Flexible Cache Error Protection using an ECC FIFO

Doe Hyun Yoon
Electrical and Computer Engineering Department
The University of Texas at Austin
doehyun.yoon@gmail.com

Mattan Erez
Electrical and Computer Engineering Department
The University of Texas at Austin
mattan.erez@mail.utexas.edu

ABSTRACT

We present ECC FIFO, a mechanism enabling two-tiered last-level cache error protection using an arbitrarily strong tier-2 code without increasing on-chip storage. Instead of adding redundant ECC information to each cache line, our ECC FIFO mechanism off-loads the extra information to off-chip DRAM. We augment each cache line with a tier-1 code, which provides error detection consuming limited resources. The redundancy required for strong protection is provided by a tier-2 code placed in off-chip memory. Because errors that require tier-2 correction are rare, the overhead of accessing DRAM is unimportant. We show how this method can save $15-25\%$ and $10-17\%$ of on-chip cache area and power respectively while minimally impacting performance, which decreases by 1% on average across a range of scientific and consumer benchmarks.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; B.3.4 [**Memory Structures**]: Reliability, Testing and Fault-Tolerance—*Error-checking*

General Terms

Reliability, Design

Keywords

soft error, error correction, last-level caches, reliability

1 Introduction

In this paper we present ECC FIFO – a strong error protection architecture aimed at large on-chip *last-level caches* (LLCs) that minimizes hardware energy and area overheads while having minimal impact on performance. Our technique is based on a two-tiered ECC scheme and off-loads all of the storage required for strong error correction to off-chip DRAM, potentially leaving only low-cost error detection on chip. We store error-correction bits in a FIFO structure in main memory, instead of in a dedicated on-chip SRAM array. The tiered approach also offers a generalization of energy-reducing non-uniform error correction techniques; low-overhead and low-latency first-tier detection or correction is performed on every access to the cache, while stronger and more costly second-tier correction is only engaged once an error is detected.

It is important to minimize the impact on area, power, and application performance of strong error protection mechanisms because of the continuing increase in soft-error propensity. The combination of growing LLC capacity, shrinking SRAM cell dimensions, and increasing fabrication variability that reduces error margins is leading to a higher *soft error rate* (SER) [19, 27, 29]. In particular, recent trends show that multi-bit burst errors in the array are becoming significant, partially because many memory cells can fall under the footprint of a single energetic particle strike [19, 21, 26].

Because of these trends, the need to account for high LLC SER and also tolerate multi-bit errors in SRAM arrays is becoming acute [3]. Researchers have already observed up to 16-bit errors in SRAM arrays and are predicting potentially higher counts in the future [1, 19, 21]. The more powerful error protection mechanisms required to correct large numbers of bit flips come at a cost of storing more redundant information or modifying physical designs, as well as increased energy requirements [12, 23]. We aim to reduce the overheads of providing increased protection against soft errors in large SRAM arrays. We base our efficient error protection architecture on four important observations that have been reported in our recent work:

- While SER is increasing and cannot be ignored at any memory hierarchy level, error events are still expected to be extremely rare for a given processor. For instance, the mean time to failure (MTTF) of a 32MB cache is around 155 days assuming 10^{-3}FIT/bit [27] (FIT: *failures in time*).
- Every write to the LLC is accompanied by computing information required for error correction as well as for detection. The latency of this operation, however, does not affect performance as long as write throughput can be supported.
- Given that SER is low, the latency and the complexity of error correction is not important, since it would occur once every several weeks or months on a given processor.

This work is supported in part by donations from the Intel corporation

- Much of the data stored in the LLC is also stored elsewhere in the memory hierarchy (clean cache lines). Replicated data is inherently soft-error tolerant, because correct values can be restored from a copy. Thus, it is enough to detect errors in clean lines.

We combine the observations above into an architecture that decouples error detection from error correction and that stores all the redundant information required for correction in a FIFO structure located in low-cost off-chip DRAM. Unlike prior techniques that reduce the cost of on-chip ECC, including our recently introduced Memory-Mapped ECC [36], ECC FIFO, which we develop in this paper, does not use any on-chip storage for strong error correction and keeps only detection, and potentially light weight correction, information on chip. Unlike prior techniques for reducing ECC cost, ECC FIFO does not change the caching behavior of applications, and we demonstrate the importance of this trait in a direct comparison to prior work. Another important feature of ECC FIFO is the flexibility it enables in choosing two-tiered code combinations. We analyze the advantages of different two-tier schemes over their one-tier counterparts with respect to protection and cost.

We evaluate our architecture using full-system cycle-based simulation of memory-intensive applications from the SPLASH2 [34], PARSEC [2], and SPEC CPU 2006 [28] suites. We show that the proposed FIFO based approach outperforms prior techniques for reducing error protection overheads on nearly every metric. We estimate $15-25\%$ and $10-17\%$ reductions in area and power-consumption of a last-level cache respectively, while performance is degraded by only 1.2% on average and by no more than 6%.

The rest of the paper is organized as follows: Section 2 describes related work; Section 3 details our architecture and discusses its benefits and requirements; Section 4 discusses protection capabilities and new tradeoffs enabled by the two-tiered approach; Section 5 compares our method with prior work and provides evaluation of ECC FIFO on an out-of-order processor and projections to *chip multi-processors (CMPs)*; and Section 6 concludes the paper.

2 Background and Related Work

In this section we briefly provide background on code-based error detection and correction mechanisms and prior work on reducing the area and power impact of these mechanisms.

2.1 Information Redundancy

A common solution to address soft errors is to apply error detecting codes (EDC) and error correcting codes (ECC) uniformly across all cache lines. A cache line is extended with an EDC and/or an ECC, and every read or write requires error detection/correction or EDC/ECC encoding respectively. Typically, error detection only codes are simple parity codes, while the most common ECCs use Hamming [7] or Hsiao [8] codes that provide *single bit error correction and double bit error detection* (SEC-DED).

When greater error detection is necessary, *double bit error correcting and triple bit error detecting* (DEC-TED) codes [17], *single nibble error correcting and double nibble error detecting* (SNC-DND) codes [4], and *Reed Solomon* (RS) codes [24] have also been proposed. They are, however, rarely used in cache memories because the overheads of ECC grow rapidly as correction capability is increased [12]. Instead of using these complex codes, multi-bit correction and detection is more commonly achieved by interleaving multiple SEC-DED codes. Table 1 compares the overhead of various ECC schemes. Note that the relative ECC overhead decreases as data size increases.

Table 1: ECC storage array overheads [27].

	SEC-DED		SNC-DND		DEC-TED	
Data bits	check bits	overhead	check bits	overhead	check bits	overhead
16	6	38%	12	75%	11	69%
32	7	22%	12	38%	13	41%
64	8	13%	14	22%	15	23%
128	9	7%	16	13%	17	13%

Examples from Current Processors.

If the first-level cache (L1) is write through and the LLC (e.g., L2) is inclusive, it is sufficient to provide only error detection on the L1 data array because the data is replicated in L2. Then, if an error is detected in L1, error correction is done by invalidating the erroneous L1 cache line and re-fetching the cache line from L2. Such an approach is used in the SUN UltraSPARC-T2 [30] and IBM Power 4 [31] processors. The L2 cache is protected by ECC, and because L1 is write-through, the granularity of updating the ECC in L2 must be as small as a single word. For instance, the UltraSPARC-T2 uses a 7-bit SEC-DED code for every 32 bits of data in L2, an ECC overhead of 22%.

If L1 is write-back (WB), then L2 accesses are at the granularity of a full L1 cache line. Hence, the granularity of ECC can be much larger, reducing ECC overhead. The Intel Itanium processor, for example, uses a 10-bit SEC-DED code that protects 256 bits of data [35] with an ECC overhead of only 5%. Other processors, however, use smaller ECC granularity even with L1 write-back caches to provide higher error correction capabilities. The AMD Athlon [9] and Opteron [11] processors, as well as the DEC Alpha 21264 [5], interleave 8 8-bit SEC-DED codes for every 64-byte cache line to tolerate more errors per line at a cost of 13% additional overhead.

