LOT-ECC: LOcalized and Tiered Reliability Mechanisms for Commodity Memory Systems *

Aniruddha N. Udipi† Naveen Muralimanohar‡ Rajeev Balasubramonian†
Al Davis† Norman P. Jouppi‡

†University of Utah
{udipi, rajeev, ald}@cs.utah.edu

‡HP Labs
{naveen.muralimanohar, norm.jouppi}@hp.com

Abstract

Memory system reliability is a serious and growing concern in modern servers. Existing chipkill-level memory protection mechanisms suffer from several drawbacks. They activate a large number of chips on every memory access – this increases energy consumption, and reduces performance due to the reduction in rank-level parallelism. Additionally, they increase access granularity, resulting in wasted bandwidth in the absence of sufficient access locality. They also restrict systems to use narrow-I/O x4 devices, which are known to be less energy-efficient than the wider x8 DRAM devices. In this paper, we present LOT-ECC, a localized and multi-tiered protection scheme that attempts to solve these problems. We separate error detection and error correction functionality, and employ simple checksum and parity codes effectively to provide strong fault-tolerance, while simultaneously simplifying implementation. Data and codes are localized to the same DRAM row to improve access efficiency. We use system firmware to store correction codes in DRAM data memory and modify the memory controller to handle data mapping. We thus build an effective fault-tolerance mechanism that provides strong reliability guarantees, activates as few chips as possible (reducing power consumption by up to 44.8% and reducing latency by up to 46.9%), and reduces circuit complexity, all while working with commodity DRAMs and operating systems. Finally, we propose the novel concept of a heterogeneous DIMM that enables the extension of LOT-ECC to x16 and wider DRAM parts.

1 Introduction

With shrinking feature sizes and increasing capacity, memory system reliability is a growing concern. In some datacenter settings, it has been suggested that storage be built entirely out of DRAM memory, and large-scale systems be created by aggregating the main memory of thousands of servers [26]. This places greater pressure on the memory system to not yield errors. High-availability servers are typically expected to provide chipkill-level reliability – the ability to withstand the failure of an entire DRAM chip. Current commercial chipkill-level reliability mechanisms [10] are based on conventional symbol-based ECC codes [14]. They impose various restrictions on system configuration, and suffer from some combination of the following problems: *(i)* wasted energy from activation overfetch, *(ii)* wasted memory bus bandwidth and energy due to forced prefetching, *(iii)* reduced energy-efficiency due to the forced use of narrow I/O DRAMs, *(iv)* lost opportunities to increase rank-level parallelism, *(v)* increased storage overheads, and *(vi)* increased circuit complexity. Recent attempts to improve chipkill design [34, 36] have only partly addressed these problems. There is thus clearly a need for a fundamentally different approach to provide efficient chipkill-level fault-tolerance.

Given the memory industry's hard constraints with respect to commodity parts in servers, it is essential to stay fully standards-compliant to even be considered for commercial adoption. We allow ourselves no leeway to modify the design of DRAM chips, DIMMs, or the JEDEC protocol, including burst length, access granularity, etc. Similarly, the fault-tolerance mechanism should be completely transparent to the cache hierarchy, OS, and applications in order to be implementation friendly. The only permitted changes are in the memory controller and system firmware [16]. Also, memory capacity by itself is affordable, as long as commodity components are used, and some of it can be traded off for RAS benefits, rather than changing standards or modifying a multitude of system components [16].

With these requirements and constraints in mind, we present LOT-ECC, a novel chipkill-level memory reliability mechanism. LOT-ECC employs multiple levels of localized and tiered error protection. Simple checksum and parity codes are deployed effectively to provide strong levels of fault tolerance while activating the smallest possible number of chips per memory access.

*This work was supported in parts by NSF grants CCF-0811249, CCF-0916436, NSF CAREER award CCF-0545959, HP, and the University of Utah.

All the requisite data mapping and verification is handled by the memory controller with minimal help from the system firmware, while being completely transparent to the DRAM devices, OS, caches, TLB, and other system components. Compared to a modern commercial chipkill implementation, LOT-ECC saves up to 44.8% memory power (static + dynamic) and reduces average memory access latency by up to 46.9%, in addition to a reduction in ECC circuit complexity, all for a small 14% additional storage overhead.

2 Background and Related Work

2.1 Existing Commercial Solutions

Current commercial chipkill solutions employ Single Symbol Correct Double Symbol Detect (SSC-DSD) codes [10, 14, 36], which operate on a set of bits (a "symbol") rather than individual bits. All errors, of all lengths, within a single symbol can be corrected. There are two popular SSC-DSD codes, the eponymous 3-check-symbol and 4-check-symbol codes [36].

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

The x4 chip, 4-check symbol code based designs suffer from several drawbacks, as described below, and summarized in Table 1. First, ECC codes are computed over large 144-bit data words. This activates a larger number of chips than absolutely required, increasing overfetch within DRAM chips [15, 20, 32, 34], and resulting in substantially increased energy consumption. Area, density, and cost constraints make overfetch inevitable to some extent within a rank of chips, but imposing additional inefficiency in order to provide fault tolerance should be avoided. Second, the wide-word requirement results in increased access granularity as burst lengths increase – a 144-bit bus with the standard DDR3 burst length of 8 already reads/writes two 64-byte cache lines per access. This forced prefetch potentially wastes bandwidth and energy unless access locality is consistently high. Third, since a large number of chips is made busy on every access, there are fewer opportunities for rank-level parallelism within a given amount of memory, potentially hurting performance. Bank contention will likely emerge as a major bottleneck if novel interconnect technologies such as silicon photonics [11, 33] substantially increase the available off-chip memory bandwidth, making parallelism more important. All of these problems are exacerbated by the fact that the structure of the ECC codes forces the use of narrow-I/O x4 DRAM chips [36]. This increases the number of DRAM chips needed to achieve a given data bus width, reducing space on the DIMM for more DRAM chips, decreasing the number of independent ranks available [17]. Additionally, for a given capacity, DIMMs with narrow chips consume more energy than those with wider I/O chips [28]. Attempts to reduce the access granularity or move to x8 or x16 DRAM chips results in a significant increase in storage overhead for the ECC codes [36]. Finally, symbol-based ECC computation and verification entails significant circuit complexity due to the involvement of Galois field arithmetic, particularly with wide symbols such as 8 or 16 bits [27, 36].

