Adaptive Reliability Chipkill Correct (ARCC)

Xun Jian
University of Illinois at Urbana Champaign
xunjian1@illinois.edu

Rakesh Kumar
University of Illinois at Urbana Champaign
rakeshk@illinois.edu

Abstract

Chipkill correct is an advanced type of error correction in memory that is popular among servers. Large field studies of memories have shown that chipkill correct reduces uncorrectable error rate by 4X [11] to 36X [12] compared to SECDED. Currently, there is a strong trade-off between power and reliability among different chipkill correct solutions. For example, commercially available chipkill correct solutions that can detect up to two failed devices and correct one (eg. SCCDCD) or two (eg. Double Chip Sparing) failed devices require accessing 36 DRAM devices per memory request. However, a weaker single chipkill correct single chipkill detect solution only requires accessing 18 devices per memory request and, therefore consumes much lower memory power. In this paper, we present Adaptive Reliability Chipkill Correct (ARCC) - an optimization to be applied to existing chipkill correct solutions to allow them to incur the low power consumption of a lower strength chipkill correct solution while maintaining similar reliability as that of a stronger chipkill correct solution. ARCC is based on the observation that, on average, only a tiny fraction of memory experiences any type of faults during the typical operational lifespan of a server. As such, it proposes relaxing the strength of chipkill correct in the beginning and then adaptively increasing the strength as needed on a page by page basis in order to reap the benefit of lower power consumption during the majority of the lifetime of a memory system. Our evaluation shows that ARCC reduces the power consumption of memory by 36%, on average, when applied to commercial SCCDCD, while keeping the storage overhead the same and maintaining similar reliability.

1 Introduction

Chipkill correct is an advanced type of error correction in memory that significantly improves the reliability of memory by allowing continued memory operation in the event of device-level faults in memory. Large scale studies show that chipkill correct reduces the Detectable Uncorrectable Error (DUE) rate of memory by 4X [11] to 36X [12] compared to Single Error Correction, Double Error Detection (SECDED). As a result, chipkill correct memory systems have become very popular among HPC systems and high end servers with large memory capacities. As the amount of memory in servers continue to increase, we envision that the adoption of chipkill correct memory systems will become even more widespread in order to maintain the same level of DUE and silent data corruption (SDC) rates in memory.

Currently, there is a strong trade-off between power and reliability among different chipkill correct solutions. Commercial chipkill correct solutions, such as single chipkill correct double chipkill detect (SCCDCD) [1] and Double Chip Sparing [5, 8], can detect up to two failed DRAM devices per rank; however, they require accessing 36 DRAM devices per memory request. On the other hand, a weaker solution that can only detect and correct up to a single failed device requires accessing only 18 DRAM devices (see Section 2); since only half as many devices are accessed per request, significant memory power can be saved. Similar trend exists for newly proposed chipkill correct solutions such as LOT-ECC [13] and VECC [15].

In this paper, we aim to improve on the present power and reliability trade-off of chipkill correct memory solutions. We observe that all existing chipkill correct solutions have a fixed level of protection strength from the start regardless of the age of the memory system; however, due to the low occurrence rate of faults in modern DRAM devices, our calculations, based on a large scale field study of over 160,000 DIMMs [12], show that on average only a tiny fraction of memory experience any type of faults in a typical operational lifespan of 5 to 7 years [9]. Therefore instead of a fixed worst case design, we propose an average case design where the memory system begins with low strength of protection, which consumes low power, and only upgrades to higher strength(s) of protection, which consumes high power, on a page by page basis as the pages become affected by faults. We call this optimization to be applied to chipkill correct solutions *Adaptive Reliability Chipkill Correct (ARCC)*. By increasing the chipkill correct strength of faulty pages at the end of every memory scrub, which can be

performed once every few hours [12], ARCC offers similar reliability as always using a strong chipkill correct solution for all the pages (see Section 6).

In this paper, we focus on applying ARCC to commercial chipkill correct solutions. Our evaluation shows that ARCC reduces the power consumption of memory by 36% when applied to commercially available chipkill correct solutions, while keeping the same storage overhead and maintaining similar reliability. We will also briefly describe how to apply ARCC to newly proposed chipkill correct solutions such as LOT-ECC and VECC.

We make the following contributions in the paper:

1. The concept of applying a weaker but more energy efficient ECC for regions in the main memory that are fault free, and dynamically increasing ECC strength of a region in the main memory after detecting faults in the memory region.

2. An efficient implementation of the concept, where adjacent smaller codewords combine to form larger codewords after faults are detected in a page, which allows ECC strength to be increased without increasing the storage overhead.

3. Comparative evaluation of the power, performance, and reliability of the above implementation relative to commercial chipkill correct solutions. Our experiments show that ARCC reduces memory power by 36% when applied to commercial chipkill correct solutions with negligible degradation to reliability.

The rest of the paper is organized as follows. Section 2 describes the relevant background and related work. Section 3 provides the motivational data for ARCC. Section 4 describes the application of ARCC to commercial chipkill correct solutions in detail. Section 5 describes the application of ARCC in other contexts. Section 6 considers the reliability implications of applying ARCC to commercial chipkill correct solutions. Section 7 describes the experimental set-up used in the results section, Section 8 presents the experimental results. Finally, Section 9 concludes the paper.

2 Background and Related Work

2.1 Commercial Chipkill Correct Solutions

Chipkill correct memory systems are designed to guarantee error correction and detection even in the event of a complete device failure in a rank, which is a group of DRAM devices needed to serve a single memory request. SCCDCD [1] and Double Chip Sparing [5, 8] are two popular commercial chipkill correct solutions. SCCDCD can correct one failed device and detect up to two failed devices. Double Chip Sparing can correct up to two failed devices as long as the two faults don't occur before one of them is first detected. Both solutions rely on symbol-based linear block codes to perform error detection and correction. In a symbol-based linear block code, each codeword is composed of multiple symbols, which are groups of bits; the symbols are categorized into data symbols and check symbols, which are the redundant information needed for error detection and correction [15]. The larger the number of check symbols per codeword, the higher the strength of error detection and correction. For example, with 2 check symbols per codeword, a bad symbol in the codeword can be detected and corrected [3]; however, when there are only 2 check symbols per codeword, if 2 bad symbols exist in the same codeword, the error can go undetected. With 4 check symbols per codeword, depending on the exact type of code employed, the second bad symbol in the codeword can either be detected or even be corrected [15].