2.2 Reducing Error Correction Overheads

Prior work on reducing on-chip ECC overhead generally break the assumption of uniform protection of all cache lines and either sacrifice protection capabilities, when compared to uniform ECC protection, or utilize different mechanisms to protect clean and dirty cache lines. Kim and Somani [14] suggest parity caching, which compromises error protection to reduce area and energy costs of ECC by only protecting those cache lines that have been most recently used in every cache set. Another scheme in this first category is In-Cache Replication (ICR) [38]. ICR increases error protection for a subset of all cache lines, which are accessed frequently, by storing their replicas in place of cache lines that are predicted to be "dead" and no longer required. Not all cache lines are replicated leading to a potentially higher uncorrectable error rate than with the baseline uniform ECC.

An approach to save energy and latency rather than area was proposed in [25]. The idea is to decouple error detection from error correction and utilize a low-latency and low-cost EDC for reading, while traditional ECC is computed and stored on every write. This ECC is used if an error is detected. A similar idea was taken further in [16], where the SEC-DED ECC portion of clean cache lines is power-gated to reduce leakage power, leaving only parity EDC active. Lines that are clean can be recovered from a deeper level in the memory hierarchy and only error detection is necessary.

Kim [13] proposes a method to decrease area in addition to energy by trading-off performance. His area-efficient scheme allows only one dirty cache line per set in a 4-way set associative cache. If a larger number of lines in a set require ECC (more than one dirty line), a write-back is forced to make space for the new ECC. In our experiments that accurately model the DRAM system (Section 5), we found that this additional memory write traffic can significantly degrade performance. The performance impact can be reduced at an area and energy cost if more ECC-capable lines are provided in each set. We will refer to this generalization scheme where n lines per set have ECC as MAXn.

Essentially, the area-efficient scheme above implements a set-associative cache for storing ECC bits. Other work proposed a fully-associative *replication cache* (R-Cache) that utilizes replication to provide error protection [37]. The R-Cache stores replicas of writes to a L1 cache, and the replicas are accessed on reads to provide detection and correction. If the R-Cache is full, the controller forces a line to be cleaned and written back to L2 so that no data is jeopardized. Although its performance impact is minimal, the R-Cache increases energy consumption and can only be used with small L1 caches due to its fully associative structure.

To tolerate multi-bit burst errors, physical bit interleaving [23] is used in some commercial designs. By interleaving bits from adjacent cache lines in the physical layout, a multi-bit error caused by a single upset appear as single-bit errors in multiple cache lines rather than a single line with a multi-bit error. While this method can reduce ECC overhead for burst errors, it requires additional power for accessing of the other interleaved words, which are not necessary, and additional area and delay due to the long word lines and column multiplexer. Hence, physical interleaving is suitable for small-scale multi-bit errors, with scaling to higher potential error counts results in large performance, area, and power overheads [12]. 2D error coding [12] can tolerate multi-bit errors in a more scalable way. Interleaving horizontal and vertical parity codes together provides higher error protection with only a modest increase in ECC storage. 2D coding, however, forces every write being read-modify-write.

Recently, we suggested Memory-Mapped ECC (MME) [36] that uses a two-tiered error protection similar to this work, but differs in how off-loading to memory is managed. MME stores tier-two error codes as cacheable data in the memory hierarchy, dynamically and transparently partitioning the LLC into data and error codes. MME's impact on performance is low in general, but can be significant because it impacts the caching behavior of the application, as some of the LLC is used for redundant information. This effect is particularly strong when cache blocking is used to closely match the working set with the cache size. ECC FIFO does not suffer from this behavior and we explore its benefits over MME in Section 5.

Finally, Lee et al. proposed eager write-back as a way to improve DRAM bandwidth utilization [15]. It eagerly writes dirty cache lines back to DRAM so that dirty evictions do not contend with demand fetches. Many cache reliability studies (e.g., [13, 16, 36]), including ours, use eager write-back as a way to reduce the average number of dirty cache lines in the LLC.

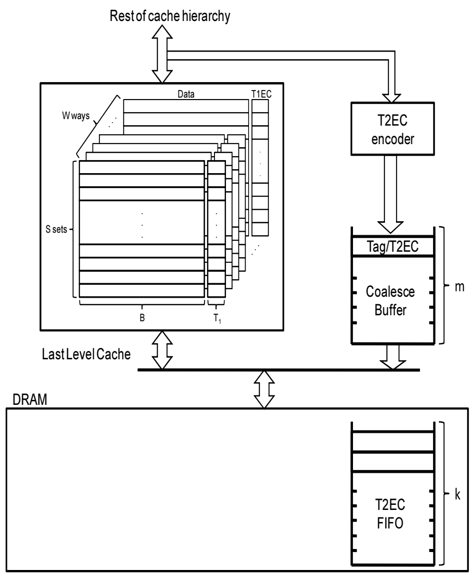


Figure 1: ECC FIFO architecture.

3 ECC FIFO

One of the challenges in designing a large LLC is supporting strong soft and transient error protection while minimizing energy, area, and performance overheads. The most common approaches today extend cache lines with redundant information in the form of ECC resulting in overheads that are proportional to the degree of error protection; a larger number of errors can be tolerated only at the cost of increased redundancy, which increases area and energy leading to lower potential performance. Our ECC FIFO architecture is designed to minimize energy and area costs of on-chip redundancy information and increase performance headroom. Our technique optimizes the common case of detecting errors, and potentially correcting a small number of errors, while providing strong error correction using a unique FIFO structure that off-loads redundant information to main memory. In the rest of this section, we explain our two-tiered protection scheme in detail and evaluate its implementation cost (Section 3.1), followed by a discussion of a potential limitation to protection due to the finite-nature of the FIFO in DRAM and a solution that allows trading off the likelihood of an uncorrectable errors with off-chip storage requirements (Section 3.2).

3.1 High Level Design and Implementation

We use a two-tiered error protection mechanism, in which the mechanisms for detecting errors and correcting errors are essentially split. The *Tier-1 error code* (T1EC) is a low-cost code that is accessed on every read and is designed to be decoded with low latency and energy. We provide dedicated on-chip storage for the T1EC in every cache line and use it exclusively for error detection or for light-weight error correction. The *Tier-2 error code* (T2EC) is more costly and provides the required redundancy for strong error correction. The T2EC, however, is only computed on a dirty write-back into the LLC because clean LLC lines can be recovered from a different level of the memory hierarchy. Just as importantly, the T2EC is only read and decoded on the rare event that the T1EC detects an error that it cannot correct (recall that it is possible to architect T1EC

Table 2: Definitions and nominal parameters used for evaluation.

LLC parameters		
number of sets	S	2048 sets
associativity	W	8 way
line size	B	64B
LLC size	$S\times\ W\times\ B$	1MB
ECC FIFO parameters		
tag size	$\log_2(S\times\ W)$	14 bits
the coalesce buffer size	m	6
T1EC size per cache line	T_1	see Section 4
T2EC size per cache line	T_2	see Section 4
T2EC FIFO size	k	see Section 3.2
Others		
DRAM bandwidth	BW	5.336GB/s
eager write-back period	T_{ewb}	10^6 cycles
processor clock	*clock*	3GHz

with light-weight correction). Therefore, we do not provide dedicated on-chip storage for the T2EC and instead write this redundant information into a FIFO structure allocated in DRAM (Figure 1). Every time a T2EC is generated, the redundant information is pushed onto the T2EC FIFO along with a tag that indicates the corresponding dirty physical LLC line. Thus, when an error is detected by the T1EC, the T2EC FIFO can be searched starting from the newest entry until a matching tag is found and the redundant information can be retrieved. The FIFO is simple to implement and allows us to easily identify the T2EC corresponding to an error with very low management overhead and with efficient transfer of T2EC data to DRAM.

We now discuss the on-chip area overhead, how to write a line into the LLC and generate the redundant information, how to write T2EC data out to DRAM through a coalesce buffer, how to manage the T2EC FIFO, and how to access the LLC and potentially correct any errors.

On-chip Area Overhead

As shown in Figure 1, the on-chip ECC storage overhead is for T1EC only and is equal to $S\times W\times T_1$ bytes, where S is the number of LLC sets, W is the LLC associativity, and T_1 is the number of bytes required for the T1EC corresponding to single LLC line of B bytes (see Table 2). For example, an 8-way interleaved parity T1EC that detects up to a 15-bit burst error requires just 1 byte of storage for a 64-byte cache line. This is a much smaller overhead than in the conventional approach of uniformly providing an 8-way interleaved SEC-DED code requiring 8 bytes for each cache line. Both the 8-way interleaved parity T1EC and the conventional 8-way interleaved SEC-DED have similar burst-error detection capability (up to 16-bit bursts for SEC-DED and 15-bit bursts for the T1EC), although the SEC-DED code can detect a larger number of non-burst bit errors. To provide a matching correction capability the T2EC is an 8-way interleaved SEC-DED that is stored in the off-chip DRAM. We discuss the off-chip storage overhead in Section 3.2.

