity DIMMs) that serve several CPU sockets in an effort to increase resource utilization [34]. This mandates that future DRAM architectures place a lower priority on locality and a higher priority on parallelism.
- *Queuing Delays*: For several years, queuing delays at the memory controller were relatively small because a single core typically had relatively few pending memory operations and DRAM systems were able to steeply increase peak memory bandwidth every year [20]. In the future, the number of pins per chip is expected to grow very slowly. The 2007 ITRS Roadmap [26] 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. This implies that requests from many cores will be competing to utilize the limited pin bandwidth. Several studies have already highlighted the emergence of queuing delay as a major bottleneck [24, 31, 40, 41, 44, 56]. A DRAM architecture that is geared towards higher parallelism will likely be able to de-queue requests faster and better utilize the available limited data bandwidth.
- *Efficient Reliability*: Recent studies have highlighted the need for DRAM architectures that are resilient to single faults or even failure within an entire DRAM chip [8, 46], especially in datacenter platforms. Because these fault-tolerant solutions are built upon commodity DRAM chips,

they incur very high overheads in terms of energy and cost. New DRAM architectures can provide much more efficient reliability if fault-tolerant features are integrated into the DRAM chip microarchitecture at design time.

- *Lower relevance of DRAM chip area*: DRAM vendors have long optimized the cost-per-bit metric. However, given that datacenters consume several billion kilowatt hours of energy every year [50], it has been shown that the 3-year operating energy costs of today’s datacenters equal the capital acquisition costs [33]. Therefore, it may now be acceptable to incur a slightly higher cost-per-bit when purchasing DRAM as long as it leads to significantly lower energy footprints during operation.

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 highly reliable, high performance operation with extremely low energy footprints, all within acceptable area bounds. In this work, we propose two independent designs, both attempting to activate the minimum circuitry required to read a single cache line.

We make the following three significant contributions:

- 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 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.
- In order to provide chipkill-level reliability [18, 35] even though we are reading a cache line out of a single DRAM device, we propose adding a checksum to each cache line in the SSA DRAM to provide error detection. We then evaluate the use of RAID techniques to reconstruct cache lines in the event of a chip failure.

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).

2. BACKGROUND AND MOTIVATION

2.1 DRAM Basics and Baseline Organization

We first describe the typical modern DRAM architecture [27]. For most of the paper, our discussion will focus on the dominant DRAM architecture today: JEDEC-style DDRx SDRAM, an example is shown in Figure 1.

Modern processors [45, 48, 54] 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 [38]. 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

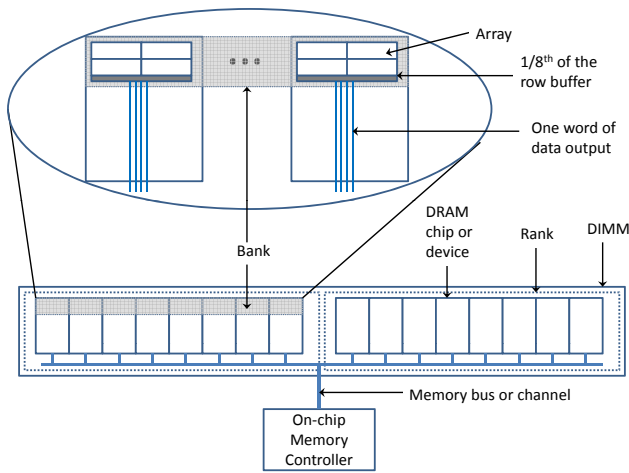


Figure 1: An example DDRx SDRAM architecture with 1 DIMM, 2 ranks, and 8 x4 DRAM chips per rank.

chips. We will call this a *rank-set*. Exactly one *rank-set* 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 1 shows an example DIMM with 16 total DRAM chips forming two rank-sets.

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-set would require 8 DRAM chips (Figure 1 only shows 8 $x4$ chips per rank-set to simplify the figure). If the DIMM supports ECC, the data bus expands to 72-bits and the rank-set would consist of 9 $x8$ DRAM chips. When a rank is selected, all DRAM chips in the rank-set receive address and command signals from the memory 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 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 is 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 is 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 eight 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-set.