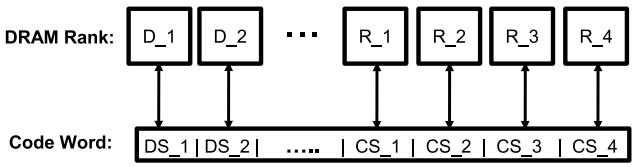

Figure 1. In commercial chipkill correct solutions, each symbol is stored in a different device in the rank. The "D" boxes represent data devices while the "R" boxes represent redundant devices.

Figure 1 illustrates how commercially available chipkill correct solutions store the symbols of each codeword. They store each symbol of a codeword in a different device in the rank. As a result, even in the event of a complete device failure, only a single symbol is lost per codeword; the lost symbol can be recovered using the remaining data symbols and check symbols in the codeword. Both SCCDCD and Double Chip Sparing use 4 check symbols per codeword. Although only 3 check symbols are required to provide single symbol correct and double symbol detect, SCCDCD uses a somewhat inefficient encoding such that all 4 check symbols are needed to provide the same level of protection [15]. On the other hand, Double Chip Sparing uses a more efficient encoding where only 3 check symbols are required to provide single symbol correct and double symbol detect. When a bad symbol is detected, the bad symbol is remapped to a spare symbol, the 4^{th} symbol. This allows Double Chip Sparing to correct up two bad symbols per codeword, as long as the second bad symbol does not occur before the first has been detected.

Since each symbol in a codeword has to be stored in its own DRAM device as illustrated by Figure 1, there has to

be as many redundant devices as there are check symbols per codeword. In order to keep the ratio of the number of redundant devices to regular data devices in a rank low, the number of data devices in the rank is chosen to be large. To keep the storage overhead the same as SECDED, commercially available chipkill correct solutions use 32 data symbols and 4 data symbols per codeword, resulting in a storage overhead of 12.5%; this translates to a rank with a total of 36 devices. Since such a large number of devices (36 compared to only 9 for SECDED) have to be accessed per memory request, commercial chipkill correct solutions consume high power.

2.2 Recently Proposed Chipkill Correct Solutions

Other chipkill correct solutions have been recently proposed that reduces the rank size via various trade-offs. For example, VECC [15] reduces the rank size of chipkill correct memory systems from 36 to 18 by reducing the number of data symbol per codeword from 32 to 16 and thus increasing the storage overhead beyond 12.5%. Since commodity ECC DIMMs with a rank size of 18 only support 16 data devices and 2 redundant devices, VECC stores the 2 check symbols needed for error detection in the redundant devices and maps the remaining check symbol(s) needed for error correction to the data devices in a different rank via the page table. For read accesses that do not contain errors, only 18 devices are accessed under VECC. For read accesses that contain errors, 36 device-accesses are required since the memory needs to be accessed a second time to retrieve the corresponding error correction check symbols if the check symbols are not already in the Last Level Cache (LLC). Similarly, for write requests, 36 device-accesses are required to update the error correction check symbols stored in memory if they are not found in the LLC.

Instead of relying a single code, LOT-ECC uses a combination of different codes for error detection and correction. For every line in memory, the data stored in each device is protected by a one's complement checksum stored in the same device for error detection and localization. In addition, the memory also stores the xor value of the data stored in each device. When an error is detected and localized to a single device, the lost data in the device can be reconstructed using the xor value. LOT-ECC reduces the number of devices required per rank down to 9. However, compared to commercial chipkill correct, LOT-ECC also requires a set of trade-offs. For example, the storage overhead of LOT-ECC is increased from 12.5% to 26.5%. In addition, LOT-ECC requires additional write accesses to memory for each write to memory to update the error correction resources. Since roughly 80% of the writes, on average, require additional writes to memory [13], this can lead to significant reduction in bandwidth for write-intensive workloads. Finally, LOT-ECC also lowers the error detection/correction guarantee of chipkill correct compared to commercial chipkill correct solutions because the one's complement checksums used for error detection in LOT-ECC can only guarantee detection of device-level faults if the output from a device with a device-level fault are all 1's or 0's. Examples of device-level faults in DRAM devices that do not result in an all stuck-at-1 or 0 output include faulty row or column address decoders that lead to the wrong row or column being read out.

ARCC can also be applied to both VECC and LOT-ECC, as will be briefly explained in Section 5. However, due to the wide adoption of commercial chipkill correct solutions, we will focus our evaluation on the application of ARCC to commercial chipkill correct solutions.

2.3 Other Related Work

PAYG [9] is a PCM-based scheme that relies on the observation that only a small fraction of lines suffer from many failed bits in PCMs; as such, instead of statically allocating a large number of error correction pointers to every line, one can rely on a small global pool of error correction pointers that are dynamically allocated to bad lines as needed to achieve similar lifetime at only a fraction of the total storage overhead.

Our goal and implementation are different – ARCC reduces the overall power consumption of chipkill correct by adaptively increasing the ECC strength of bad pages by combining adjacent lines in those pages into larger and, therefore, more power hungry lines with stronger ECC protection, while keeping the lines in error-free pages small with weaker ECC protection.

3 Motivation

There is a fundamental trade-off between power and reliability in chipkill correct solutions, when the storage overhead is held constant. Increasing the amount of ECC bits per word improves the error correction/detection capability of the codeword; however, this increases the storage overhead of the ECC bits. In order to keep the storage overhead the same while increasing the amount of ECC bits per word, the number of data bits in the word has to be increased as well; this in turn increases memory power consumption since more devices have be to accessed per memory request. Consider for example a memory configuration consisting of a single channel with 2 ranks and 36 devices per rank. If we were to reduce the number of check symbols per codeword from 4 down to 2, the size of a rank can be reduced from 36 down to 18 without affect the storage overhead. Our motivational experiments using quad-core multiprogrammed

SPEC benchmarks show that having a rank size of 18 instead of 36 reduces memory power consumption by 36.7% on average. However, the downside of using only 2 redundant check symbols is that it only guarantees the detection of a single bad symbol per codeword; the reliability that it provides is significantly worse than using 4 redundant check symbols per codeword as does commercial chipkill correct solutions, which guarantee detection of up to 2 bad symbols per codeword. ARCC is an optimization that seeks to improve the power-reliability trade-off between a stronger chipkill correct solution with more ECC bits per codeword and a weaker chipkill correct solution with fewer ECC bits per codeword by offering similar reliability as the former while consuming similar power as the latter.