LLC Write

We assume the cache level preceding the LLC (e.g., L1 if the LLC is L2) is a write-back cache, hence, the LLC is always accessed at a coarse granularity of a cache line. When a cache line is evicted into the LLC, as well as when a line is written into the LLC from DRAM, a T1EC is encoded and written along with the data into the LLC. Because a clean line can be corrected by re-fetching the line from DRAM, a T2EC is encoded only for a dirty line written back into the LLC from the level above. The encoded T2EC is packed with a *tag* , which is a pointer to the corresponding physical data line in the LLC. A tag is composed of the set number and the way number of the cache line so that the T2EC can later be associated with a detected error. The tag requires $\log_2(S\times W)$ bits, e.g., 18 bits for a 16MB LLC with 64B cache lines. The packed tag/T2EC pair is pushed into a coalesce buffer and then written into the FIFO with DRAM-burst granularity. The coalesce buffer is necessary to achieve high DRAM throughput because a single tag/T2EC is only 10.5 bytes in the example above (8 bytes for an interleaved SEC-DED T2EC and 2.5 bytes for the tag), smaller than the minimum burst size of modern DRAM controllers, which are architected for LLC-line granularity (e.g., 64 bytes). Using the parameters above, we can coalesce up to 6 tag/T2EC pairs into a single DRAM write-back.

LLC Read

When a cache line is read from the LLC (fill into the preceding level, or write-back into DRAM), the T1EC is used to detect and potentially correct errors. If the T1EC detects an error that it can correct, the LLC controller immediately corrects the error and re-writes the cache line. If, on the other hand, the error is not correctable by the T1EC, then a copy of the line within the memory hierarchy is re-fetched into the LLC if the line is clean and the T2EC FIFO is used if the line is dirty.

The first step required to correct an error using T2EC is to identify the FIFO entry that contains the T2EC data. The FIFO is searched sequentially to find the tag, which corresponds to the LLC physical line in which the error was found. The search starts from the newest FIFO entry, which can still be in the on-chip coalesce buffer, and proceeds from the current DRAM tail towards the head. Once the most recent matching pair is found, the T2EC (and the T1EC, if needed) attempts to correct the errors. Thus, the worst case penalty of T2EC error correction is the required time to read and compare all tag/T2EC pairs from the FIFO. This takes roughly $k\times\ (B/m)/BW$ seconds, where k is the size of the FIFO in number of pairs, m is the number of pairs in a cache line and BW is the expected throughput of the DRAM channel in bytes/sec (see Table 2). For instance, the worst case correction latency is around 1.66 ms when k is one million entries, m is 6, B is 64 bytes, and BW is 6.4GB/s. This overhead, however, is entirely negligible given the rare occurrence of an error that requires T2EC for correction, which we estimate at one error every 155 days for a 32MB cache in today's technology [27, 29].

T2EC FIFO

We propose to use a large circular FIFO buffer in the main memory space to store the T2EC information. The FIFO has two significant advantages over other T2EC storage options: (i) a FIFO allows for arbitrary coalescing with a trivial and small on-chip buffer that can still maximize DRAM throughput; and (ii) a FIFO is easy to manage in software and provides a clear way to identify the most recent update to a LLC physical line's T2EC data. Information on the FIFO size and base address is stored within the LLC controller and can be set by privileged hypervisor or O/S instructions, or through the BIOS.

One caveat to using a circular buffer is that a T2EC push into the FIFO overwrites the oldest entry. If the physical cache line that corresponds to the overwritten T2EC FIFO

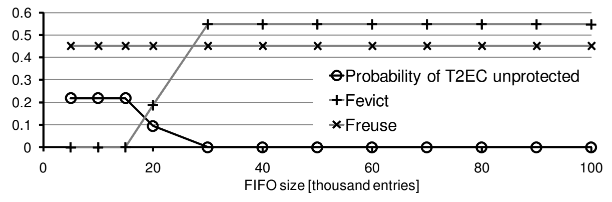


Figure 2: P_{unprot}, F_{reuse}, and F_{evict} in the `lbm` application

entry has not been modified or evicted from the LLC then its T2EC information is overwritten and lost. At that point, the line becomes *T2EC unprotected* and the system will not be able to recover from an error within it.

3.2 T2EC Overwrite and Unprotected Lines

In this section, we analyze the effects of the T2EC FIFO size on T2EC protection capability, which we define as the probability that a T2EC cannot correct the LLC line due to a T2EC overwrite in the circular buffer (P_{unprot}). Even though the buffer is finite, the probability of unprotected lines is small for two main reasons: (i) inherent memory access patterns in applications that limit the lifetime of dirty lines in the LLC through reuse and capacity/conflict evictions; and (ii) limiting dirty line lifetime by programming the LLC controller to periodically clean dirty lines.

3.2.1 Reused and Evicted Dirty Lines

In many applications the natural access pattern leads to relative short lifetime of dirty lines in the LLC. Cache lines are often re-written by the application as new results are generated, rendering earlier computed T2EC data stale and unnecessary. In other cases, LLC lines are evicted to make room for new lines being fetched, and again any existing corresponding T2EC data in the FIFO becomes obsolete. We denote the fraction of T2EC entries that correspond to reused lines in the LLC as F_{reuse} and the fraction of entries corresponding to evicted lines as F_{evict}.

For example, `lbm` from the SPEC CPU 2006 suite [28] has streaming memory access patterns, and hence, the vast majority of dirty lines are written-back into DRAM before a T2EC overwrite occurs. Simulation results (parameters as described in Section 5) show that F_{evict} goes up rapidly as the FIFO size is increased from 15K to 30K entries so that the probability of T2EC unprotected reaches 0 when the FIFO is larger than 30K entries, which is 313KB of DRAM space (Figure 2).

3.2.2 Periodically Cleaning Dirty Lines

Although some applications (such as `lbm`) do not suffer from T2EC unprotected with a reasonable FIFO size, other applications may cause overwrites leading to unprotected lines regardless of the FIFO depth. We utilize the previously proposed *eager write-back* technique [15] to bound the lifetime of a dirty line in the LLC. We explain this in detail and derive a model to analyze the T2EC unprotected probability with eager write-back.

The original eager write-back technique, proposed in [15], opportunistically writes dirty lines into DRAM when a dirty line is in the least-recently used position and a DRAM issue slot is available. This reduces pressure on DRAM when demand fetches are necessary and increases performance. We utilize a more predictable approach, which retains the performance advantages, in which lines are periodically probed

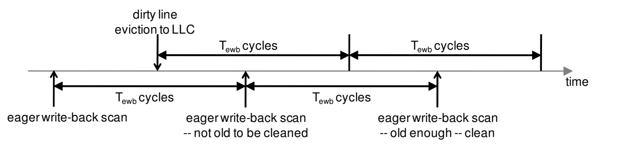


Figure 3: An example time-line of dirty line eviction to the LLC and eager write-back.

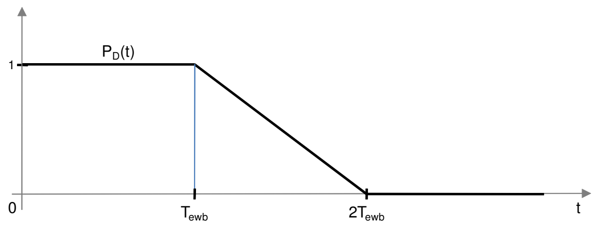
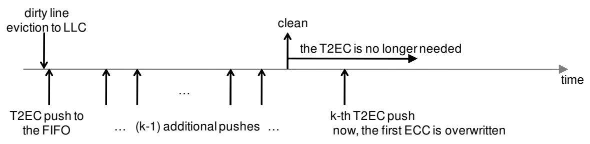


Figure 4: Probability that a dirty cache line remains dirty in time.

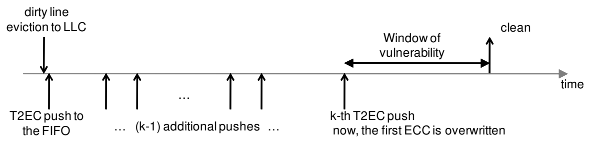
and written back if dirty. This is similar to the policy used in cache error protection studies [13, 16] and decay caches [10]. Our eager write-back implementation scans each cache line with a predetermined period, T_{ewb} cycles, and eagerly writes a dirty line older than the period.