2.2 Motivational Data

Recent studies have indicated the high energy needs of datacenters [50] and that memory contributes up to 40% of total server power consumption [11, 33, 37]. We start with a workload characterization on our simulation infrastructure (methodology details in Section 4.1). Figure 2 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). 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 don't have much reuse in the row-buffer. This data is also corroborated by prior work exploiting this fact through the use of Micro-Pages [47]. This means that the energy costs of activating an entire 8 KB row is amortized over very few accesses, wasting significant energy.

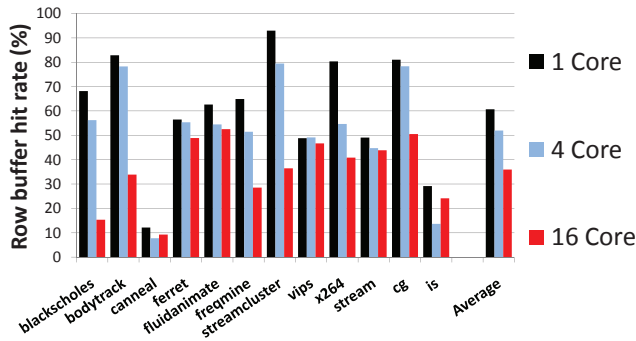


Figure 2: Row buffer hit rate trend

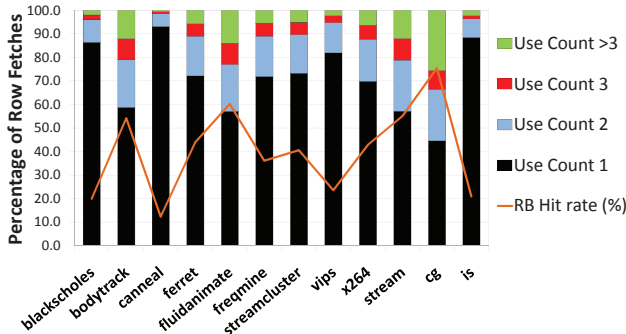


Figure 3: Row use count for 8 cores

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 3.1) requires minor changes to DRAM chip microarchitecture, but is compatible with existing DRAM standards and interfaces. The second novel design (Single Subarray Access, Section 3.2) requires non-trivial changes to DRAM chip microarchitecture and its interface to the memory controller. Section 3.3 describes our novel chipkill solution for the proposed architecture.

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 gets 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.

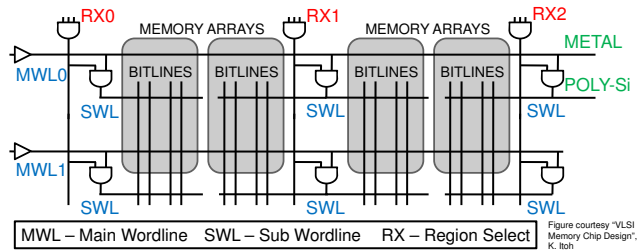


Figure 4: Hierarchical wordline with region select.

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 [26] 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 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 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 4). 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 [25] to reduce delay (rather than energy). Until now, other techniques (metal shunted wordlines [28], for instance) partially achieved what has been perceived as the

¹Many 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 [27]. 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 [29]. We refer to our scheme as Posted-RAS because the RAS is saved in a register until the arrival of the CAS.

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 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 [39, 49] (more details in Section 4.1) to quantify the area overhead. For the specific DRAM part described in Section 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 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.

3.2 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 sub-section 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

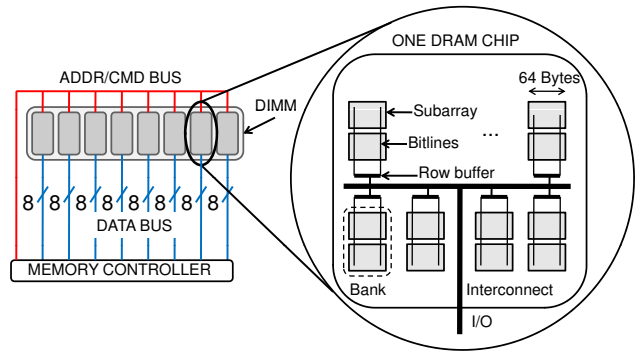


Figure 5: SSA DRAM Architecture.

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.