With x8 DRAMs, on the other hand, b is 8, allowing just three check symbols to protect as many as 255 data symbols. We consider three protection strategies, as summarized in Table 1. While it would be most efficient from a storage overhead perspective to use a configuration of 3 ECC chips + 255 data chips, the access granularity would be unacceptably large. Reducing access granularity to a single cache line would require 3 ECC chips + 8 data chips, but storage overhead rises to 37.5%. Reducing storage overhead to 18.75% through a 3 ECC + 16 data configuration ends up reading/writing two cache lines at a time, in addition to requiring a non-standard 152-bit channel. The server industry has therefore stayed away from x8 DRAMs for chipkill-correct systems so far. Similar tradeoffs can be made with x16 or wider DRAMs, but at the cost of much sharper increases in either access granularity or storage overhead.

2.2 Virtualized ECC

Virtualized ECC (VECC) [36] is a recent academic proposal which attempts to provide chipkill-level reliability while exploiting the benefits of x8 and x16 DRAMs, without requiring non-standard DIMMs. It separates error detection and error correction into two tiers, similar also to other prior work [29, 34, 35]. Further, it proposes storing some ECC information (the second tier T2EC, responsible for data correction if an error is detected) in data memory, similar to an earlier proposal for last level caches [35]. However, VECC still suffers from some significant drawbacks. It continues to use conventional symbol-based ECC codes, inheriting all of their constraints with respect to DIMM/rank organization and access granularity. First, it requires 144-bit channels, which was not a problem in the original paper since it evaluated a DDR2 channel with a burst

Table 1. Commercial and academic chipkill implementations; burst length of 8

Design	Bus-width	Granularity (Cache lines)	Storage overhead	Problems
SSC-DSD x4, 4-check symbol code (commercial)	128b data + 16b ECC	2	12.5%	Overfetch, forced prefetching, reduced rank-level parallelism, GF arithmetic, x4 restriction
SSC-DSD x8, 3-check symbol code, Option 1	2040b data + 24b ECC	31	1.17%	Significant overfetch, non-power-of-2 data length, forced prefetching, non-standard channel width, GF arithmetic
SSC-DSD x8, 3-check symbol code, Option 2	64b data + 24b ECC	1	37.5%	Significant storage overhead, non-standard 88-bit channel, GF arithmetic
SSC-DSD x8, 3-check symbol code, Option 3	128b data + 24b ECC	2	18.75%	Overfetch, forced prefetching, reduced parallelism, non-standard 152-bit channel, GF arithmetic
VECC x8, 3-check symbol code, 1 symbol virtualized	128b data + 16b ECC	2Rd/4Wr	18.75%	Overfetch, forced prefetching (both even worse for writes), performance impact due to writes, reduced parallelism, GF arithmetic, modifies various system components
SSA x8, checksum+parity	64b data + 8b ECC	1	25%	Increased storage overhead, performance impact due to extra writes to global parity, modifies DRAM microarchitecture
LOT-ECC x8, checksum+parity	64b data + 8b ECC	1	26.5%	Increased storage overhead

length of 4 – every access is only 144*4 bits, a single 64-byte cache line. However, most servers today use DDR3 channels, with a minimum burst length of 8, forcing prefetch. Half-burst accesses (length = 4) are possible in DDR3, but this simply masks out the last four bursts – no transfer occurs, and half the bandwidth is wasted. Second, 18 x8 chips are made busy on each access (twice the minimum required), increasing overfetch and reducing rank-level/bank-level parallelism. Third, these problems are exacerbated for memory write operations. Specifically, the data mapping adopted by VECC leaves open the possibility of the T2EC being placed in a failed DRAM chip if a single rank is used. It therefore forces the T2EC code to be placed in a different rank (also 18 x8 chips), thus raising the number of chips touched when writing to memory to 36. It also means that we can only tolerate failure in at most 1 in 36 DRAM chips. Finally, VECC requires modifications to several components of the system, including the operating system, memory management unit, caches, etc., making it difficult to implement.

2.3 Single Subarray Access (SSA)

Another recent proposal to improve chipkill design is SSA [34]. It was able to reduce access granularity to just 9 x8 DRAM chips and one cache line, at a tolerable storage overhead of 25%. However, it was specifically targeted at a novel DRAM microarchitecture (Single Subarray Access, SSA) also proposed in the same paper, and required cache lines to be localized to a single DRAM chip unlike commodity architectures. It exploited certain RAS features built into the microarchitecture, such as additional space in each row to store local ECC information. Also, it suffered from non-trivial penalties during write operations since some ECC information computed across several different cache lines had to be updated every time *any* of those lines was written to.

2.4 Other Related Work

Memory reliability has received increased attention from the architecture community in recent years. Schroeder et al. studied DRAM failures in large-scale datacenters at Google [30]. The Rochester Memory Hardware Error Project [22] characterized failures not just at a platform level, but went right down to individual bits. Both papers highlighted the importance of memory errors going forward, but did not propose novel reliability mechnisms. Papers on novel memory architectures such as Mini-Rank [37] and MC-DIMM [9] have included discussions on their impact on ECC and/or chipkill, but have not proposed any fundamental changes.

Key Differentiators: *LOT-ECC solves most of the problems discussed above. It works with just a single rank of nine x8 chips, improving access granularity, energy consumption, and performance. Also, both commercial designs and VECC define chipkill as the failure of 1 chip out of 36, whereas LOT-ECC supports 1 dead chip in 9, a significant boost in reliability guarantee. As with prior designs, the target fault model is the failure of one entire chip, and the detection of failures in at least 2 chips. Finally, LOT-ECC is transparent to everything but the memory controller and the firmware, making it less invasive, and more implementation friendly. LOT-ECC pays for this reduction in energy,improvement in performance, and simplicity through a slight increase in storage overhead for the ECC codes – a total of 26.5% (both tiers together), compared to 18.75% with VECC and 12.5% with commercial SSC-DSD.*

3 Proposal: LOT-ECC Design

LOT-ECC takes a novel, fundamentally different approach to reliability. It splits protection into multiple tiers to detect errors, isolate their location, and reconstruct the erroneous data, and maps all data and codes into the same row-buffer to allow efficient access.