Figure 3 shows an example time-line of a dirty line in the LLC, from the time it is evicted into the LLC until the time it is cleaned by an eager write-back. Each cache line is scanned once per T_{ewb} cycles so that a dirty line is eagerly written back to DRAM within a maximum of $2 \times T_{\text{ewb}}$ cycles after the line is evicted into the LLC. Based on this observation and the fact that the periodic scanning is independent of the eviction times, we can define $P_{\text{D}}(t)$ as the expected probability that a dirty cache line that was written into the LLC at time $t = 0$ remains dirty at time t, as shown in Figure 4. Since eager write-back does not clean dirty lines younger than T_{ewb} cycles, $P_{\text{D}}(t) = 1$ for the first T_{ewb} cycles, after which $P_{\text{D}}(t)$ decreases linearly until it reaches 0 at time $t = 2 \times T_{\text{ewb}}$ cycles. Figure 5 illustrates the two potential time-lines of a dirty LLC line that is neither evicted nor reused with respect to being cleaned by an eager write-back or overwritten in the T2EC FIFO. Figure 5(a) depicts the case that a line is cleaned by an eager write-back before its T2EC data is overwritten and is thus fully protected, whereas Figure 5(b) shows a window of vulnerability opening if the T2EC is overwritten before the line is written back.

To summarize the discussion above, the factors determining whether a line becomes T2EC unprotected are the period of time in which a T2EC entry is required to protect the cache line (the T2EC entry's *valid-time*) and the time for the T2EC to be overwritten in the FIFO (its *overwrite-time*); when a T2EC entry's valid-time is longer than its overwrite-time, the cache line becomes T2EC unprotected. The valid-time is the property of the LLC and memory access pattern; the time it takes for the line to be cleaned by an eager write-back determines the valid-time unless the line is reused or evicted before it is cleaned. The overwrite-time is a function of the FIFO depth and the rate at which dirty lines are written into the LLC (property of memory access pattern and parameters of the caches closer to the core). Because the valid- and overwrite-times are application-specific and dynamic, we model their impact on unprotected lines


dirty line eviction to LLC
clean
the T2EC is no longer needed
...
time
T2EC push to the FIFO
... (k-1) additional pushes ...
k-th T2EC push now, the first ECC is overwritten

(a) a protected case


dirty line eviction to LLC
Window of vulnerability
clean
...
time
T2EC push to the FIFO
... (k-1) additional pushes ...
k-th T2EC push now, the first ECC is overwritten

(b) an unprotected case

Figure 5: Time-lines of dirty lines with T2EC pushes to the FIFO.

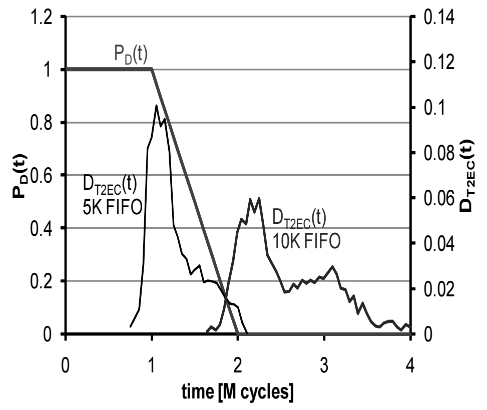


Figure 6: Examples of $D_{\mathbf{T2EC}}(t)$ for 5k and 10k entry FIFOs from omnetpp application overlaid on $P_{\mathbf{D}}(t)$ when $T_{\mathbf{ewb}}$ is 1M cycles.

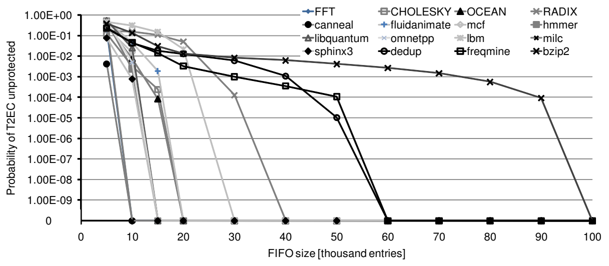


Figure 7: Probability of T2EC unprotected when $T_{\mathbf{ewb}}$ is 1M cycles. A FIFO larger than 40k entries is required only in three applications: dedup, freqmine, and bzip2.

using a probability density function of the overwrite-time, $D_{\mathrm{T2EC}}(t)$. $D_{\mathrm{T2EC}}(t)$ is the distribution of T2EC overwrite-time that a T2EC entry that was computed at time $t = 0$ is overwritten at time $t > 0$. Figure 6 shows examples of $D_{\mathrm{T2EC}}(t)$ from `omnetpp` with different FIFO sizes. A larger FIFO shifts $D_{\mathrm{T2EC}}(t)$ towards longer overwrite times so that eager write-back cleans nearly all the dirty lines before the T2EC is overwritten. We can now write the probability of an unprotected line as in Equation 1.

$$P_{\mathrm{unprot}} = (1 - F_{\mathrm{reuse}} - F_{\mathrm{evict}}) \times \int_0^\infty P_{\mathrm{D}}(t) \times D_{\mathrm{T2EC}}(t)\, dt \quad (1)$$

We used a detailed simulator, described in Section 5, to collect T2EC overwrite-time information for a variety of benchmark applications and a range of FIFO sizes. We then applied the model of Equation 1 and compared the resulting expected fraction of unprotected LLC lines to that from the cycle-accurate simulation. Although we do not present the detailed comparison results in this paper, the model and the simulation agreed to within 1%.

Impact of FIFO Size and Eager Write-Back Period

Figure 7 shows the decrease in probability of T2EC unprotected as the size of the FIFO increases for a range of applications when the eager write-back period is 1M cycles. All but 3 of the applications (`dedup`, `freqmine`, and `bzip2`) are fully protected with a FIFO of 40K entries requiring only 417KB of DRAM storage. The deepest FIFO required to avoid overwrites with $T_{\mathrm{ewb}} = 1M$ cycles is 100K-entries for `bzip2`, requiring about 1MB of DRAM storage. Figure 8 shows the effects of varying T_{ewb} on a few representative applications. All but 4 of the applications evaluated behaved similarly to `OCEAN` of the SPLASH2 suite, requiring a FIFO smaller than 100K entries even with $T_{\mathrm{ewb}} = 5M$ cycles. `RADIX`, `bzip2`, `freqmine`, and `dedup` required deeper buffers, with `dedup` requiring a 220K-entry (2.5MB) buffer with $T_{\mathrm{ewb}} = 5M$ cycles.

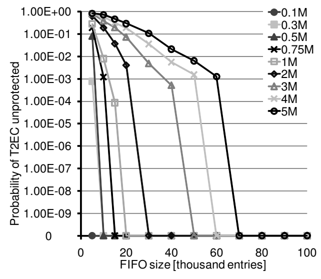
Guaranteeing No T2EC Unprotected Lines.

If the T2EC FIFO is sufficiently large, then eager write-back will always clean a dirty line before the corresponding T2EC is overwritten. With eager write-back, the maximum T2EC valid-time is $2 \times T_{\mathrm{ewb}}$ cycles so we need to choose a FIFO deep enough to make the minimum T2EC overwrite-time longer than the maximum valid-time. The minimum T2EC overwrite-time is k/R, where R is the maximum FIFO fill rate, and a FIFO greater than $2 \times T_{\mathrm{ewb}} \times R$ prevents T2EC unprotected.

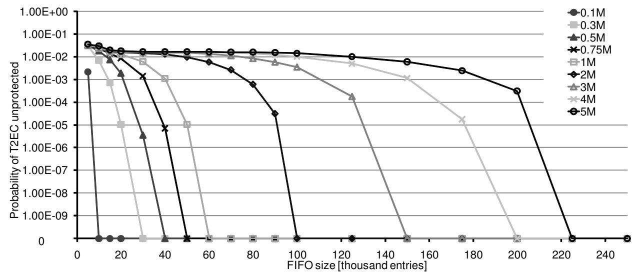
The maximum FIFO fill rate is essentially the maximum possible rate of dirty line writes into the LLC. At worse, every store instruction can cause a dirty line eviction into the LLC. Alternatively, the fill rate may be limited by the total DRAM write bandwidth (to fill the FIFO) or the bandwidth of the LLC controller. In our simulated system (see Table 5), the DRAM write bandwidth sets the tightest bound. The FIFO size required is $2 \times T_{\mathrm{ewb}} \times m/B \times BW/clock$, where *clock* is the clock frequency of eager write-back operations. This corresponds to a FIFO of $346,070$ entries requiring 3.6MB. While this is a fairly large buffer, its size is very small compared to today's main-memory capacities and replaces valuable on-chip area.