Intuitively, pages with a fault can benefit much from double symbol detection/correction per codeword, because if an additional bad symbol occurs in a codeword that already contains a bad symbol, the second bad symbol can still be detected/corrected via double symbol detection/correction. On the flip-side, if a page is completely fault free, the added value of double symbol detection/correction compared to only single symbol detection/correction is significantly smaller; for these pages, using 2 symbols per codeword might suffice. Our reliability analysis in Section 6 confirms this intuition; it shows that adaptively upgrading codewords from single symbol protection to double symbol protection as the codewords become affected by faults incurs negligible reliability degradation compared to always applying double symbol protection to every codeword.

Meanwhile, large field studies have shown that only 2.95% [12] to 8% [11] of DIMMs suffer any type of faults per year. Also, most of these faults affect a small fraction of the DIMM (such as the single-bit and row faults). By considering the different types of faults studied in [12] and making the worst case assumption that each type of device-level fault considered in the work results in every memory location under the device-level circuitry becoming corrupted, we calculated the average fraction of 4KB physical pages in a memory channel that contain one or more faulty locations. The channel consists of 2 ranks with 36 devices per rank. Figure 2 shows that the fraction of pages with fault is just a few percent during most of the lifetime of the memory channel, even for a worst case failure rate that is 4X as high as what was measured in [12]. Since fault free pages can be protected using only 2 symbols per codeword instead of 4 and since most pages are fault free, an adaptive approach that provides weaker protection for a page in the beginning and only upgrades the protection strength when the page contains a fault can lead to substantial power savings with similar reliability as always providing stronger protection.

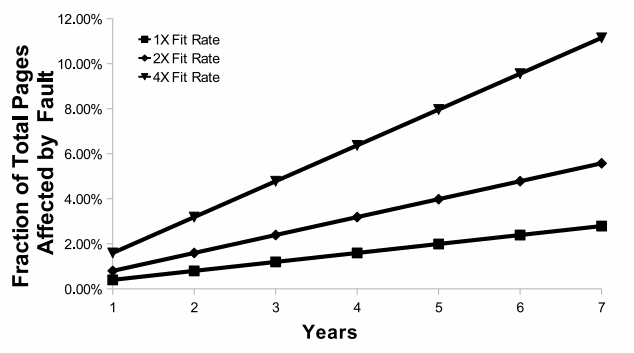

Figure 2. Average fraction of 4KB pages in a memory channel that has been affected by faults for different operational lifespans.

4 Adaptive Reliability Chipkill Correct

ARCC reactively increases the strength of protection of every codeword in a page when the page is detected with a fault by doubling the number of check symbols per codeword. Conceptually, ARCC does so by joining two codewords stored in two separate memory channels into a single large codeword, which, therefore, has twice the number of check symbols but the same storage overhead as the smaller codewords. In Section 4.1, we describe in detail how to apply ARCC to commercial chipkill correct solutions to provide similar reliability as always using 4 check symbols per codeword while incuring the low memory power consumption of using only 2 check symbols per codeword.

4.1 Applying ARCC to Commercial Chipkill Correct Solutions

We refer to a physical page where there are 4 check symbols per codeword as an *upgraded* page and a page where there are only 2 check symbols per codeword as a *relaxed* page. The top half of Figure 3 shows the data layout of a relaxed page. In the figure, there are 2 memory channels in the memory system, where each memory channel can serve a memory request for a 64B line independently. We consider a common memory configuration where each physical page contains 4KB of data, which is equivalent to 64 64B lines. Conventional physical address mapping policies (e.g. SDRAM_BASE_MAP, SDRAM_HIPERF_MAP, SDRAM_CLOSE_PAGE_MAP [14]) deployed in systems with multiple memory controllers map adjacent 64B lines to different memory channels in order to reduce the latency of accessing adjacent lines in memory; this is reflected in Figure 3, where alternate lines belong to alternate memory channels (X and Y). Each 64B line, in turn, consists of multiple codewords; in the example in Figure 3, each line consists of 4 codewords, which are delimited by the horizontal

lines in the figure; each codeword is composed of 16 data and 2 check symbols, which are represented by the shaded region. Since each symbol maps to a different DRAM device in commercial chipkill correct solutions, the 18 symbols of a codeword in a relaxed page are stored across 18 DRAM devices controlled by the same memory controller.

When an error is detected during memory scrubbing, ARCC increases the protection strength of the page with error by increasing the number of check symbols per codeword from 2 to 4. To double the number of check symbols per codeword without increasing the check to data symbol ratio which increases storage overhead, ARCC combines 2 adjacent 64B lines, each stored in separate channel, in a page into a single 128B line, referred to as an *upgraded line* from now on, where each codeword in the 128B line contains 4 check symbols and 32 data symbols. To convert a relaxed page into an upgraded page, all lines in the relaxed page are read out to compute the new codewords in each line and are stored back to memory afterwards. Note that since the two 64B lines in each upgraded line belong to 2 separate memory channels, the entire upgraded line can be read out in the time it takes to read a single 64B line by accessing the two memory channels in parallel. The bottom half of Figure 3 illustrates one way of combining 2 adjacent 64B lines into an upgraded line, where each symbol maintains its original size and the number of codewords per upgraded line is the same as the number of codewords per line under the relaxed mode. An alternative design is to reduce the size of each symbol by half and as a result, double the number of codewords per upgraded line. This flexibility is important since different symbol sizes require different types of error detection and correction (EDAC) controllers; by providing this flexibility, ARCC provides freedom in choosing the EDAC controller to use for the upgraded line.

4.2 Implementation Details

The following sections describe the different modifications needed to support ARCC as well as the associated overheads.

4.2.1 Page Table

Each physical page entry and the corresponding TLB entry is modified to contain an additional 1-bit flag to indicate the chipkill correct strength, relaxed or upgraded, the page currently operates in. The value of the flag is updated at the end of a memory scrub. If the memory scrubber detects an error in a physical page, the chipkill correct strength of the physical page is to be upgraded. To upgrade a page affected by faults, only the page itself needs to be accessed to recalculate each upgraded line in the page; pages without faults are not affected. When an upgraded physical page is accessed, both 64B lines in each upgraded line will be accessed.