3.1 Local Error Detection (LED)

The first layer of protection afforded by LOT-ECC is *local error detection (LED)*. The function of this code is to perform an immediate check following every read operation to verify data fidelity. Additionally, it needs to identify the exact location of the failure, at a chip-granularity within a rank. To ensure such chip-level detection (required for chipkill), the LED information itself needs to be maintained at the chip level – associated not with each cache line as a whole (as in symbol-based ECC codes), but with every cache line "segment", the fraction of the line present in a single chip in the rank. In Figure 1, for example, cache line A is divided into segments A_0 through A_8. The corresponding local error detection codes are L_{A0} through L_{A8}.

Data Layout: Consider a standard ECC rank with nine x8 DRAMs, and a burst length of 8, as shown in Figure 1. Instead of treating this as eight data chips and one ECC chip, we propose storing both data and LED information on *all nine chips*. Since we need 512 data bits (one cache line) in total, each chip will have to provide 57 bits towards the cache line. An x8 chip supplies 64 bits per access, which are interpreted as 57 bits of data (A_0 in Figure 1, for example), and 7 bits of *LED* information for those 57 bits (L_{A0}). Across 9 chips, this translates to 513 bits of data and 63 bits of LED. Since a cache line is only 512 bits, there is one *surplus bit*, which we will utilize as part of the second layer of protection (Section 3.2. At this point, simply note that the 7-bit LED *detects* errors in 57 bits of data – code details follow in Section 3.5.

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

3.2 Global Error Correction (GEC)

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

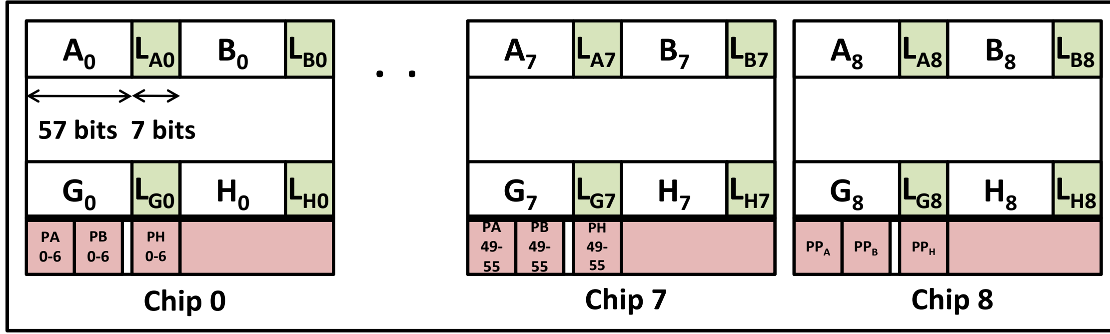
Data Layout: In line with our objective to provide strong fault-tolerance of 1 dead chip in 9, and to minimize the number of chips touched on each access, we choose to place the GEC code in the same rank as its corresponding cache line, unlike prior proposals [36].

We set apart a specially-reserved region (shaded red in Figure 1) in each of the 9 chips in the rank for this purpose. This is a subset of cache lines in every DRAM page (row), although it is shown as a distinct set of rows in Figure 1 for clarity. This co-location ensures that reads or writes to the GEC information will be guaranteed to be a row-buffer hits when made in conjunction with the read or write to the actual data cache line, thus reducing its performance impact. With this new data-mapping, a 9 KB row buffer can only accommodate 8 KB of data+LED; the rest is used to store the GEC codes for that 8 KB. This requires some basic arithmetic circuitry at the memory controller to appropriately calculate the row-id for each physical page.

Figure 2 shows the way the GEC information is laid out in the reserved region, for an example cache line A. Similar to the data bits, the 57-bit parity PA is itself distributed among all 9 chips. The first seven bits of the PA field (PA_{0-6}) are stored in the first chip, the next seven bits (PA_{7-13}) are stored in the second chip, and so on. Bits PA_{49-55} are stored on the eighth chip. The last bit, PA_{56} is stored on the ninth chip, in the *surplus bit* borrowed from the Data+LED region (see Section 3.1).

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

Next, consider the case where in addition to a fully failed chip, there is an error in a second chip – we still need to be able to detect, if not correct, such a failure



A, B, C, D, E, F, G, H – Cache Lines, each comprised of segments X_0 through X_8

L_{XN} – L1 Local Error Detection for Cache Line X, Segment N

$[PX_0{:}PX_N]$ – L2 Global Error Correction across segments X_0 through X_8

PP_X – Parity across GEC segments $PX_{0\text{-}6}$ through $PX_{49\text{-}55}$

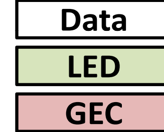


Figure 1. LOT-ECC shown with a single rank of nine x8 DRAM chips

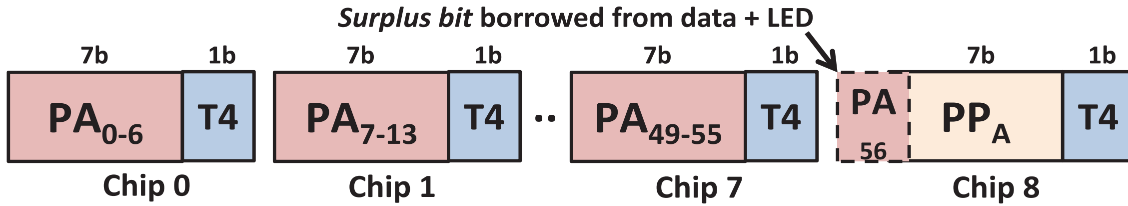


Figure 2. Data layout for one cache line's GEC in the red-shaded GEC region

under our fault model. If this second error is also a full-chip failure, it will be detected by the LED along with the initial data read, and flagged as an uncorrectable double-chip failure. On the other hand, if this second error occurs just in the GEC region of interest, it needs to be detected during the GEC phase.