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 5 shows the entire memory channel and how various sub-components 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 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 5, 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 intra-chip or bank-level concurrency. We next examine our new design in greater detail.

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 isn’t 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.

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 [39, 49]. Figure 5 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 is 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 [27].

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 acti-

vated. 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 [38] 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 .

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. The impact on reliability is discussed in the next sub-section. 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 a superior to SBA, although it requires a larger re-design 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.

3.3 Chipkill

Recent studies have shown that DRAMs are often plagued with errors and can lead to significant server downtime in datacenters [46]. Therefore, a low-power DRAM design targeted at datacenters must be amenable to an architecture that provides a high standard of reliability. A common expectation of business-critical server DRAM systems is that they are able to withstand a single DRAM chip failure. Just

as an entire family of error-resilient schemes can be built for bit failures (for example, Single Error Correction Double Error Detection, SECDED), a family of error-resilient schemes can also be built for chip failure (for example, Single Chip error Correction Double Chip error Detection, SCCDCD), and these are referred to as *Chipkill* [18, 35]. We now focus on the design of an SCCDCD chipkill scheme; the technique can be easily generalized to produce stronger flavors of error-resilience.

First, consider a conventional design where each word (say 64 bits) has been appended with an 8-bit ECC code, to provide SECDED. For a chipkill scheme, each DRAM chip can only contribute one bit out of the 72-bit word. If a chip were to contribute any more, chip failure would mean multi-bit corruption within the 72-bit word, an error that a SECDED code cannot recover from. Therefore, each 72-bit word must be striped across 72 DRAM chips. When a 64-byte cache line is requested, 72 bytes are read out of the 72 DRAM chips, making sure that each 72-bit word obtains only a single bit from each DRAM chip. Such an organization was adopted in the Dell Poweredge 6400/6450 servers [35]. This provides some of the rationale for current DRAM systems that stripe a cache line across several DRAM chips. This is clearly energy inefficient as 72 DRAM chips are activated and a very small fraction of the read bits are returned to the CPU. It is possible to reduce the number of DRAM chips activated per access if we attach ECC codes to smaller words as has been done in the IBM Netfinity systems [18]. This will have high storage overhead, but greater energy efficiency. For example, in a design attaching an ECC word to 8 bits, say, one may need five extra DRAM chips per eight DRAM chips on a single DIMM. ECC gets progressively more efficient as the granularity at which it is attached is increased.

In the SSA design, we intend to get the entire cache line from a single DRAM chip access. If this DRAM chip were to produce corrupted data, there must be a way to re-construct it. This is a problem formulation almost exactly the same as that for reliable disks. We therefore adopt a solution very similar to that of the well-studied RAID [20] solution for disks, but that has never been previously employed within a DIMM. Note that some current server systems do employ RAID-like schemes across DIMMs [2, 6]; within a DIMM, conventional ECC with an extra DRAM chip is employed. These suffer from high energy overheads due to the large number of chips accessed on every read or write. Our approach is distinct and more energy-efficient. In an example RAID design, a single disk serves as the “parity” disk to eight other data disks. On a disk access (specifically in RAID-4 and RAID-5), only a single disk is read. A checksum associated with the read block (and stored with the data block on the disk) lets the RAID controller know if the read is correct or not. If there is an error, the RAID controller re-constructs the corrupted block by reading the other seven data disks and the parity disk. In the common error-free case, only one disk needs to be accessed because the checksum enables self-contained error detection. It is not fool-proof because the block+checksum may be corrupted, and the checksum may coincidentally be correct (the larger the checksum, the lower the probability of such a silent data corruption). Also, the parity overhead can be made arbitrarily low by having one parity disk for many data disks. This is still good enough for error detection and recovery because the checksum has already played the role of detecting and identifying the cor-

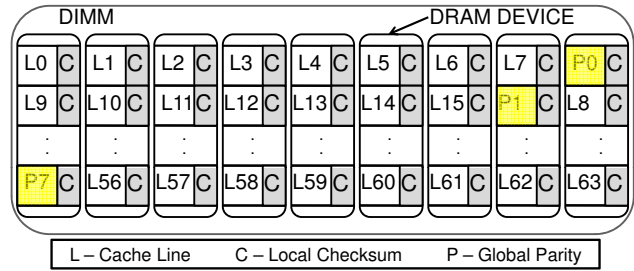


Figure 6: Chipkill support in SSA (only shown for 64 cache lines).

rupted bits. The catch is that writes are more expensive as every write requires a read of the old data block, a read of the old parity block, a write to the data block, and a write to the parity block. RAID-5 ensures that the parity blocks are distributed among all nine disks so that no one disk emerges as a write bottleneck.