Sensitivity to Cache Size

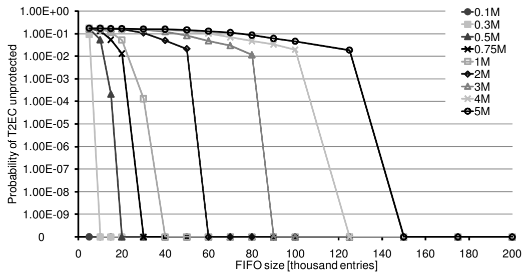
Intuitively, a larger L1 or a smaller L2 will reduce the number of T2EC unprotected lines; a larger L1 reduces the rate of dirty evictions into the LLC, and as a result, T2EC overwrite-time increases. Similarly, a smaller L2 will cause cache lines to be replaced more often, due to capacity constraints, which then decreases T2EC valid-time. We ran simulations varying the L1 size between $32-128$KB and the L2 size between $512-2048$ KB. The behavior of T2EC unprotected is as expected and described above. The overall impact of cache sizes is relatively small and we omit the full results for brevity. The conclusion is that the FIFO depth



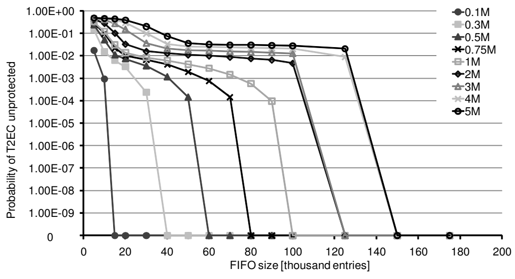
(a) OCEAN



(b) dedup



(c) RADIX



(d) bzip2

Figure 8: Probability of T2EC unprotected varying $T_{\mathbf{ewb}}$.

required to eliminate the possibility of T2EC unprotected lines is unchanged in most applications and only changes by $5 - 10$K in four of the 8 applications we tested.

4 Error Protection Tradeoffs

In this section, we evaluate improvements in protection capabilities and discuss new tradeoffs enabled by the two-tiered approach. One of the main advantages of our two-tiered error protection is the flexibility it provides in choosing the T1EC and T2EC. The most important design considerations of applying ECC FIFO are as follows:

- The T1EC should be chosen to minimize on-chip overhead as it is stored along with every cache line.
- T1EC should be computed with low latency and maximize error detection; correction can be left to T2EC.
- The T2EC size determines overall protection capabilities, but large T2ECs increase DRAM traffic, which degrades the performance of some applications.
- The T2EC FIFO depth and eager write-back period must be selected to limit the number of unprotected lines.

With these considerations, Table 3 describes the configurations and burst error protection capabilities of a number of one-tiered baseline error codes as well as two-tiered error codes. All of the two-tiered codes have identical T2EC size (8B) per 64B cache line, hence the impact on performance will be roughly the same. This leaves the tradeoff to be between the error protection capability (T1EC detection and T2EC correction) and on-chip area (T1EC size).

The PS configuration (parity T1EC and SEC-DED T2EC) allows us to directly compare our architecture to the baseline one-tier 8-way interleaved Hamming SEC-DED codes (S), which are commonly used in related and prior work. The two-tiered PS provides the same bursty error correction capability (up to 8-bit bursts) as the one-tier SEC-DED (S) with reduced on-chip area. The number of detectable errors is different, however. The interleaved parity can detect bursts up to 15 bits long, whereas the conventional interleaved SEC-DED Hamming codes can detect up to 16-bit bursts (8 interleaved double bit errors). The codes also differ in their detection capability for non-burst multi-bit errors, which we evaluate later in this section using error injection.

The PN (parity T1EC and nibble-based SNC-DND T2EC) and PB (parity T1EC and byte-based RS T2EC) configurations can correct even longer bursts. Correction capability is up to 4 nibbles (13–16 bits) and 4 bytes (25–32 bits) respectively, while the on-chip overhead is still very low; even the 32-way parity T1EC of PB requires only 4 bytes, half the space of the baseline (S).

The SD configuration uses a DEC-TED code, which is separable into an 8-bit SEC-DED, with an additional 8 bits providing detection and correction for one more bit error [22]. In our two-tier SD, the 8-bit SEC-DED portion is the T1EC that correct nearly all errors, those that have up to 8-bit bursts, with very low latency. The T2EC DEC-TED, which is constructed from the 8 bits stored on chip in the T1EC and the 8 bits that are retrieved from the T2EC FIFO, is used only in the very rare case of an error that was detected by the T1EC but cannot be corrected by it. Note that the T1EC has only half the on-chip storage overhead of the one-tier DEC-TED configuration (D).

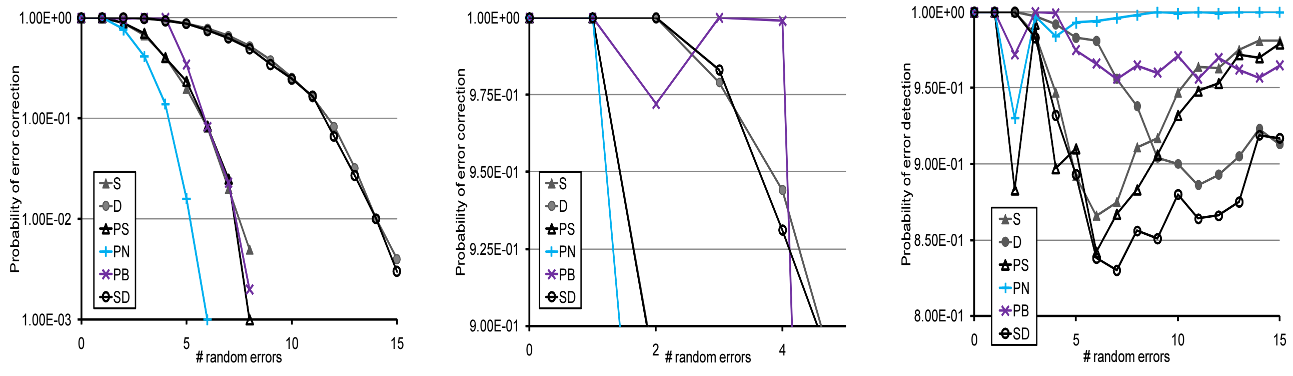
Storage, Energy, and Latency Overheads

The T2EC encoding complexity plays an important role, affecting both energy requirements and latency. The latency of computing T2EC, however, can be hidden, because there are no dependencies on T2EC data. Thus, choosing different T2ECs with regards to the encoding complexity only impacts energy consumption, which is relatively low.

Table 4 compares area overheads, leakage power, and array energy per access of the two-tiered error protection, MAXn, and baseline schemes. We use CACTI 5 [32] to report the properties of a 1MB LLC and related ECC in a 45nm process. Our architecture only dedicates on-chip storage to the T1EC and significantly reduces the area and leak-

Table 3: Burst error protection of baseline and two-tiered error codes assuming 64B cache line (baseline codes are regarded as T1EC without T2EC).

	T1EC		T2EC		T1EC	T1EC	T2EC
	code	size	code	size	burst error correction	burst error detection	burst error correction
baseline							
S	8 way SEC-DED	8B	N/A	N/A	8	16	N/A
D	8 way DEC-TED	16B	N/A	N/A	16	24	N/A
two-tiered error protection							
PS	8 way parity	1B	8 way SEC-DED	8B	N/A	15 bits	8 bits
PN	16 way parity	2B	4 way SNC-DND	8B	N/A	31 bits	4 nibbles
PB	32 way parity	4B	8 bit symbol RS	8B	N/A	63 bits	4 bytes
SD	8 way SEC-DED	8B	8 way DEC-TED	8B	8bits	16 bits	16 bits



(a) Error Correction (log scale) (b) Error Correction (zoomed) (c) Error Detection

Figure 9: Random Error Protection Capabilities.

age overheads of strong ECC. For up to 8-bit burst error correction (S, MAXn, and PS), the two-tiered error protection has much lower area overhead (2.4%) compared to the one-tier SEC-DED baseline (20.4%) and MAXn (3.4% – 6.5% for $n = 1 - 4$). With the increased burst error protection of PN and PB, the area overhead is very small (4.9% for PN, and 9.8% for PB). In addition, 16-bit burst error correction in the two-tiered (SD) has the same overhead of the baseline S, whereas it is only supported with a 41.7% area overhead using a one-tier equivalent code (D).