For example, assume that the second chip has completely failed – we have now lost A_1, and PA_{7-13}. If, in addition, there is an error in the GEC region of the first chip, there is a possibility that one or more of the bits PA_{0-6} are corrupt. The reconstruction of lost bits PA_{7-13} from PP_A and PA_{0-6}, PA_{14-20}, PA_{21-27}, .., PA_{56} may itself be incorrect. To handle this problem, we use the remaining 9 bits (marked $T4$, for Tier-4, in Figure 2) to build an error *detection* code across GEC bits PA_0 through PA_{55}, and PP_A. Note that neither exact error location information nor correction capabilities are required at this stage since the reliability target is only to *detect* a second error, and not necessarily correct it. We can therefore build a code using various permutations of bits from the different chips to form each of the $T4$ bits. This should include multiple bits from the same chip, and bits from different columns across chips to maximize the probability of detection.

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

If one of the first eight chips, say chip-1, fails, 57 bits of data (A_1) are lost, in addition to GEC parity information PA_{7-13}. We first read A_0 - A_8, and the LED associated with A_1 (L_{A1}), indicates a chip error. We then read GEC segments PA_{0-6}, PA_{14-20}, PA_{21-27}, .., PA_{49-55}, $PA_{56}$and PP_A to recover the lost GEC bits PA_{7-13}, thereby reconstructing GEC parity PA. Combined with values A_0 and A_2 - A_7, data value A_1 can be reconstructed, thus recovering the entire original cache line. If, on the other hand, the ninth were to fail, only 56 bits of data are lost (A_8), in addition to PP_A, and the surplus bit PA_{56}. The lost 56 bits can be recovered simply from the 56 bits of parity stored in the first eight chips (PA_{0-55}), and clean segments A_0 - A_7. The loss of surplus bit PA_{56} is immaterial. Across these cases, the fidelity of the GEC parity bits themselves is guaranteed by $T4$.

Impact on Memory Reads and Writes: Read operations do not need to access GEC information unless an error is detected, which is a rare event. GEC, therefore, has no significant impact on reads. Write operations, on the other hand, need to update the GEC (which includes P_X, PP_X, and $T4$) when any data is modified. In a basic implementation, each cache line write gets transformed into two writes – one to the data location (for a full 576 bits of data+LED+surplus bit) and another to its corresponding GEC location (72-bits). We minimize the performance impact of this additional write by mapping data and GEC to the same DRAM row, guaranteeing a row-buffer hit. This trick is similar to Loh and Hill's optimization to store tag and data for a cache line in the same row in a large DRAM cache [23]. Additionally, note that there isn't a need for a read-before-write of the data cache lines themselves as in some earlier proposals [34], since all bits contributing to the GEC code are from a single cache line, already available at the controller. This also helps keep performance impact low.

Although only 72 bits of GEC+T4 code need to be updated per write, we are forced by the DDR3 protocol to complete a burst of 8 per access (an entire 72-byte "cache line" size of data). We could, therefore, potentially combine as many as 8 different GEC updates into a single write command, minimizing performance impact further. This is low-overhead since writes are already buffered and streamed out intermittently from the memory controller to avoid frequent bus turnaround. Additional logic can easily be implemented to coalesce as many GEC writes as possible. Every write is transformed into $1 + \delta$ writes ($\delta \leq 1$) depending on the application's access characteristics and the write-buffering policies. $\delta = 1$ in a non-coalesced basic LOT-ECC implementation, and 0.125 in an oracular design since eight GEC words fit in a single "cache line", and could potentially be coalesced into a single write. The values of δ for the PARSEC [12] benchmark workloads are shown in Figure 3 (details on methodology follow in Section 4). Note that for the purposes of this paper, we employ a very rudimentary coalescing scheme where the "window" we look into to find suitable writes to coalesce is simply the set of writes that are sent out by default each time the bus is turned around for a "write burst". More complicated schemes that may selectively buffer writes for longer in an effort reduce δ are certainly possible – we leave such optimizations to future work.

If complete coalescing is not possible (based on the addresses being written to), data masking [18] can be employed to only write the appropriate bits into memory. Note that the complete burst of 8 has to performed nonetheless – some pieces of data are just masked out while actually writing to DRAM.

Table 2. LOT-ECC with x16 DRAMs

Design	Access Granularity (Cache lines)	Storage Overhead	Customization Required
Option 1	2	26.5%	None
Option 2	1	50.0%	None
Option 3	10	37.5%	None
Heterogeneous DIMMs	1	37.5%	Commodity x16 & x8 DRAMs on a custom DIMM

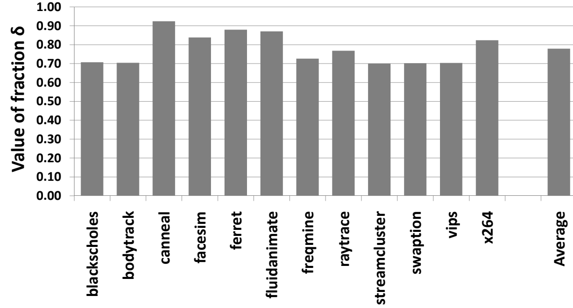
3.3 Storage Costs of LOT-ECC

For each 64-byte cache line, LOT-ECC requires the following ECC bits: (1) 63 bits of LED information, at 7 bits per chip, (2) 57 bits of GEC parity, spread across the nine chips, (3) 7 bits of third-level parity, PP_X, and (4) 9 bits of $T4$ protection, 1 bit per chip. This adds up to a total of 136 bits, a storage overhead of 26.5%. Out of this 26.5%, 12.5% is provided by the 9th chip on standard ECC DIMMs, and the other 14% is stored in data memory.