We adopt the same RAID-5 approach in our DRAM SSA design (Fig. 6). The DRAM array microarchitecture must now be modified to not only accommodate a cache line, but also its associated checksum. We assume an eight bit checksum, resulting in a storage overhead of 1.625% for a 64-byte cache line. The checksum function uses bit inversion so that stuck-at-zero faults do not go undetected. The checksum is returned to the CPU after the cache line return and the verification happens in the memory controller (a larger burst length is required, not additional DRAM pins). We cannot allow the verification to happen at the DRAM chip because a corrupted chip may simply flag all accesses as successfully passing the checksum test. The DIMM will now have one extra DRAM chip, a storage overhead of 12.5% for our evaluated platform. Most reads only require that one DRAM chip be accessed. A write requires that two DRAM chips be read and then written. This is the primary performance overhead of this scheme as it increases bank contention (note that an increase in write latency does not impact performance because of read-bypassing at intermediate buffers at the memory controller). We quantify this effect in the results section. This also increases energy consumption, but it is still far less than the energy of reliable or non-reliable conventional DRAM systems.

Most chipkill-level reliability solutions have a higher storage overhead than our technique. As described above, the energy-efficient solutions can have as high as 62.5% overhead, the Dell Poweredge solution has a 12.5% overhead (but requires simultaneous access to 72 DRAM chips), and the rank-sub-setting DRAM model of Ahn et al. [8] has a 37.5% overhead. The key to our higher efficiency is the localization of an entire cache line to a single DRAM chip and the use of checksum for self-contained error detection at modest overhead (1.625%) plus a parity chip (12.5% for 8-way parity). Even on writes, when four DRAM accesses are required, we touch fewer DRAM chips and read only a single cache line in each, compared to any of the prior solutions for chipkill [8, 18, 35]. Therefore, our proposed SSA architecture with chipkill functionality is better than other solutions in terms of area cost and energy. As we show in Section 4, the performance impact of write contention is also low because of the high degree of bank concurrency afforded by SSA.

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

Table 1: General parameters

4. RESULTS

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 1. Our simulation infrastructure uses Virtutech’s SIMICS [5] 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 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 [52] framework and from [27]. DRAM timing information was obtained from Micron datasheets [38].

Area, latency and energy numbers for DRAM banks were obtained from CACTI 6.5 [1], 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 multi-level 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 work, we do not explore this trade-off, instead focusing on the maximum energy reduction possible. The DRAM energy parameters used in our evaluation are listed in Table 2. We evaluate our proposals on subsets of the multi-threaded PARSEC [13], NAS [9] and STREAM [4] 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.

Component	Dynamic Energy (nJ)
Decoder/Wordline - Baseline	1.429
Decoder/Wordline - SBA	0.024
Decoder/Wordline - SSA	0.013
Bitlines - Baseline	19.282
Bitlines - SBA/SSA	0.151
Termination Resistors Baseline/SBA/SSA	7.323
Output Drivers	2.185
Others - Baseline/SBA/SSA	1.143
Low-power mode	Background Power (mW)
Active	104.5
Power Down (3 mem. cyc)	19
Self Refresh (200 mem. cyc)	10.8

Table 2: Energy parameters

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 relative to the baseline organization.

4.2.1 Energy Characteristics

Figure 7 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 8 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 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, pre-decoder, inter-bank bus, bus termination, etc. remains the same. Hence, their contribution to the total energy increases as bitline energy goes down. In an open page policy, there is the possibility of open row hits that are extremely energy efficient since they don’t involve any additional activation. We constructed a simple analytical model to estimate how many cache lines had to be read out of a row buffer for the open-page baseline to match the energy per access of SBA/SSA. The average