Random Error Behavior

In order to further assess the error protection capability of the two-tiered codes, we used random-error injection and measured error protection and detection capabilities. For each error scenario presented in Figure 9, we generated 1000 random cache lines and injected each line with random bit flips, including varying the number of erroneous bits from $1 - 15$ for each protection scheme. Figure 9 depicts the probability of errors corrected and detected in the presence of multi-bit random error.

Figure 9(a) shows the probability that the error codes detected and corrected the injected errors. We make three observations regarding the results: (i) in general, two-tiered error protection matches the correction capabilities of its one-tiered counterparts; (ii) in some cases, such as PS vs. S, two tiers provide better protection because errors in both the cache line and the off-chip redundant information are extremely unlikely; and (iii) the symbol-based PN and PB error codes perform relatively poorly with respect to random errors, even though they have superior burst-error correction capabilities (Table 3). Also, note that PB cannot correct all double- and quad-bit errors, even though it uses 1-byte RS, because the parity-based T1EC cannot detect all even-numbered random errors (Figure 9(b)).

Figure 9(c) depicts the rate of correctly detected errors, including errors that are correctable. The behavior of the one-tier and two-tier approaches is quite different, because the stronger T2EC only comes into play if T1EC detects an error. The parity-based T1EC schemes, perform poorly in detecting two random errors, but the greater interleaving degree of PN and PB have very good detection rates for a large number of errors. Both the one-tier and two-tier SEC-DED and DEC-TED based codes perform well with a small number of errors, but cannot handle a larger number of errors. One conclusion we draw is that further study of error detection techniques is necessary if a large number of random errors become a possibility.

5 Evaluation

In this section, we evaluate the cost and performance impact of ECC FIFO. We use the Simics full system simulator [18] with the GEMS [20] toolset. We use the OPAL out-of-order processor model of GEMS to simulate a SPARC V9 4-wide superscalar core with a two-level write-back exclusive cache hierarchy implemented in the Ruby module of GEMS. To accurately account for the impact our technique has on memory bandwidth and performance we integrated DRAMsim [33] into GEMS. We use applications from the SPEC CPU 2006 [28], PARSEC [2], and SPLASH2 [34] benchmark suites that stress the memory subsystem. We do not present results for applications that are not memory intensive as they are not sensitive to the T2EC traffic. We believe that simulating a single core with the given parameters proves the utility of ECC FIFO and conclusions can be drawn on the performance and overheads if implemented within a multi-core processor, and we discuss ECC FIFO in the context of CMPs in Section 5.2.5.

Table 5 describes the baseline system configuration. As mentioned in Section 3.2, we use eager write-back [15] to

Table 4: **Area, leakage power, and array energy per read access.**

	baseline			MAXn			two-tiered error protection			
	no ECC	S	D	MAX1	MAX2	MAX4	PS	PN	PB	SD
Leakage Power (W)	1.4	1.6	1.8	1.5	1.5	1.6	1.4	1.5	1.5	1.6
Array Energy per Read (nJ)	2.0	2.4	2.9	2.1	2.1	2.7	2.1	2.1	2.2	2.4
Area (mm^2)	10.0	12.0	14.1	10.3	10.4	10.6	10.2	10.5	10.9	12.0
ECC area overhead (%)	-	20.7	41.7	3.4	4.2	6.5	2.4	4.9	9.8	20.7

Table 5: **Simulated system parameters.**

Processor Core	SPARC V9 ISA 4-wide superscalar (3GHz)
L1 Cache	split I/D cache each 64KB 2-way set associative 64B cache lines write-back 1 cycle latency
L2 Cache	unified 1MB cache 8-way set associative L1 exclusive 64B cache lines eager write-back ($T_{\text{ewb}} = 10^6$ cycle) L2 latency is 12 cycles, including ECC encoding/decoding
DRAM	single channel DDR2 DRAM (5.336GB/s) 667MHz 64-bit data bus open page Read and Instruction Fetch First

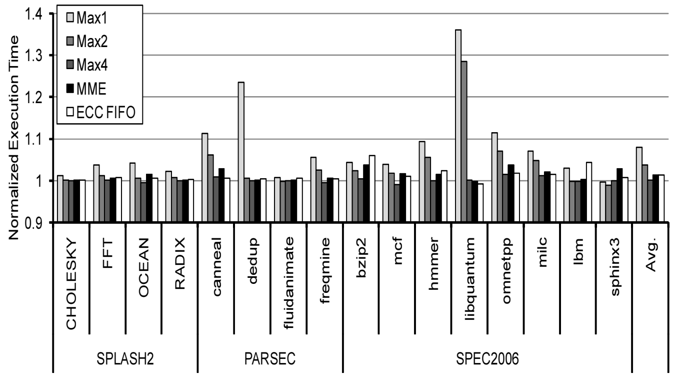


Figure 10: Normalized execution time.

limit the lifetime of dirty lines in the LLC and to improve the baseline performance; eager write-back improves the baseline performance by 6–10% in most applications and 26% in `libquantum`. In `lbm` and `bzip2`, however, eager write-back degrades performance significantly and we disable it for the prior-work techniques, as it is not needed for correctness or protection. We also disable eager write-back for ECC FIFO in `lbm`, because the probability of T2EC unprotected is negligible, even without eager write-back. We keep eager write-back at 1M cycles for `bzip2` with ECC FIFO to maintain a low T2EC unprotected rate. Note that we applied different eager write-back periods for `fluidanimate` (0.25M) and `libquantum` (1.2M) to optimize the performance of the baseline scheme.

5.1 Workloads

We use a mix of the SPLASH2 [34], PARSEC [2], and SPEC CPU 2006 [28] workloads. We concentrated on applications with large working sets that stress the memory hierarchy and highlight the differences between our architecture and prior work. For the detailed cycle-based simulations, we ran the applications from SPLASH2 and PARSEC to completion using a single thread and small to medium problem sizes: tk15.O for `CHOLESKY`; 64K samples for `FFT`; a 514×514 grid for `OCEAN`; 1M samples for `RADIX`, and the simsmall inputs for all PARSEC applications. For the SPEC CPU 2006 workloads, we used SimPoint [6] to extract five representative regions of 200M instructions and report their weighted sums.

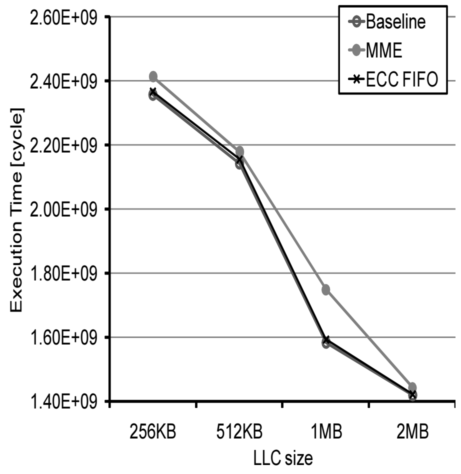


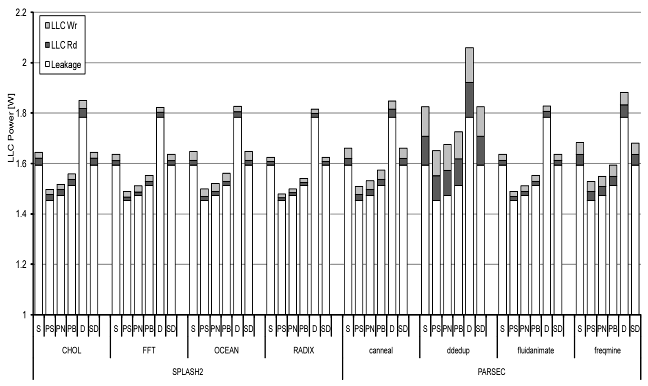
Figure 11: `OCEAN` 258×258 **performance of the baseline, MME, and ECC FIFO varying LLC size.**