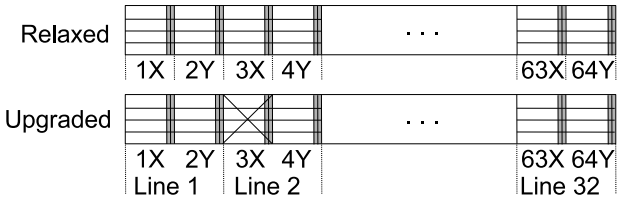

Figure 3. Data layout of a physical page in relaxed and upgraded chipkill correct modes. The letters 'X' and 'Y' appended after each line number indicate to which channel the line belongs. Each shaded rectangle represent a check symbol in a codeword. The line with the cross contains a fault, which causes the page to be in upgraded.

We assume that the OS is started up in the upgraded mode for every page. After the page table has been populated, a memory scrub is immediately performed to determine the fault-free pages to set them to relaxed mode.

4.2.2 Memory Scrubbing

ARCC upgrades the chipkill correct strength of a page after faults are detected in a page during memory scrubbing. Our reliability analysis in Section 6 assumes an ideal memory scrubber that is capable of detecting all faults at the end of each memory scrub. A conventional memory scrubber which simply reads out and writes back the memory content during each scrub may leave many hidden stuck-at-1 or stuck-at-0 faults undetected. Therefore, to adhere closely to such an ideal memory scrubber, we modified a conventional memory scrubber to execute the following steps:

1. Read a line and store its value aside.
2. Write all 0's to the line location in memory and then read the location in memory to see if only 0's are returned. If true, go to step 3. If false, a stuck-at-1 fault may be present; go to step 4 and upgrade the page afterwards.
3. Write all 1's to the line and then read the line to see if only 1's are returned. If true, go to step 4. If false, a stuck-at-0 fault may be present; go to step 4 and upgrade the page afterwards.
4. Correct any errors in the original content of the line and write the line back to memory.

Optionally, in order to reduce the overhead of alternating between reads and writes during a memory scrub, steps 1 to

4 can be performed in batches for multiple consecutive lines at a time.

Although memory scrubbing is twice more expensive in ARCC (due to the 2 additional reads and writes for all 0's and all 1's) compared to conventional memory scrubbing, the performance overhead of memory scrubbing is still negligible since memory scrubbing takes a few seconds per memory channel while it is performed once every few hours [12]. Consider, for example, a 128-bit wide memory channel with 4GB of 667MHz DDR2 memory. Accessing the entire memory content takes $4 \cdot 1024^3 \cdot 8/128/(667 \cdot 10^6) = 0.4s$. A memory scrub required by ARCC takes $0.4 \cdot 6 = 2.4s$ per memory scrub. Assuming a memory scrub rate of once 4 hours, $2.4s/(4 \cdot 3600)$ results in only 0.0167% reduction in maximum effective memory bandwidth.

4.2.3 Last Level Cache

The LLC needs to be modified in order to accommodate both the relaxed 64B lines and the upgraded 128B lines in the LLC simultaneously; during a write to memory, both sub-lines of an upgraded line need to be written back to memory at the same time in order to update all 4 check symbols in each codeword in the upgraded line. One way to accommodate both 64B and 128B lines in the LLC is to implement the LLC as a Sectored Cache [10]. However, since the Sectored Cache can degrade the effective size of the cache when there is low spatial locality in the applications, we propose an alternative LLC design to accommodate the two different cacheline sizes.

We observe that since the physical addresses of the two sub-lines in an upgraded line are consecutive, the two sub-lines will be mapped to 2 adjacent sets in a conventional LLC with 64B cachelines. We propose including an additional bit to the tag of each cacheline to indicate whether or not the cacheline belongs to an upgraded line. When an upgraded line is brought into the LLC, the flag is set to 1. When a line is selected for eviction, its flag is checked to see if it is a sub-line of an upgraded line; if it is, the 2^{nd} sub-line of the same upgraded line can be found in the adjacent set as the line with the same tag. In order to prevent a sub-line from being forcefully evicted due to the lack of reuse in the second sub-line, the LLC cache replacement policy uses the recency of the most recently used sub-line as the recency value of both sub-lines for eviction selection.

The main overhead in the cache is due to the fact that cache replacement requires a second tag access to find the recency value of the other sub-line in an upgraded line. The performance overhead of this second tag access is small because LLC replacements are required only by LLC misses, which are less frequent that LLC hits. In addition, the latency of the second tag access is much smaller than that of the memory access due to the LLC miss. In our experiments, we modified the cache implementation such that every cache replacement takes twice as long and did not observe any noticeable effect on performance.

4.2.4 Memory Controller

The two sub-lines in each upgraded line have to be read from and written to memory at the same time in order to provide error detection/correction. One design is to logically partition the memory queue of each controller into two, one for the sub-lines of upgraded lines and one for the relaxed lines. The sub-line queue maintains a strict FIFO ordering to ensure that the pairing of the sub-lines in each queue is always correct. The memory controllers can then alternatingly issue requests from the queue for the sub-lines and the queue for the regular 64B lines.

An alternative design is to augment each memory queue entry with a new flag with multiple bits. The first bit of the flag is set to 1 to indicate that the line is a sub-line of an upgraded line. When the first bit is set to 1, the remaining bits in the flag serve as a pointer to the physical queue entry in the other memory channel where the 2^{nd} sub-line resides. When a sub-line is at the head of the memory access queue, memory access in the queue is stalled until the 2^{nd} sub-line is found. The corresponding sub-line in the second memory channel is to be found via the pointer and then promoted to the head of its memory access queue so that the pair of sub-lines can be issued together.

Due to the large number of devices per rank (36), commercial chipkill correct solutions require 2 physical memory channels, each controlling 18 devices, to operate in lockstep as a single logical channel [13]. As a result, a single EDAC controller targeting codewords with 4 check symbols is sufficient for every pair of memory controllers. However, ARCC requires an additional EDAC controller for each memory controller to target codewords with 2 check symbols.

5 Applying ARCC to Other Contexts

ARCC is a versatile optimization that can be applied to different contexts to provide the reliability of a stronger chipkill correct solution while consuming the same memory power as a weaker chipkill correct solution. In the following sections, we briefly describe various other contexts where ARCC is applicable.