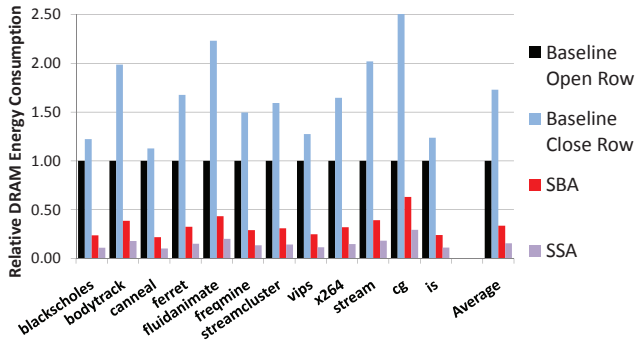


Figure 7: DRAM dynamic energy consumption

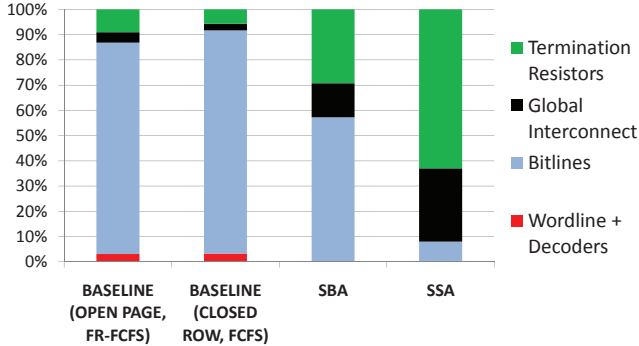


Figure 8: Contributors to DRAM dynamic energy

level of row reuse needs to be 23, and we already saw in Figure 3 that the reuse level in applications is much lower.

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 2, obtained from the Micron datasheet and power system calculator [3, 38]. 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 just 3 memory cycles. We vary I (Idle cycle threshold) as multiples of the wake-up time W . Figures 9 and 10 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

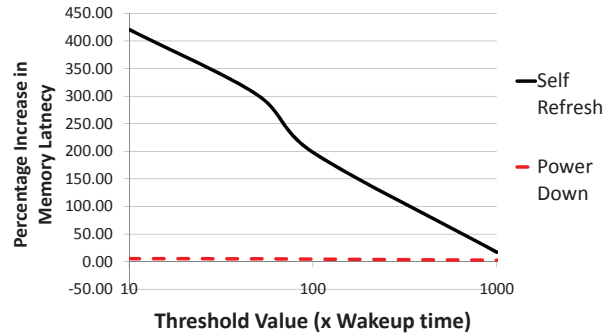


Figure 9: Memory latency impact of using low-power states

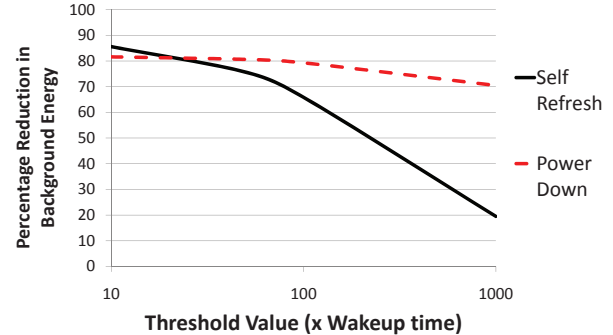


Figure 10: Energy reduction using low-power states

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” 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.

4.2.2 Performance Characteristics

Employing either the SBA or SSA schemes impacts memory access latency (positively or negatively) as shown in Figure 11. Figure 12 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 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 12, for these three models, the queuing delay is the dominant contributor to total memory access