5.2 Performance Results and Analysis

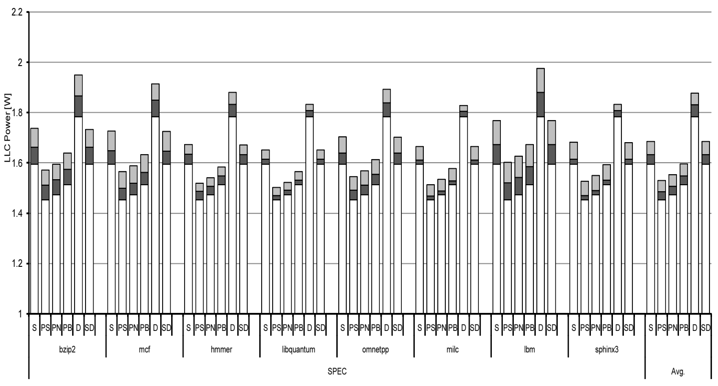
This section contains multiple subsections in which we evaluate overall performance results and compare the impact of ECC FIFO to that of MAXn schemes [13] and Memory-Mapped ECC [36]; analyze LLC power consumption in various error coding schemes discussed in Section 4 and show the potential of saving LLC power consumption by off-loading the T2EC to DRAM; evaluate the impact of additional DRAM traffic needed for off-loading the T2EC to DRAM, which only degrades performance if it competes with demand fetches; and analyze sensitivity to DRAM bandwidth to assess the performance impact of ECC FIFO on CMPs, where DRAM bandwidth is more scarce.

5.2.1 Impact on Performance

Figure 10 compares the execution times of MAXn, MME, and ECC FIFO normalized to the execution time of the baseline configuration. As shown, the impact on application performance of ECC FIFO is minimal, with an average performance penalty of less than 1.2% and a maximum degradation of just under 6% in `bzip2` and 4% in `lbm`. The degradation in `bzip2` is not from ECC traffic, but rather because of the use of eager write-back to bound the lifetime of dirty lines (recall that eager write-back is enabled only with ECC FIFO for `bzip2`). ECC FIFO without eager write-back degrades the performance of `bzip2` by only 1.3%, but requires a larger FIFO of 2MB in DRAM. These results are encouraging because of the benefits our scheme provides

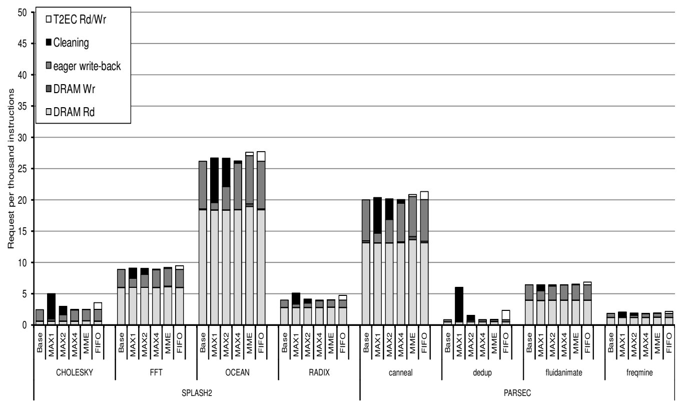


(a) SPLASH2 and PARSEC

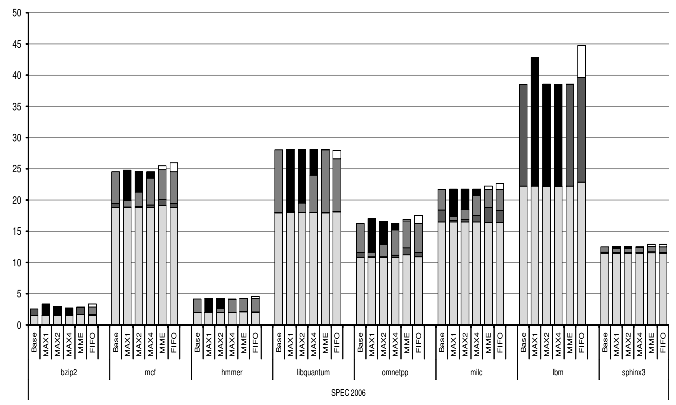


(b) SPEC CPU 2006

Figure 12: LLC power consumption estimated with CACTI 5 for a 1MB, 8-way, 64B cache line LLC in 45nm technology.



(a) SPLASH2 and PARSEC



(b) SPEC CPU 2006

Figure 13: DRAM traffic comparison.

in reducing on-chip area and leakage power by minimizing dedicated on-chip ECC storage. The MAXn technique, on the other hand, requires significant tradeoff between performance loss and area gains. MAX1 averages over 8% performance loss with several applications experiencing 10 - 36% degradation, MAX2 degrades 8% on average while `libquantum` is significantly degraded by 30%, and MAX4 performs only slightly better than ECC FIFO and MME. MAX4 requires, however, much more area than MME and ECC FIFO as shown in Table 4.

The anomalous behavior of `OCEAN`, `fluidanimate`, `freqmine`, `bzip2`, `mcf`, `libquantum`, and `sphinx3`, where the performance of MAX2 and MAX4 is slightly better than baseline, is a result of increased eager write-backs for lines that are being forced clean in order to guarantee protection. We also note that applications with small working sets, such as WATER-NSQUARED (SPLASH2) and blackscholes (PARSEC) experience no performance drop with ECC FIFO, as well as with MAXn and MME, and we do not report their results. All other applications from the benchmark suites, which we do not discuss in this paper, are also unaffected by the two-tiered approach.

5.2.2 Comparison with MME

MME's performance degradation is 1.3% on average (Figure 10), which is comparable to that of the ECC FIFO (1.2% on average). Note that the area / error protection tradeoff in MME is the same as that of ECC FIFO because both schemes use two-tiered error protection and offloading to DRAM. MME, however, can impact performance significantly when the working set of an application closely matches the LLC size. Figure 11 compares the execution times of `OCEAN` with a 258×258 grid on the baseline, MME, and ECC FIFO configurations as the LLC size is varied. As the LLC size grows from 512KB to 1MB, the baseline performance is improved by 26%. ECC FIFO's performance penalty is consistently less than 1% across all LLC sizes. MME, however, degrades performance significantly (10%) with a 1MB LLC since the effective LLC size is reduced by sharing the LLC between data and error codes.

5.2.3 Impact on LLC

ECC FIFO bypasses the LLC to accesses T2EC information, and thus does not change the caching behavior and only impacts the operation of the DRAM system. ECC FIFO, however, reduces power consumption of the LLC in two ways: decreased energy per access and reduced leakage power.

Figure 12 compares LLC power consumption of the baseline (S and D) and the ECC FIFO (PS, PN, PB, and SD) estimated using CACTI 5 [32] for the cache parameters in Table 5 in a 45nm technology. PS saves 9% of LLC power consumption compared to S, while SD, PN, and PB consume

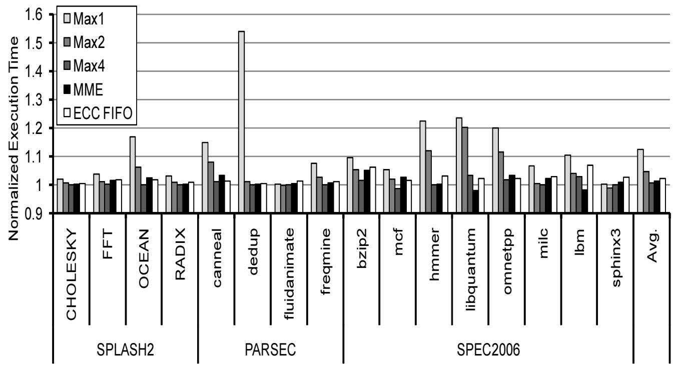


Figure 14: Normalized execution time (2.667 GB/s DRAM bandwidth).

10%, 17%, and 15% less power than D, respectively. Note that PS and SD provide similar error protection as S and D, respectively, and PN and PB can protect against even longer error bursts, even compared to D.

5.2.4 Impact on DRAM Traffic

Figure 13 shows the total DRAM traffic broken down into components of data reads and writes of LLC lines (*DRAM Rd* and *DRAM Wr*), eager write-backs of cache lines (*eager write-back*), MAXn writes for cleaning lines (*Cleaning*), and T2EC traffic of MME and ECC FIFO (*T2EC Rd/Wr*). Although MAXn does not increase overall memory traffic by much in most cases, the type of traffic and its timing differ significantly from the baseline. Unlike eager write-back traffic, which is scheduled during idle DRAM period [15], MAXn's cleaning writes compete with demand fetches and degrade performance as shown in Figure 10. More detailed analysis on how MAXn's cleaning traffic degrades performance can be found in [36]. Compared to the baseline, ECC FIFO increases memory traffic by 9% on average, which is much larger than MME's traffic penalty (2%). It is, however, not critical to performance in that T2EC write-back is one-way traffic and can be scheduled during idle DRAM period.