5.1 Enabling Stronger Forms of Chipkill Correct

ARCC can be used to support even stronger strengths of chipkill correct than what is commercially available without requiring the design of new ECC DIMMs. Consider,

for example, when ARCC is applied to Double Chip Sparing [4, 8], which can correct up to 2 bad symbols per codeword with 4 check symbols per codeword. When a codeword under the upgraded mode develops a second bad symbol, all the codewords in the page that contains the affected codeword can be further upgraded to an even stronger upgraded mode of having 8 check symbols per codeword by striping each codeword across 4 memory channels instead of just 2. This provides each codeword in the page with 4 additional spare symbols to remap bad symbols to. An alternative design is to divide the large codeword with 8 check symbols into 2 smaller codewords each with 4 check symbols and remap the the 2 bad symbols such that they are divided equally between the 2 smaller codewords, so that each smaller codeword can correct yet another bad symbol when it occurs in the future. Since only a small fraction of memory that are affected by one bad symbol is also affected by a second bad symbol, the number of pages in the second upgraded mode should be only a tiny fraction of the pages in the first upgraded mode. As a result, ARCC can provide multiple upgraded modes with similar power and performance characteristics as having just one upgraded mode.

5.2 Applying ARCC to VECC and LOT-ECC

ARCC can also be used to support recently proposed chipkill correct solutions such as LOT-ECC and VECC.

To provide double chipkill correct, VECC uses 4 check symbols per codeword [15]. Instead of always using 4 check symbols per codeword, ARCC can be applied to VECC to reduce the number of check symbols per codeword in a page to 2 when the page is fault free. As a result, ARCC can reduce the number of devices per rank from 18 (see Section 2) to 9 (8 data devices and 1 redundant device for the 1 detection check symbol) while maintaining the same storage overhead as VECC to reduce power consumption.

The LOT-ECC configuration described in [13] that uses 9 devices per rank is only capable of correcting a single bad symbol, as does SCCDCD, but not two bad symbols, as does Double Chip Sparing. However, we observe that LOT-ECC can be extended to provide Double Chip Sparing by using 18 devices per rank without increasing the storage overhead. In the 18-device configuration, the data of a line is stored evenly across 16 devices, while the 17^{th} device stores the xor value of the symbols in the 16 data devices and the 18^{th} device is the spare device to which the correct value of the bad symbol is remapped. The one's complement checksums needed for error detection and localization (see Section 2) are to be stored in a different line in the same row. However, the 18-device configuration incurs high power overhead in two ways. First, it requires accessing twice as many devices per memory request as the 9-device configuration. Second, due to the fact that the one's complement checksums needed for error detection are now stored in a different line than the data, the 18-device ARCC also requires an additional read access to the checksum line for every read to memory in addition to the additional write access per memory writeback (see Section 2). ARCC can be used to dynamically convert a relaxed page that uses 9-device LOT-ECC to an upgraded page that uses 18-device LOT-ECC when errors are detected in the page. By applying ARCC to LOT-ECC to provide Double Chip Sparing, ARCC can reduce the DUE rate of LOT-ECC by 17X [4] at a small power overhead (see Section 8), since only a small fraction of pages, on average, have faults in memory and, therefore, need to operate in the upgraded mode.

6 Reliability Analysis

In this section, we explore the reliability impact of applying ARCC to commercial chipkill correct solutions.

6.1 Impact on DUE Rate

ARCC does not degrade the DUE rate of commercial chipkill correct solutions. When ARCC is applied to SCCDCD, ARCC does not degrade the DUE rate since ARCC always guarantees correction of a single bad symbol in a codeword, just as SCCDCD. ARCC also does not degrade the DUE rate when it is applied to Double Chip Sparing, which corrects up to two bad symbols per codeword. This is because just like ARCC, Double Chip Sparing cannot correct the second bad symbol unless the first bad symbol has been detected before the second bad symbol occurs (see Section 2).

6.2 Impact on SDC Rate

In the ARCC implementation described in Section 4.1, each codeword contains only 2 check symbols at the beginning, which only guarantees the detection of a single bad symbol. After an error is detected in a page (which results in at most one bad symbol per codeword), ARCC reactively increases the number of check symbols per codeword to 4 by doubling the size of each codeword in the page in order to be able to detect 2 bad symbols per codeword. However, it is possible for a second bad symbol to occur in the same codeword before the first bad symbol is detected; such errors cannot be detected. On the other hand, commercial chipkill correct solutions constantly allocate 4 check symbols per codeword, and, therefore, always guarantee the detection of 2 bad symbols. However, the probability of 2 or more faults in 2 or more *different* devices affecting the *same*

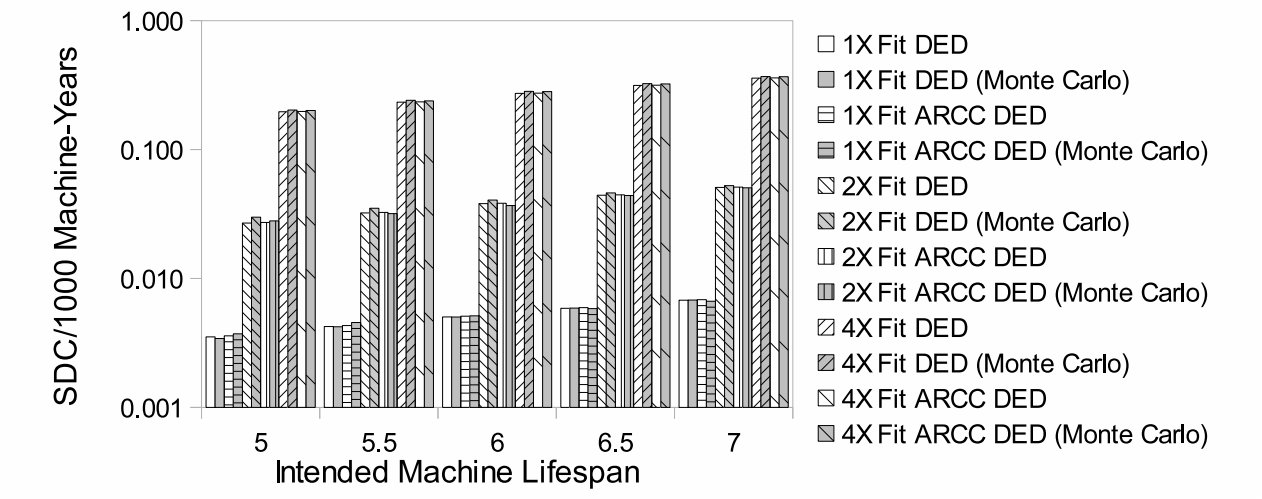

Figure 4. Comparison between the SDC rate of simultaneous double error detection (DED) as provided by commercial SCCDCD and that of the reduced double error detection (ARCC DED) as provided by applying ARCC to commercial SCCDCD.

codeword *and* occurring within the *same* scrub period of a few hours is small.