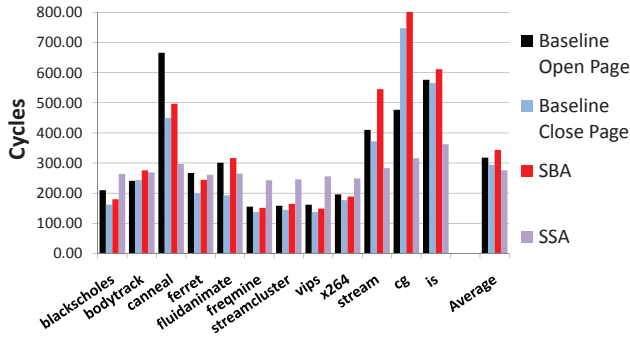


Figure 11: Average main memory latency

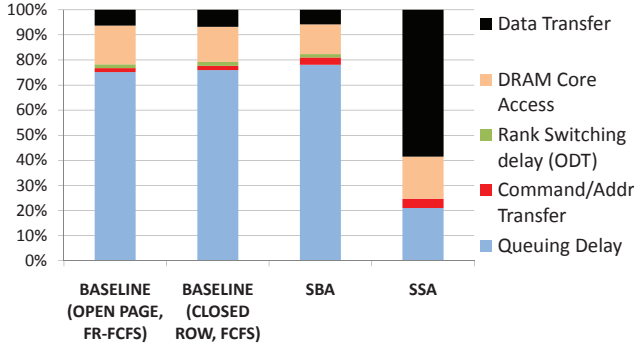


Figure 12: Contributors to total memory latency

latency. Prior work [15] 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. As a result, overall, SSA in fact outperforms all other models.

Figure 13 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 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 six

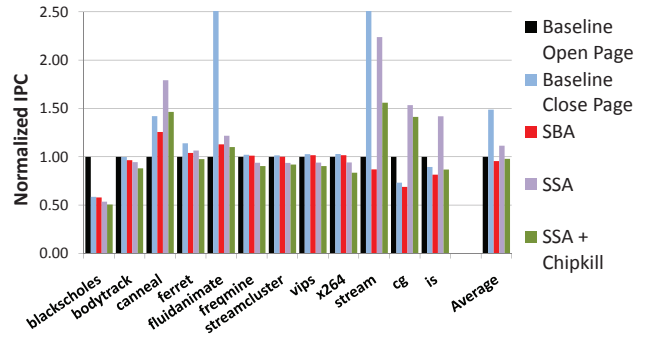


Figure 13: Normalized IPCs of various organizations

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 queuing pressures on memory controllers. Figure 13 shows the IPC degradation caused when we augment SSA with our chipkill solution. Note that this is entirely because of the increased bank contention during writes. On average, the increase in memory latency is a little over 70%, resulting in a 12% degradation in IPC. Compared to the non-chipkill SSA, there is also additional energy consumption on every write, resulting in a 2.2X increase in dynamic energy to provide chipkill-level reliability, which is still significantly lower than a baseline organization.

4.2.3 System Level Characteristics

To evaluate the system level impact of our schemes, we use a simple model where the DRAM subsystem consumes 40% of total system power, (32% dynamic and 8% background). Changes in performance are assumed to linearly impact the power consumption in the rest of the system, both background and dynamic. Having taken these into account, on average, we see 18% and 36% reductions in system power with SBA and SSA respectively.

5. RELATED WORK

The significant contribution of DRAM to overall system power consumption has been documented in several studies [10, 32, 37]. A majority of techniques aimed at conserving DRAM energy try to transition inactive DRAM chips to low power states [30] as effectively as possible to decrease the background power. Researchers have investigated prediction models for DRAM activity [19], adaptive memory controller policies [23], compiler-directed hardware-assisted data layout [16], management of DMA and CPU generated request streams to increase DRAM idle periods [22, 43] as well as managing the virtual memory footprint and physical memory allocation schemes [14, 17, 21] 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 ac-

cess. Zheng et al. suggest the subdivision of a conventional DRAM rank into mini-ranks [56] comprising of a subset of DRAM devices. Ahn et al. [7, 8] propose a scheme where each DRAM device can be controlled individually via a demux register per channel that is responsible for routing all command signals to the appropriate chip. In their multi-core 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 [53]. On the other hand, Sudan et al. [47] 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 [36]), and design for systems with photonic interconnects (Vantrease et al [51] and Beamer et al. [12]). Yoon and Erez [55] 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. Our SBA mechanism with Posted-RAS is a novel way to reduce activation and can eliminate overfetch. The SSA mechanism re-organizes the layout of a DRAM chip to support small subarrays and the mapping of data to only activate a single subarray. Our chipkill solution that uses checksum-based detection and RAID-like correction has not been previous considered and is more effective than those used for prior DRAM chipkill solutions [8, 18, 35].