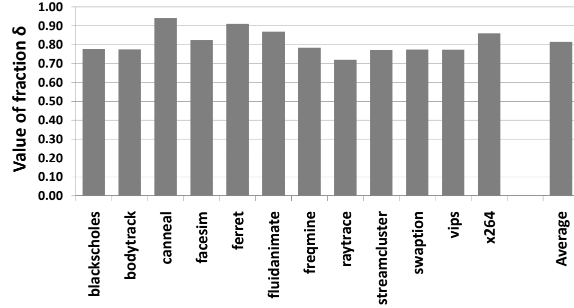
3.4 LOT-ECC with Wide-I/O DRAMs

Wider-I/O DRAM parts are favored over narrower DRAMs since they are typically more power efficient [36] and enable greater rank-level parallelism. To enable their widespread adoption, there is a need to develop efficient reliability mechanisms that work with such configurations. However, there are several unknowns regarding the use x16 and x32 DRAMs in future commodity DIMMs – data bus width increases or the provisioning of additional chips for ECC support, for example. We therefore present a brief discussion of LOT-ECC implementation under a few possible scenarios, and propose the concept of *heterogeneous DIMMs* to address some concerns with these options. For clarity, we only quantify overheads with x16 DRAMs; similar ideas can be extended to x32 DRAMs and beyond. The options are summarized in Table 2.

Option 1 - Wide Memory Channels: If the natural progression of memory system design is to simply increase the channel width as wider-I/O DRAMs become common, LOT-ECC can be implemented with little modification. For instance, consider a rank of nine x16 DRAMs. The 128 bits supplied by an x16 DRAM in a burst of 8 would simply be interpreted as 114 data bits and 14 checksum LED bits, the same storage overhead as with x8 DRAMs. GEC operation remains unchanged. There is necessarily an increase in access granularity and overfetch, independent of LOT-ECC. This will also double with the further doubling of I/O width to x32 or x64. Total storage overhead remains at 26.5%.



(a) Xeon 7500 based system



(b) Xeon 5500 based system

Figure 3. Quantification of GEC coalescing opportunity

Option 2 - Increasing Storage Overhead: If access granularity is fixed at exactly one cache line, the minimum rank size with x16 chips is 5 chips,4 data and 1 ECC. Each chip provides 128 bits per burst of 8, interpreted as 103 data bits (since 103 * 5 chips = 512-bit cache line). This leaves 25 bits per chip to store the LED code, which provides very strong error protection, but is likely overkill and wasteful of storage area (the overhead is 24.3%). GEC overhead increases as well, since the global parity is a 103-bit entity computed over five 103-bit data segments. After including storage for T3 and T4, the total overhead is about 50%.

Option 3 - Optimizing Storage Overhead: If storage overhead is an important consideration, it can be fixed at about 12.5%, paying for it through an increase in access granularity. With x16 chips and a 5-chip rank, for example, 9 reads can be issued consecutively, reading out a total of 80 bits per cycle * burst of 8 cycles * 9 accesses = 5,760 bits. This results in a very large access granularity of 10 cache lines (5120 bits) plus their LED codes, a storage overhead of 12.5%. The GEC overhead remains approximately 25%, similar to Option 2, for an overall ECC storage overhead of about 37.5%. There may also be some forced prefetching since a single burst also partly reads bits from a second cache line.

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

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

3.5 DRAM Errors and LED Checksum

The target of the LED code (Section 3.1) is to *detect* errors in 57 bits of data through the use of a 7 bit checksum. The 57 bits of data are divided into 8 blocks of 7 bits each. The last bit is padded with 6 zeros to make it a 7-bit value. A 7-bit checksum is produced by performing an integer one's complement addition of these nine blocks. The carry-out information of the MSB is preserved via being wrapped around and added back to the LSB, improving error detection capability. We refer the reader to Maxino [24] for further details on the operation of checksums.

Prior work has shown that DRAM errors occur in a few major ways – single-bit error, double-bit error, row-failure, column-failure, row-column failure, pin failure, and full chip-failure [13, 21, 22, 31]. We consider each of these in turn, and show how the proposed LED checksum handles error detection. Table 3.5 summarizes both the raw FIT[1] and the *effective* FIT, *i.e.,* the failure rate with a fault tolerance mechanism in place.

Single-bit error: This is among the most common failure modes in DRAM, and can be either due to a soft-error or a hard-error. Any checksum is guaranteed to detect this kind of error.

Double-bit error: These are defects that span two *nearest neighbor* memory cells and are *not* the coincidence of two independent single bit errors [31]. Adjacent bits

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

Table 3. LOT-ECC LED Error Detection

Error-type	Raw FIT	Eff. FIT (Chip-kill)	Eff. FIT (LOT-ECC)	Eff. FIT (Simple SEC-DED)
Single-bit soft	5000 [21]	0	0	0
Single-bit hard	12.6 [31]	0	0	0
Double-bit	0.7 [31]	0	0	0
Row	6.3 [31]	0	0	6.3
Pin	4.1 [31]	0	0	0
Row-Column	4.2 [21]	0	0	4.2
Chip	13.7 [31]	0	0	13.7
Multiple random (hard+hard)	-	0	1.1E-6	>0
Multiple random (hard+soft)	-	0	4.7E-4	>0
Multiple random (soft+soft, NO scrubbing)	-	0	0.19	>0
Failure of 2 chips in 18, errors in >2 chips per rank, etc.	-	>0	0	>0

Soft errors will likely be caught by patrol scrubbers, and will rarely occur in conjunction with other errors

will fall into adjacent checksum-columns[2], only flipping a single bit in each column. This will alter the checksum value, flagging the error.

Row-failure: These are caused by a failure of one of the chip's row drivers, causing all cells in one row of the affected chip to fail. We employ bit inversion in the checksum computation so that such stuck-at faults are detected [34]. With this in place, the checksum detects an entire row incorrectly being either all 0 or all 1.

Column-failure: These are caused by the failure of a senseamp in an array, making the entire column of bits unreadable. For a given request, this is equivalent to a single-bit failure, and can be detected by the checksum.

Row-column failure: This error occurs when an intersecting row and column fail. It does not require any special handling beyond a simple row or column error depending on the data being accessed.

Pin-failure: This error occurs when a data-pin (DQ) gets stuck at 0 or 1. It might also be caused by a failure of one of the column senseamps or column decoders, with similar effects as a failed pin. With the proposed LED mechanism, since each block is 7 bits, and each pin outputs 8 bits per access, there is a "wrap-around" that occurs naturally. This means that the eight potentially erroneous bits coming from the failed pin fall into distinct checksum-columns, with the exception of the last bit. Since we employ an addition-based checksum (rather than XOR based), the presence of the carry-bit protects against the simultaneous inversion of those two overlapping bits; note that the undetectable combination of 0 to 1 flip in one bit and 1 to 0 flip in the other cannot occur with stuck-at faults.