To understand the extent of degradation in detection reliability when applying ARCC to commercial chipkill, we used the chipkill correct reliability models given in [6]. The fault rates that were used as inputs to the models are taken from a recent large field study on DRAM errors [12], and include those of lane, device, bank, column, and row faults. A memory scrub period of 4 hours was assumed, which is consistent with the memory scrub period used in [12]. The memory configuration of the baseline commercial chipkill correct solution consists of a memory channel with 2 ranks, with 36 devices per rank, and, therefore, a total of 72 DRAM devices. We used the double chipkill correct/detect model provided in our technical report [6] to calculate the error detection reliability of the baseline SCCDCD. Since SCCDCD+ARCC cannot detect a second bad symbol in a codeword unless it occurs after the first bad symbol in the codeword has been detected, its error detection reliabililty is the same as that of the error correction reliability of Double Chip Sparing. Therefore, we used the Double Chip Sparing reliability model in [6] to calculate the error detection reliability of SCCDCD+ARCC. To report the SDC rate, we converted the reliability output to the SDC rate. When calculating the SDC rate, we assume that all DIMMs in a machine are to be replaced as soon as the first undetectable error occurs in the machine, so that the same faulty machine does not contribute multiple silent data corruptions. To validate the results of the reliability models, we also performed Monte Carlo simulations (detains in [6]). Figure 4 shows the number of SDCs in 1000 machine-years calculated using the output from the reliability model for both the SCCDCD baseline and SCCDCD+ARCC for different intended lifespans of a machine and for different factors of the failure rate. The figure shows that the increase to SDC rate of SCCDCD+ARCC over SCCDCD alone is insignificant.

Table 1. Memory Configurations

Name	Tech	I/O	Chan	Ranks/Chan	Rank Size
Baseline	DDR2	X4	2	1	36
ARCC	DDR2	X8	2	2	18

Table 2. Processor Microarchitecture

SS Width 2	IQ Size 16	Phys Regs 72FP/72INT	LSQ Size 32LQ/32SQ
L1 D$, I$ 32 kB	L1 Assoc 2	L1 lat. 1 cycle	L2$ 1MB
L2 Assoc 16	L2 lat. 10 cycles	Cacheline Size 64B	L2 MSHR 240

7 Methodology

Table 1 summarizes the memory configurations for comparison against commercial chipkill correct solutions. The commercial chipkill correct baseline simulated consists of a single memory channel with 2 ranks per channel and 36 devices per rank. Due to the large number of devices per rank, both the burst length and the I/O width of each device must be small to satisfy the chosen cacheline size of 64B. As such, DDR2 X4 devices are chosen for the SCCDCD baseline. The corresponding memory configuration for ARCC consists of 2 memory channels with 2 ranks

Table 3. Workloads simulated

Mix1	mesa;leslie3d;GemsFDTD;fma3d
Mix2	omnetpp;soplex;apsi;mesa
Mix3	sphinx3;calculix;omnetpp;wupwise
Mix4	lucas;gromacs;swim;fma3di
Mix5	mesa;swim;apsi;sphinx3
Mix6	sjeng;swim;facerec;ammp
Mix7	milc;GemsFDTD;leslie3d;omnetpp
Mix8	facerec;leslie3d;ammp;mgrid
Mix9	applu;soplex;mcf2006;GemsFDTD
Mix10	mcf2006;libquantum;omnetpp;astar
Mix11	calculix;swim;art110;omnetpp
Mix12	lbm;facerec;h264ref;ammp

Table 4. Fraction of pages upgraded for each type of device-level fault.

Fault Type	Fraction of Pages Upgraded
Lane	100%: It causes both ranks per channel to be upgraded.
Device	1/2: It causes 1 out the 2 ranks to be upgraded.
Subbank	1/16: It causes 1 out of the 8 banks in a single rank to be upgraded.
Column	1/32: It causes half of the pages in a single bank to be upgraded.

per channel and 18 devices per rank. Both configurations have the same total number of devices. In order to provide the same output granularity with 18 devices per rank as that of the baseline with 36 devices per rank, we increased I/O width of each device from X4 to X8 for ARCC. DRAMsim [14] was used to model memory power and timing. The DRAM device parameters are taken from Micron datasheets [7]. We assume that there are 2 4KB pages per row in memory. For the row buffer and physical address mapping policies, we used the closed page policy and the high performance mapping policy, respectively, as provided by DRAMsim. Meanwhile, we simulated a quad-core processor running 12 mixed SPEC workloads for 2 billion cycles using M5 [2], a full system simulator. Table 2 describes the CPU microarchitecture while Table 3 shows the workloads that were used.

We used the following methodology to study the power and performance degradation due to upgraded pages in ARCC as faults develop over time:

1. We estimated the power and performance overhead associated with each type of device-level fault, introduced in Section 6 for the memory configuration summarized in Table 1 by setting the fraction of memory affected by that type of fault to upgraded mode and repeating the experiments. Table 4 lists the fraction of pages upgraded for each type of fault.

2. Using Monte Carlo simulations, we simulate a 7 year lifespan for 10000 memory channels to capture when and what type of device-level faults occur during the 7 simulated years in the 10000 channels. The fault rates from [12] were used.

3. For each recorded fault type in each simulated memory channel, we added the overhead associated with that fault to the performance and power of that channel, starting from the time that the fault occurs.

4. For each year X in the intended lifetime of a channel, we averaged the power and performance of the 10000 memory channels from the beginning of the first year to the end of year X to provide an estimate for the overall power and performance of ARCC in the presence of faults.