6. CONCLUSIONS

We propose two 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). Our solutions will require non-trivial initial design effort on the part of DRAM vendors and will incur minor area/cost increases. A similar architecture will likely also be suitable for emerging memory technologies such as PCM and STT-RAM. The memory energy reductions from our techniques are substantial for both dynamic (6X) and background (5X) components. We observe that fetching exactly a cache line with SSA can improve performance in some cases (50% on average) due to its close-page policy and also because it helps alleviate bank contention in some memory-sensitive applications. In other applications that are not as constrained by bank contention, the SSA policy can cause performance degradations (13% on average) because of long cache line data transfer times out of a single DRAM chip.

Any approach that reduces the number of chips used to store a cache line also increases the probability of correlated errors. With SSA, we read an entire cache line out of a single DRAM array, so the potential for correlated errors is increased. In order to provide chipkill-level reliability in concert with SSA, we introduced checksums stored for each

cache line in the DRAM, similar to that provided in hard drives. Using the checksum we can provide robust error detection capabilities, and provide chipkill-level reliability through RAID techniques (however in our case, we use a Redundant Array of Inexpensive DRAMs). We show that this approach is more effective in terms of area and energy than prior chipkill approaches, and only incurs a 12% performance penalty compared to an SSA memory system without chipkill.

7. ACKNOWLEDGMENTS

This work was supported in parts by NSF grants CCF-0430063, CCF-0811249, CCF-0916436, NSF CAREER award CCF-0545959, SRC grant 1847.001, and the University of Utah. The authors would also like to thank Utah Arch group members Kshitij Sudan, Manu Awasthi, and David Nellans for help with the baseline DRAM simulator.

8. REFERENCES

- [1] CACTI: An Integrated Cache and Memory Access Time, Cycle Time, Area, Leakage, and Dynamic Power Model. <http://www.hpl.hp.com/research/cacti/>.
- [2] HP Advanced Memory Protection Technologies - Technology Brief. <http://www.hp.com>.
- [3] Micron System Power Calculator. [http://www.micron.com/support/part info/powercalc](http://www.micron.com/support/part%20info/powercalc).
- [4] STREAM - Sustainable Memory Bandwidth in High Performance Computers. <http://www.cs.virginia.edu/stream/>.
- [5] Virtutech Simics Full System Simulator. <http://www.virtutech.com>.
- [6] M. Abbott et al. Durable Memory RS/6000 System Design. In *Proceedings of International Symposium on Fault-Tolerant Computing*, 1994.
- [7] 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.7(1), 2008.
- [8] J. H. Ahn, N. P. Jouppi, C. Kozyrakis, J. Leverich, and R. S. Schreiber. Future Scaling of Processor-Memory Interfaces. In *Proceedings of SC*, 2009.
- [9] D. Bailey et al. The NAS Parallel Benchmarks. *International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.
- [10] L. Barroso. The Price of Performance. *Queue*, 3(7):48–53, 2005.
- [11] L. Barroso and U. Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.
- [12] S. Beamer et al. Re-Architecting DRAM Memory Systems with Monolithically Integrated Silicon Photonics. In *Proceedings of ISCA*, 2010.
- [13] C. Benia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical report, Department of Computer Science, Princeton University, 2008.
- [14] P. Burns et al. Dynamic Tracking of Page Miss Ratio Curve for Memory Management. In *Proceedings of ASPLOS*, 2004.