Chip-failure: For a given cache line, handling a chip error is conceptually equivalent to handling a row failure, and the same arguments with regard to error detection hold good. The difference lies primarily in post-error service operation. With a row error, there is a possibility of re-mapping the data from the failed row (upon reconstruction) into one of several *spare rows* typically provisioned on DRAM chips, called Memory Page Retire [31]. With an entire chip failure, the DIMM is immediately flagged for replacement.

Multiple random bit errors: Addition-based checksums are susceptible to silent data corruption if an even number of bits in a given checksum-column flip. In our LED checksum, Column 0, for example, consists of bits 0, 7, 14, 21, 28, 35, 42, 49, and 57 from a single row. Since there is 6-bit gap between the closest two bits, it is highly unlikely that the same failure "cause" (other than a row or column failure), such as a soft-error strike, can result in such bit flips. This means that the two have to be *independent* single-bit events. Since each error is already rare in general, the probability of a *combination* of errors affecting the same data word in the same locality is extremely low [22]. To quantify, consider the 12.6 FIT number for a single-bit hard error. If a single DRAM has 16 M cache line segments, the probability of an error in a given segment during a billion hour period is 12.6 / 16 M, which is 7.51E-7 – this is the *segment FIT*. The probability of two independent errors affecting the same segment is therefore 5.64E-13 per billion hours. With each segment being 64 bits, the probability that the two erroneous bits are among the nine bits 0, 7, 14, 21, 28, 35, 42, 49, and 57 is (9C2)/(64C2). Since errors can occur in any of the 7 checksum-columns, we multiply this value by 7, giving 0.125. The probability of a two bits flipping in a way that is undetected is, therefore, 5.64E-13 * 0.125, which is just 7E-14. Finally, this can occur in any of 16 M segments, giving an overall *undetectable* FIT of 1.1E-6. Next, consider the combination of a single-hit hard error and a single-bit soft error. Calculations similar to the one above indicate an undetectable FIT of 4.7E-4. Finally, consider the case of two soft errors. Modern servers typically employ some form of patrol scrubbing which periodically corrects any soft errors in the array. It is therefore unlikely that two soft errors will occur in the same row leading to an unde-

[2] the set of bits that line up vertically for addition during the checksum generation process

tectable error. Even worst-case calculations for a system with *no* scrubbing, and a *pessimistic* assumption of 5000 FIT for soft errors [21] results in an undetected error rate of 0.19 FIT. The odds of 4 or 6 or 8 independent random errors all affecting the same segment are even smaller.

Combination Errors: Even when the individual raw FIT of a fault is as high as 5000 (as with the single-bit soft error), we showed that the probability of multiple simultaneous occurrences leading to silent data corruption is extremely small. This is even more true for other combinations such as multiple column decoder/senseamp failures on the same data word, or a sensamp failure plus a single-bit soft error, etc., since their individual raw FITs are already orders of magnitude smaller than 5000. Such *combination errors*, therefore, will have a negligible impact on overall reliability levels.

Discussion: Existing commercial chipkill mechanisms formally guarantee catching *any and all* possible combinations of bit flips within a chip, paying significant power and performance penalties to guard against obscure bit-flip combinations that occur rarely, if ever. However, field studies indicate that DRAM errors typically fall under a very small number of failure modes with specific root causes. LOT-ECC is cognizant of these, and captures all commonly occurring faults, while providing substantial power and performance benefits. Additionally, LOT-ECC captures some failure scenarios that current mechanisms cannot. Examples include the failure of 2 chips in 18, the simultaneous occurance of a single chip failure and single bit errors in *all* other chips, etc.

Given the significant power/performance overheads of fault-tolerance, the burden of memory reliability will likely not be foisted solely upon ultra-robust catch-all ECC codes, especially considering trends such as rising burst lengths, wider chip I/Os, and increasing system capacities. Strong yet practical fault-tolerance codes, combined with RAS features such as patrol scrubbing [1], spare-row mapping, bit steering, memory page retire [31], will provide the best power-performance-reliability tradeoff at the system level. These will likely "cure" a majority of faults before they deteriorate into undetectable errors.

4 Benefits and Results

4.1 Methodology

Performance Studies: To study the performance characteristics of our designs, we use a detailed in-house DRAM simulator. We accurately model refresh overheads, address/command bus, data bus, bank/rank/channel contention, and the memory controller queue. We also model a separate write-queue that holds write operations (and the associated GEC writes) as they come in to the memory controller, and issues them in bursts to amortize bus-turnaround overheads. We use close-page row-buffer management with a simple FCFS policy, since the access pattern has little impact on the operation of LOT-ECC. Close-page policies are the norm today for industry standard servers from all major vendors in an effort to improve performance in typical workloads with relatively low locality [8, 19, 25]. The one exception we make is for write operations – the data line and GEC lines (coalesced or otherwise) are guaranteed to be co-located in a single row buffer, making a semi-open-page policy beneficial. We adopt the baseline address mapping policy from Jacob et al. [17]. DRAM timing parameters were obtained from Micron DDR3 datasheets. We employ a synthetic traffic generator with a tunable rate of traffic injection, and a tunable fraction of write operations for a majority of our experiments. We use the PARSEC [12] benchmarks to evaluate the GEC coalescing scheme.

Power Calculation: To accurately calculate memory power consumption, we directly use Micron's DDR3 power calculator spreadsheet [6]. The spreadsheets require inputs regarding bus utilization, bank utilization, etc., which we obtain from our performance simulation. The calculator accounts for activation power, read/write power, termination power, and background power. It also incorporates support for low-power modes, which we assume are oracularly applied whenever all banks are idle. We measure power when the system is under heavy load, with traffic just below the system's saturation point, as determined by our performance studies.