8 Results

When ARCC is applied to commercial chipkill correct solutions, which require 36 devices to be accessed per memory request, significant power reduction can be achieved as ARCC requires only 18 devices to be accessed for pages that are fault-free and 36 devices to be accessed for pages with faults (see Table 1). Figure 5 shows the DRAM power and performance improvement from applying ARCC to commercial chipkill correct solutions when there are no faults in memory. Performance of a mixed workload is reported as the sum of the IPCs of all the benchmarks in the workload. On average, ARCC reduces power consumption by 36.7% and improves performance by 5.9%. The power benefit across the workloads are relatively uniform due to the fact that ARCC saves the same amount of power every memory access regardless of the application characteristics. The slight performance improvement is due to having twice the number of ranks under ARCC, which increases the amount of rank-level parallelism. Different benchmarks experience different amount of benefits from the increased rank-level parallelism, which explains the variation in performance.

Figure 6 shows the power consumption of ARCC in the presence of a single device-level fault in memory normal-

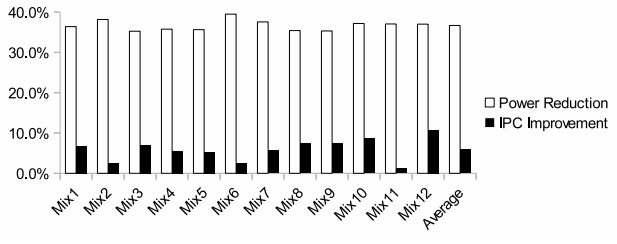

Figure 5. Power and Performance Improvement from applying ARCC.

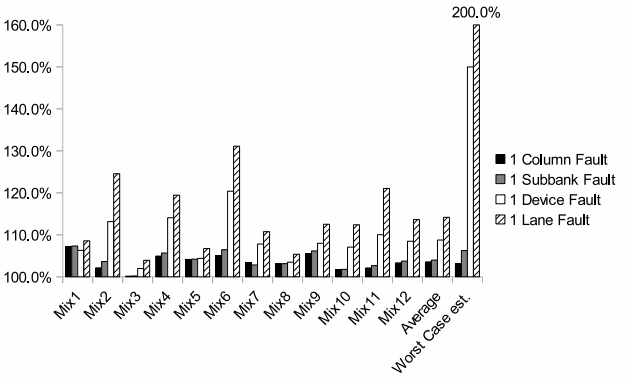

Figure 6. Power consumption of ARCC when there are different types of faults in memory, normalized to when there is no fault.

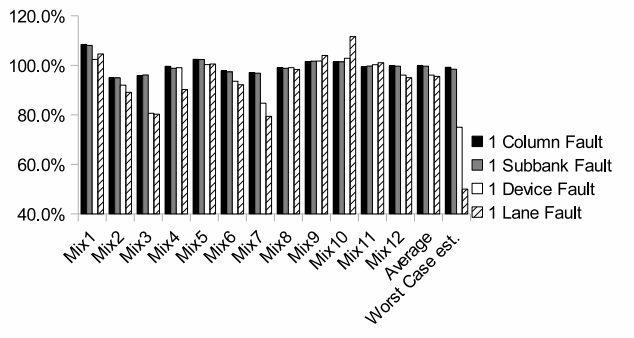

Figure 7. Performance of ARCC when there are different types of faults in memory, normalized to when there is no fault.

ized to when the memory is fault free. Results are presented for different types of device-level faults. For example, "1 Device Fault" represents the scenario where half of the pages have been upgraded due to a device fault (Table 4). As expected, the power consumption of ARCC increases in the presence of faults in memory. This is because 2 adjacent 64B lines instead of 1 are required to be accessed when a 128B line in an upgraded page is accessed. In the worst case scenario, when there is no spatial locality in the application, the second 64B line is always not useful to the workload. In this scenario, the power consumption of accessing an upgraded page is twice that of accessing a normal page. As shown in the "Worst Case est." bar in the figure, the worst case memory power increases by the fraction of pages in memory that are upgraded. In reality, the second 64B line is useful for many workloads due to the presence of spatial locality; as a result, the power overhead due to device-level faults in memory is much smaller as shown in the figure.

Figure 7 shows the IPC of ARCC in the presence of a single device-level fault in memory normalized to when the memory is free of faults. While some workloads, such as *Mix2* to *Mix7*, show a clear degradation in performance when there are faults in memory, some others such as *Mix1* and *Mix10* show improvement in performance. We attribute this to the different amount of spatial locality in the different workloads. Although applications with low spatial locality suffers when 2 adjacent cachelines have to be accessed for every memory access after upgrading faulty pages, applications with high spatial locality on the other hand actually benefits from having to access 2 adjacent cachelines because this acts like an useful prefetch. In the worst case scenario, when there is no spatial locality in the application and bandwidth is the bottleneck, ARCC can degrade performance by as much as 50% in the presence of a lane fault.

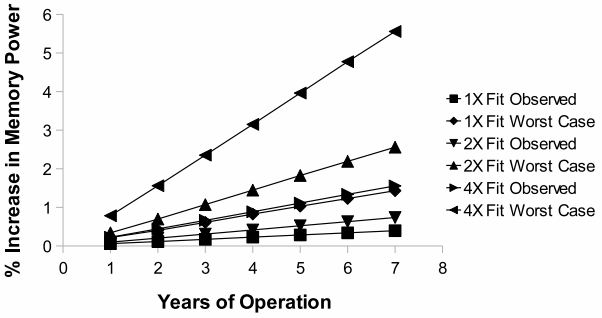

Figure 8. Average increase in power consumption as a function of time compared to fault-free memory.

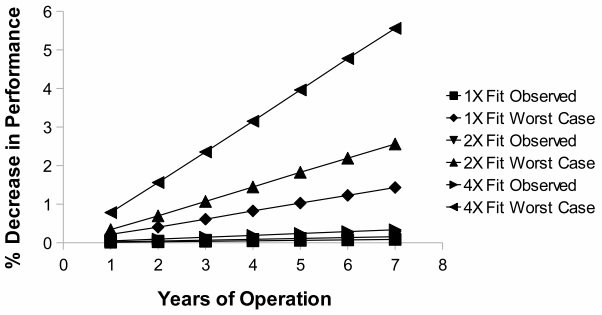

Figure 9. Average decrease in performance as a function of time compared to fault-free memory.

However, due to spatial locality in the applications studied, there was negligible performance degradation on average.