- [15] 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.
- [16] V. Delaluz et al. DRAM Energy Management Using Software and Hardware Directed Power Mode Control. In *Proceedings of HPCA*, 2001.
- [17] V. Delaluz et al. Scheduler-based DRAM Energy Management. In *Proceedings of DAC*, 2002.
- [18] T. J. Dell. A Whitepaper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory. Technical report, IBM Microelectronics Division, 1997.
- [19] X. Fan, H. Zeng, and C. Ellis. Memory Controller Policies for DRAM Power Management. In *Proceedings of ISLPED*, 2001.
- [20] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 4th edition, 2007.
- [21] 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.
- [22] H. Huang, K. Shin, C. Lefurgy, and T. Keller. Improving Energy Efficiency by Making DRAM Less Randomly Accessed. In *Proceedings of ISLPED*, 2005.
- [23] I. Hur and C. Lin. A Comprehensive Approach to DRAM Power Management. In *Proceedings of HPCA*, 2008.
- [24] E. Ipek, O. Mutlu, J. Martinez, and R. Caruana. Self Optimizing Memory Controllers: A Reinforcement Learning Approach. In *Proceedings of ISCA*, 2008.
- [25] K. Itoh. *VLSI Memory Chip Design*. Springer, 2001.
- [26] ITRS. International Technology Roadmap for Semiconductors, 2007 Edition. <http://www.itrs.net/Links/2007ITRS/Home2007.htm>.
- [27] B. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems - Cache, DRAM, Disk*. Elsevier, 2008.
- [28] M. Kumanoya et al. An Optimized Design for High-Performance Megabit DRAMs. *Electronics and Communications in Japan*, 72(8), 2007.
- [29] O. La. SDRAM having posted CAS function of JEDEC standard, 2002. United States Patent, Number 6483769.
- [30] A. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power Aware Page Allocation. In *Proceedings of ASPLOS*, 2000.
- [31] C. Lee, O. Mutlu, V. Narasiman, and Y. Patt. Prefetch-Aware DRAM Controllers. In *Proceedings of MICRO*, 2008.
- [32] C. Lefurgy et al. Energy management for commercial servers. *IEEE Computer*, 36(2):39–48, 2003.
- [33] K. Lim et al. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In *Proceedings of ISCA*, 2008.
- [34] K. Lim et al. Disaggregated Memory for Expansion and Sharing in Blade Servers. In *Proceedings of ISCA*, 2009.
- [35] D. Locklear. Chipkill Correct Memory Architecture. Technical report, Dell, 2000.
- [36] G. Loh. 3D-Stacked Memory Architectures for Multi-Core Processors. In *Proceedings of ISCA*, 2008.
- [37] D. Meisner, B. Gold, and T. Wenisch. PowerNap: Eliminating Server Idle Power. In *Proceedings of ASPLOS*, 2009.
- [38] Micron Technology Inc. *Micron DDR2 SDRAM Part MT47H256M8*, 2006.
- [39] 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.
- [40] O. Mutlu and T. Moscibroda. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *Proceedings of MICRO*, 2007.
- [41] O. Mutlu and T. Moscibroda. Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems. In *Proceedings of ISCA*, 2008.
- [42] U. Nawathe et al. An 8-Core 64-Thread 64b Power-Efficient SPARC SoC. In *Proceedings of ISSCC*, 2007.
- [43] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini. DMA-Aware Memory Energy Management. In *Proceedings of HPCA*, 2006.
- [44] B. Rogers et al. Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling. In *Proceedings of ISCA*, 2009.
- [45] V. Romanchenko. Quad-Core Opteron: Architecture and Roadmaps. http://www.digital-daily.com/cpu/quad_core_opteron.
- [46] B. Schroeder, E. Pinheiro, and W. Weber. DRAM Errors in the Wild: A Large-Scale Field Study. In *Proceedings of SIGMETRICS*, 2009.
- [47] 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.
- [48] R. Swinburne. Intel Core i7 - Nehalem Architecture Dive. <http://www.bit-tech.net/hardware/2008/11/03/intel-core-i7-nehalem-architecture-dive/>.
- [49] S. Thoziyoor, N. Muralimanohar, and N. Jouppi. CACTI 5.0. Technical report, HP Laboratories, 2007.
- [50] U.S. Environmental Protection Agency - Energy Star Program. *Report To Congress on Server and Data Center Energy Efficiency - Public Law 109-431*, 2007.
- [51] D. Vantrease et al. Corona: System Implications of Emerging Nanophotonic Technology. In *Proceedings of ISCA*, 2008.
- [52] D. Wang et al. DRAMsim: A Memory-System Simulator. In *SIGARCH Computer Architecture News*, volume 33, September 2005.
- [53] F. A. Ware and C. Hampel. Improving Power and Data Efficiency with Threaded Memory Modules. In *Proceedings of ICCD*, 2006.
- [54] D. Wentzlaff et al. On-Chip Interconnection Architecture of the Tile Processor. In *IEEE Micro*, volume 22, 2007.
- [55] D. Yoon and M. Erez. Virtualized and Flexible ECC for Main Memory. In *Proceedings of ASPLOS*, 2010.
- [56] H. Zheng et al. Mini-Rank: Adaptive DRAM Architecture For Improving Memory Power Efficiency. In *Proceedings of MICRO*, 2008.