Target System Configurations: We evaluate our designs on memory systems based on two contemporary high-end server processors from Intel. The same basic controller/channel/DIMM organization and reliability mechanisms are used by all major server vendors. First, we consider Xeon 7500 based systems [3, 4, 7], with a typical configuration as shown in Table 4.1. All Xeon 7500 based servers mandatorily implement *Lockstep Mode* in the memory system. This gangs two x72 ECC-DIMMs and channels together to form a 144 bit bus, essential for today's reliability mechanisms.

Second, we consider Xeon 5500 based systems [2, 5], with a typical configuration as shown in Table 4.1. Since these processors have an odd-number of channels (three, in this case), the effect of lockstep mode is pronounced. While the processor does allow operation in non-lockstep mode if strong reliability is not desired, chipkill mode simply leaves one channel empty, and gangs the other two together. This results in a significant performance hit. Note that design decisions leading to an odd number of channels are a result of complex interplay between pin availability, expected bandwidth demand based on target workloads and processor horse-

Table 4. Main Memory Configuration

Parameter	Xeon 7500	Xeon 5500
Protocol	DDR3	DDR3
Channels	8	3
DIMMs/channel	2	3
Ranks/DIMM	2 for x4; 4 for x8	2 for x4, 4 for x8
Banks/Rank	8	8
Capacity	128 GB	72 GB

power, re-use of designs for varying market segments (desktops, laptops, and servers), etc. It is therefore certainly possible that such designs will continue to exist in future commercial processors, despite the large impact on power and performance for reliability.

Reliability Models Evaluated: Our baseline model is a modern commercial Single Symbol Correct Double Symbol Detect (SSC-DSD) design, as described in Section 2. This is currently supported only with x4 DRAMs. Additionally, it requires the use of lockstep-mode operation. Our second baseline is VECC [36], which enables the use of x8 DRAMs in lockstep mode, while suffering a small write-penalty. We obtain the T2EC cache miss rate, *i.e.,* the fraction of writes that require an additional "redundant information write", from the original paper, and include it in our simulator – the VECC model is therefore an optimized baseline, accounting for T2EC caching. We also optimistically assume that all the hardware/software support mechanisms required to make VECC work happen oracularly with no performance or power impact. We first evaluate the basic LOT-ECC design, which allows us to upgrade to x8 DIMMs, *without* the use of lockstep mode. In this mode, *every* write is transformed into two writes – one data and one GEC. We then show the benefits of coalescing GEC writes to reduce their performance impact. Finally, we show an oracular design with perfect coalescing, where only one in eight writes suffers from the overhead of an additional GEC write.

4.2 Power Savings

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

Dynamic power: Commercial chipkill solutions touch thirty-six x4 chips both to read and write [10]. VECC [36] activates eighteen x8 chips per read and thirty-six x8 chips per write (data + T2EC). LOT-ECC, on the other hand, activates the absolute minimum number required, just nine x8 chips (writes may require an additional row-buffer access to the same chips). This change in activation granularity results in a reduction in dynamic energy consumption. LOT-ECC also eliminates *forced prefetching*, since only one 64-byte cacheline is touched in a standard burst-of-8 access, further reducing dynamic energy.

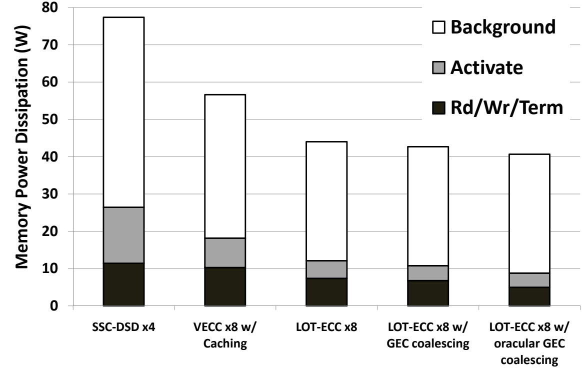


Figure 4. Power in Xeon 7500 systems

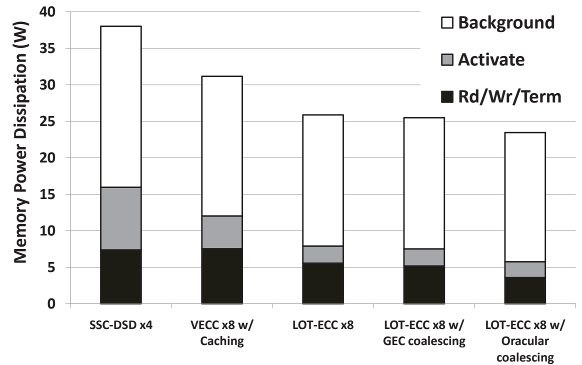


Figure 5. Power in Xeon 5500 systems

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

The combined power consumption is shown in Figures 4 and 5. In the Xeon 7500-based system, LOT-ECC reduces power consumption by 43.1% compared to the SSC-DSD baseline. Reducing the number of writes to GEC locations through GEC coalescing increases the savings to 44.8%, based on average δ from Figure 3. With an oracular coalescer, savings can potentially increase to 47.4%. The trends are similar with 5500-based systems, although the power savings are somewhat lower. This is because some of the savings are offset by increased static power consumption due to the increased number of DIMMs made possible when LOT-ECC relaxes the lockstep mode constraint and populates the third channel. Power savings with respect to the SSC-DSD baseline are 31.9% with basic LOT-ECC, 32.9% with GEC coalescing, and 38.3% with oracular GEC coalescing. VECC's energy consumption lies between those of SSC-DSD and LOT-ECC; we confirm that VECC's power savings compared to the SSC-DSD baseline are in line with numbers reported in the original paper. Compared to this baseline, LOT-ECC with coalescing reduces power consumption by 24.6% in 7500-based systems, and 18.2% in 5500-based systems.