5.2.5 Multi-Core Considerations

We have so far discussed and evaluated ECC FIFO in the context of a single core and now briefly discuss multi-core and multi-processor implications. ECC FIFO is easily integrated with any cache coherence mechanism because the T2EC FIFO in DRAM is inherently private. On the other hand, the increased traffic of ECC FIFO may hurt the performance due to relatively lower DRAM bandwidth per core in CMPs. We evaluated ECC FIFO in a system with low DRAM bandwidth of only 2.667 GB/s, which is half the bandwidth of the baseline system. As Figure 14 shows, the relative performance of ECC FIFO is not sensitive to memory bandwidth.

6 Conclusions and Future Work

This paper presents the ECC FIFO that exploits flexible two-tiered error protection and off-loads the overheads of strong error codes to DRAM to achieve both low cost and high reliability in a LLC. With the minimized dedicated on-chip ECC storage, both the leakage power and energy per access are reduced leading to 10 - 17% LLC power reduction in addition to 15 - 25% of area saving, while performance is degraded only by 1.2% on average and no more than 6%. It is achieved with a simple and unique FIFO structure that off-loads ECC storage overheads to DRAM, and outperforms prior techniques. ECC FIFO performs better than MME especially when the reduced effective cache size of MME hurts performance.

The two-tiered error protection used in this study offers great design flexibility, and several two-tiered error codes that provide strong error protection with reduced on-chip overheads are presented. We have also shown that the probability of T2EC unprotected due to FIFO overwrite can be managed to be very small with a reasonably sized FIFO. In future work, we will explore how to reduce off-chip traffic by buffering or caching T2EC and how to manage eager write-back and the FIFO adaptively with regards to memory access patterns and system resources.

7 References

[1] H. Ando, K. Seki, S. Sakashita, M. Aihara, R. Kan, K. Imada, M. Itoh, M. Nagai, Y. Tosaka, K. Takahisa, and K. Hatanaka. Accelerated Testing of a 90nm SPARC64 V Microprocessor for Neutron SER. In *Proc. the IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, April 2007.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical Report TR-811-08, Princeton Univ., January 2008.

[3] L. Chang, D. M. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, R. K. Montoye, L. Sekaric, S. J. McNab, A. W. Topol, C. D. Adams, K. W. Guarini, and W. Haensch. Stable SRAM Cell Design for the 32nm Node and Beyond. In *Digest of Technical Papers of Symp. VLSI Technology*, June 2005.

[4] C. L. Chen and M. Y. Hsiao. Error-correcting Ccodes for Semiconductor Memory Applications: A State-of-the-art Review. *IBM J. Research and Development*, 28(2):124–134, March 1984.

[5] Digital Equipment Corporation. *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.

[6] G. Hamerly, E. Perelman, J. Lau, and B. Calder. SimPoint 3.0: Faster and More Flexible Program Analysis. In *Proc. the Workshop on Modeling, Benchmarking and Simulation*, June 2005.

[7] R. W. Hamming. Error Correcting and Error Detecting Codes. *Bell System Technical J.*, 29:147–160, April 1950.

[8] M. Y. Hsiao. A Class of Optimal Minimum Odd-weight-column SEC-DED codes. *IBM J. Reserach and Development*, 14:395–301, 1970.

[9] J. Huynh. *White Paper: The AMD Athlon MP Processor with 512KB L2 Cache*, May 2003.

[10] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In *Proc. the 28th Ann. Int'l Symp. Computer Architecture (ISCA)*, June-July 2001.

[11] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 23(2):66–76, March-April 2003.

[12] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In *Proc. the 40th IEEE/ACM Int'l Symp. Microarchitecture (MICRO)*, December 2007.

[13] S. Kim. Area-Efficient Error Protection for Caches. In *Proc. the Conf. Design Automation and Test in Europe (DATE)*, March 2006.

[14] S. Kim and A. K. Somani. Area Efficient Architectures for Information Integrity in Cache Memories. In *Proc. the 26th Ann. Int'l Symp. Computer Architecture (ISCA)*, May 1999.

[15] H.-H. S. Lee, G. S. Tyson, and M. K. Farrens. Eager Writeback - A Technique for Improving Bandwidth Utilization. In *Proc. the 33rd annual IEEE/ACM international Symp. Microarchitecture (MICRO)*, November-December 2000.

[16] L. Li, V. S. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Soft Error and Energy Consumption Interactions: A Data Cache Perspective. In *Proc. the Int'l Symp. Low Power Electronics and Design (ISLPED)*, August 2004.

[17] S. Lin and D. J. C. Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[18] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. SIMICS: A Full System Simulation Platform. *IEEE Computer*, 35:50–58, February 2002.

[19] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. Characterization of Multi-Bit Soft Error Events in Advanced SRAMs. In *Technical Digest of the IEEE Int'l Electron Devices Meeting (IEDM)*, December 2003.

[20] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *SIGARCH Computer Architecture News (CAN)*, 33:92–99, November 2005.

[21] K. Osada, K. Yamaguchi, and Y. Saitoh. SRAM Immunity to Cosmic-Ray-Induced Multierrors based on Analysis of an Induced Parasitic Bipolar Effect. *IEEE J. Solid-State Circuits*, 39:827–833, May 2004.

[22] A. M. Patel and M. Y. Hsiao. An Adaptive Error Correction Scheme for Computer Memory System. In *Proc. the Fall Joint Computer Conf., part I*, December 1972.

[23] N. Quach. High Availability and Reliability in the Itanium Processor. *IEEE Micro*, 20(5):61–69, Sept.-Oct. 2000.

[24] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *J. Soc. for Industrial and Applied Math.*, 8:300–304, June 1960.

[25] N. N. Sadler and D. J. Sorin. Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache. In *Proc. the Int'l Conf. Computer Design (ICCD)*, October 2006.

[26] N. Seifert, V. Zia, and B. Gill. Assessing the Impact of Scaling on the Efficacy of Spatial Redundancy based Mitigation Schemes for Terrestrial Applications. In *Proc. the IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, April 2007.

[27] C. Slayman. Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations. *IEEE Trans. Device and Materials Reliability*, 5:397– 404, September 2005.

[28] Standard Performance Evaluation Corporation. SPEC CPU 2006. http://www.spec.org/cpu2006/, 2006.

[29] J. Standards. JESD89 Measurement and Reporting of Alpha Particles and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices, JESD89-1 System Soft Error Rate (SSER) Method and JESD89-2 Test Method for Alpha Source Accelerated Soft Error Rate, 2001.

[30] Sun Microsystems Inc. *OpenSPARC T2 System-On-Chip (SOC) Microarchitecture Specification*, May 2008.

[31] J. M. Tendler, J. S. Dodson, J. S. F. Jr., H. Le, and B. Sinharoy. POWER4 System Microarchitecture. *IBM J. Research and Development*, 46(1):5–25, January 2002.

[32] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. Technical report, HP Laboratories, April 2008.

[33] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, and B. Jacob. DRAMsim: A Memory-System Simulator. *SIGARCH Computer Architecture News (CAN)*, 33:100–107, September 2005.

[34] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. the 22nd Ann. Int'l Symp. Computer Architecture (ISCA)*, June 1995.

[35] J. Wuu, D. Weiss, C. Morganti, and M. Dreesen. The asynchronous 24MB On-Chip Level-3 Cache for a Dual-Core Itanium®-Family Processor. In *Proc. the Int'l Solid-State Circuits Conf. (ISSCC)*, February 2005.

[36] D. H. Yoon and M. Erez. Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches. In *Proc. the 36th Int'l Symp. Computer Architecture (ISCA)*, June 2009.

[37] W. Zhang. Replication Cache: A Small Fully Associative Cache to Improve Data Cache Reliability. *IEEE Trans. Computer*, 54(12):1547 –1555, December 2005.

[38] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. ICR: In-Cache Replication for Enhancing Data Cache Reliability. In *Proc. the Int'l Conf. Dependable Systems and Networks (DSN)*, June 2003.