The fraction of pages that are faulty in a memory system increases with time. Using the methodology described in Section 7, we calculated the average power and performance degradation during different years in the lifetime of the memory system in Table 1. The results are shown in Figures 8 and 9 respectively. The worst case estimate curves in the figures assume that there is no spatial locality in the application and therefore a memory access to an upgraded pages consumes twice as much power and reduces effective bandwidth by half compared to a memory access to a relaxed page. The figures show that the degradation both in terms of the worse case estimate and measured overheads is small. This is due to the fact that, on average, only a tiny fraction of memory are affected by faults (see Figure 2). In fact, power benefits from ARCC even at the end of 7 years for 4X the memory fault rate reported in [12] is no less than 30%.

8.1 Applying ARCC to LOT-ECC

As described in Section 5, ARCC can be applied to LOT-ECC to enable Double Chip Sparing by converting a page from the 9-device LOT-ECC to 18 device LOT-ECC, again by requiring two channels, each channel with 9 devices per rank, to operate in lockstep during memory accesses to the upgraded pages. Compared to applying ARCC to commercial chipkill correct solutions, applying ARCC to LOT-ECC not only incurs the power overhead of accessing twice as many devices per memory request, but also requires an additional read access for every regular read access. Therefore, in the worst case scenario, where an application consists of 100% read accesses and has no spatial locality, a memory access to an upgraded page is equivalent to 4 memory accesses to a relaxed page. As a result, the power of memory accesses to upgraded pages can differ by a factor of 4 compared to the power of memory accesses to relaxed pages. Similarly, the effective bandwidth for memory accesses to the upgraded pages is also reduced by a factor of 4. Figure 10 shows the average power increase and performance degradation due to faults during different years in the lifetime of the memory system of ARCC + LOT-ECC compared to that of the regular 9-device LOT-ECC for the worst case application scenario. The figure shows that for the memory fault rate reported in [12], the average power increase/performance degradation during the 7 year period is only 1.6%. This is a small cost for reducing the DUE rate by 17X by providing Double Chip Sparing [4]. Even for a fault rate 4X as high as that of the memory fault rate reported in [12], the average power increase and performance degradation are expected to be no more than 6.3%, on average.

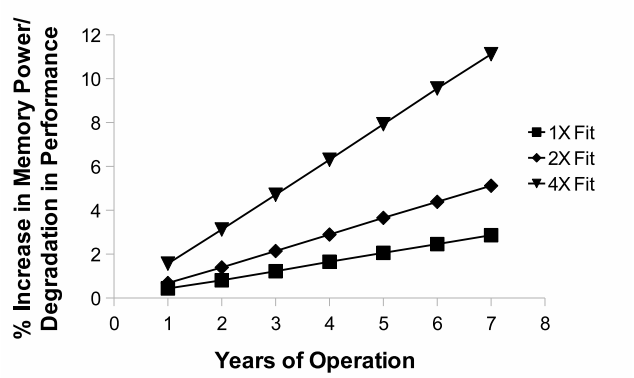

Figure 10. Average increase in power and decrease in performance of ARCC+LOT-ECC vs 9-device LOT-ECC for the worst case application scenario as a function of time.

9 Conclusions

In this paper, we propose ARCC, a novel optimization for existing chipkill correct solutions that aims to provide the reliability of a high strength of chipkill correct solution at the same memory power overhead of a low strength chipkill correct solution without increasing the storage overhead. Based on the observation that only a small fraction of memory experiences faults in memory for a typical operational lifespan of a memory system, ARCC begins with a low chipkill correct strength and adaptively increases the chipkill correct strength on a page by page basis as they become affected by faults, in order to take advantage of the power benefit of weaker chipkill correct solution for all fault-free pages while providing the reliability of a stronger chipkill correct solution. We presented an efficient implementation of the concept, where the number of check symbols per codeword is doubled after faults are detected in a page by combining two adjacent codewords in two different channels into a single large codeword without increasing the overall storage overhead. We performed a comparative evaluation of the power, performance, and reliability of this implementation relative to commercial chipkill correct solutions. Our experiments show that this implementation reduces memory power by 36% when applied to commercial chipkill correct solutions with negligible degradation to reliability. ARCC not only can be used to reduce the power consumption of existing commercial chipkill correct memories with 4 check symbols per codeword, but also provides an implementation for stronger chipkill correct solutions in the future, such as those with 8 check symbols per codeword, with only small increase to memory power consumption.

10 Acknowledgements

This work was supported in part by NSF and Oracle. We thank Vilas Sridharan, Nathan Debaraladeen, Sean Blanchard, and anonymous reviewers for their helpful feedback.

References

[1] AMD. BIOS and Kernel Developer's Guide for AMD Family 15h Models 00h-0Fh Processors. 2012.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhard. The M5 Simulator: Modeling Networked Systems. *IEEE Micro*, 2006.

[3] C. Chen and M. Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM Journal of Research and Development*, 1984.

[4] HP. RAS Features of the Mission-Critical Converged Infrastructure. 2010.

[5] Intel. RAS Features of the Mission-Critical Converged Infrastructure. 2010.

[6] X. Jian, S. Blanchard, N. Debardeleben, V. Sridharan, and R. Kumar. *Reliability Models for Double Chipkill Detect/Correct Memory Systems (UILU-ENG-13-2201)*. http://passat.crhc.illinois.edu/rakeshk/chipkillModel.pdf, 2013.

[7] MICRON. *512Mb: x4, x8, x16 DDR2 SDRAM - Micro*. http://download.micron.com/pdf/datasheets/dram/ddr2/512MbDDR2.pdf.

[8] M. Ohmacht, R. A. Bergamaschi, and S. Bhattacharya. Blue Gene/L Compute Chip: Memory and Ethernet subsystem. 2005.

[9] M. K. Qureshi. Pay-As-You-Go: Low-Overhead Hard-Error Correction for Phase Change Memories. *Micro*, 2011.

[10] J. B. Rothman and A. J. Smith. Sector Cache Design and Performance. *University of California, Berkeley, Technical Report*, 1999.

[11] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM Errors in the Wild: a Large-Scale Field Study. *SIGMETRICS*, 2009.

[12] V. Sridharan and D. Liberty. Field Study of DRAM Errors. *SELSE*, 2012.

[13] A. Udipi, N. Muralimanohar, R. Balsubramonian, A. Davis, and N. Jouppi. LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems. *ISCA*, 2012.

[14] University of Maryland. University of Maryland Memory System Simulator Manual.

[15] D. H. Yoon and M. Erez. Virtualized ECC: Flexible Reliability in Main Memory. *Micro*, 2010.