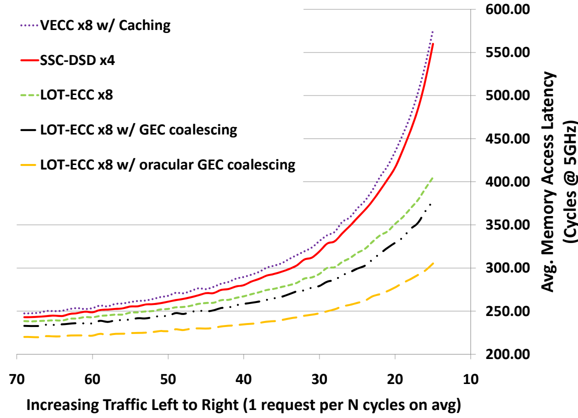


Figure 6. Latency in Xeon 7500 systems

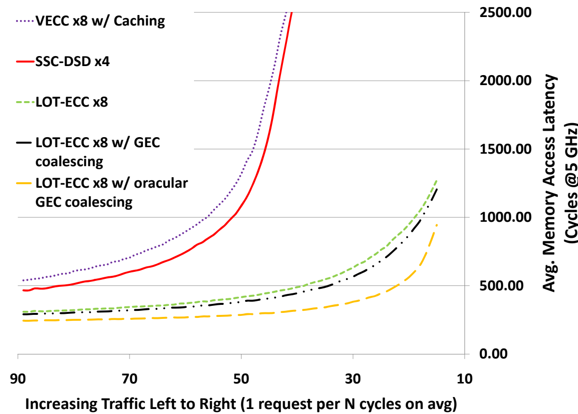


Figure 7. Latency in Xeon 5500 systems

4.3 Performance Gains

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

Figures 6 and 7 shows the average memory access latency as a function of traffic. With the 7500-based system, at a traffic rate of 1 request per 40 cycles (just under saturation), the basic implementation of LOT-ECC x8 provides 4.6% latency reduction compared to the SSC-DSD baseline. GEC coalescing further enhances the improvement to 7.7%, with an oracular policy providing a maximum of 16.2% reduction in latency. Latency reductions are much more substantial in the 5500-based system, since LOT-ECC relaxes the lockstep-mode constraint, providing substantially increased channel bandwidth. At a traffic rate of 70 (SSC-DSD in lockstep mode saturates soon after), latency reductions are 42.9% with LOT-ECC x8, 46.9% with GEC coalescing, and an oracular maximum of 57.3%. VECC actually performs marginally worse than the SSC-DSD baseline (confirming the results in the original paper). Compared to VECC, LOT-ECC with coalescing reduces average memory latency by almost 11% in 7500-based systems, and 54% in 5500-based systems. Additionally, LOT-ECC allows the channel to operate without saturating even under substantially higher traffic rates.

4.4 Positive Impact on System Design

LOT-ECC imposes few restrictions on the choice of DRAM parts, DIMM layout, DDR protocol, burst length, etc. In fact, with larger DRAM parts such as x16 or x32, the size of the data segment on each chip increases, and it is often more efficient to build strong error detection codes over larger data words. It is as yet unclear what DIMM configurations the memory industry will adopt with x16 or x32 chips, and LOT-ECC can work well in any case Additionally, LOT-ECC requires absolutely no modification to the DRAM parts or interface. It only requires support from the memory controller (data mapping and interpreting bits as either data, LED, or GEC) and system firmware (to reduce the amount of memory space visible to the OS to accommodate GEC). These are relatively more amenable to change [16], and require the participation of fewer design teams, removing a few hurdles for commercial adoption. Another consideration is that Galois field arithmetic over 16-bit or 32-bit symbols (required with the switch to x16 or x32 parts) can get complicated to implement [27, 36], increasing complexity, latency, and energy consumption. LOT-ECC utilizes simple additive checksums and parity calculations to provide strong fault tolerance, reducing the required design effort, and saving a small amount of power.

4.5 Storage Overhead

The price LOT-ECC pays to achieve excellent power, performance, and complexity characteristics is a small increase in storage overhead. Conventional schemes spread data across a large number of DRAMs, keeping ECC overhead at 12.5%. LOT-ECC utilizes this space to store the LED, and requires a further 14% storage area for GEC. However, since only cheap commodity memory is used, this will likely be a price that server vendors will be willing to pay [16]. Additionally, overall TCO is reduced due to the substantial energy savings.

5 Conclusions

The power, performance, storage, and complexity overheads of providing strong levels of fault-tolerance are already very significant. Various trends in memory system design such as increasing burst length, wider I/O DRAM chips, increasing contribution of memory to system-level power/performance, larger memory capacities, greater need for parallelism, and less reli-

able hardware will exacerbate this problem. There is a need to fundamentally rethink memory error protection in a manner that is cognizant of these trends and improves efficiency. We present LOT-ECC, a novel chipkill-level reliability mechanism that employs simple multi-tiered checksum and parity fields that effectively address various DRAM failure modes. It enables the use of minimally-sized ranks of x8 DRAMs, reducing energy consumption by up to 44.8% and average memory access latency by up to 46.9%, while simultaneously simplifying implementation. Moreover, it achieves these benefits without affecting the design of commodity memory DIMMs or channels, and is also transparent to other system components such as the OS and caches. It pays for all these advantages through a small increase in storage overhead, considered an acceptable cost in the memory industry as long as commodity DRAMs are used.

References

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

[2] Dell Help Me Choose: Memory. `http://goo.gl/paF1c`.

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

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

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

[6] Micron System Power Calculator. `http://goo.gl/4dzK6`.

[7] Oracle's Sun Fire X4800 Server Architecture. `http://goo.gl/h0efI`.

[8] N. Aggarwal et al. Power Efficient DRAM Speculation. In *Proceedings of HPCA*, 2008.

[9] J. Ahn et al. Future Scaling of Processor-Memory Interfaces. In *Proceedings of SC*, 2009.

[10] AMD Inc. BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors.

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

[12] C. Bienia et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical report, Princeton University, 2008.

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

[14] C. L. Chen. Symbol Error Correcting Codes for Memory Applications. In *Proceedings of FTCS*, 1996.

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

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

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

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

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

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

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

[22] X. Li et al. A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility. In *Proceedings of USENIX*, 2010.

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

[24] T. Maxino. *The Effectiveness of Checksums for Embedded Networks*. PhD thesis, 2006.

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

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

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

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

[29] N. N. Sadler and D. J. Sorin. Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache. In *Proceedings of ICCD*, 2006.

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

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

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

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

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

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

[36] D. Yoon and M. Erez. Virtualized and Flexible ECC for Main Memory. In *Proceedings of ASPLOS*, 2010.

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