Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches

Doe Hyun Yoon
Electrical and Computer Engineering Department
University of Texas at Austin
Austin, TX, 78712
doehyun.yoon@gmail.com

Mattan Erez
Electrical and Computer Engineering Department
University of Texas at Austin
Austin, TX, 78712
mattan.erez@mail.utexas.edu

ABSTRACT

This paper presents a novel technique, *Memory Mapped ECC*, which reduces the cost of providing error correction for SRAM caches. It is important to limit such overheads as processor resources become constrained and error propensity increases. The continuing decrease in SRAM cell size and the growing capacity of caches increases the likelihood of errors in SRAM arrays. To address this, redundant information can be used to correct a value after an error occurs. Information redundancy is typically provided through error-correcting codes (ECC), which append bits to every SRAM row and increase the array's area and energy consumption. We make three observations regarding error protection and utilize them in our architecture: (1) much of the data in a cache is replicated throughout the hierarchy and is inherently redundant; (2) error-detection is necessary for every cache access and is cheaper than error correction, which is very infrequent; (3) redundant information for correction need not be stored in high-cost SRAM.

Our unique architecture only dedicates SRAM for error detection while the ECC bits are stored within the memory hierarchy as data. We associate a physical memory address with each cache line for ECC storage and rely on locality to minimize the impact. The cache is dynamically and transparently partitioned between data and ECC with the fraction of ECC growing with the number of dirty cache lines. We show that this has little impact on both performance (1.3% average and $< 4\%$) and memory traffic (3%) across a range of memory-intensive applications.

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 Memory Mapped ECC – 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 several important observations and is unique in off-loading the storage requirements of error correction to low-cost off-chip DRAM. We store error-correction bits as addressable data within the memory hierarchy itself instead of in a dedicated SRAM array. This is a general technique that can be used to dynamically and transparently offload any meta-data that is associated with a physical cache line to higher levels of the storage hierarchy. We also offer a generalization of energy-reducing non-uniform error correction techniques using a tiered approach; 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 with the continuing increase in on-chip cache size, soft error propensity grows as well. This increase in *soft error rate* (SER) is the result of a decrease in the critical charge required to flip a stored bit (Q_{crit}) as the physical dimensions of the SRAM devices shrink. There are two main mechanisms in which a soft error occurs. The first mechanism is a result of charge that surpasses Q_{crit} accumulating in the device due to a particle strike (typically a neutron in the form of a cosmic ray). While the likelihood of a particle striking a particular SRAM cell is proportional to the SRAM cell size, the decreasing Q_{crit} offsets this effect leading to a roughly constant SER per SRAM cell of 10^{-3}FIT/bit [22, 31, 29]. The overall capacity of the LLC, however, is growing, leading to a rapid rise in the likelihood of soft errors in the LLC. The second mechanism relates to supply noise, which can also overcome the potential difference associated with Q_{crit} and lead to a bit flip. The increase in variability and the need to lower supply voltage (V_{DD}) on top of smaller Q_{crit} in deep sub-micron fabrication compounds the soft error problem. Another trend in SRAM

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

soft errors is the increased likelihood of multi-bit errors in the array, partially because many memory cells can fall under the footprint of a single energetic particle strike leading to a burst of consecutive errors [22, 24, 28].

Because of these trends, the need to account for high LLC SER and also tolerate multi-bit errors in SRAM arrays is becoming acute, and SRAM cell stability will be a primary concern for future technologies [4]. Researchers have already observed up to 16-bit errors in SRAM arrays and are predicting potentially higher counts in the future [1, 22, 24]. The more powerful error protection mechanisms required to correct large numbers of bit flips come at a cost of storage of the redundant information as well as increased energy requirements. We aim to reduce the overheads of providing increased protection against soft errors in large SRAM arrays. We base our Memory Mapped ECC efficient error protection architecture on five important observations:

- While SER is increasing and cannot be ignored at any memory hierarchy level, error events are still expected to be extremely rare when compared to the processor cycle time. For instance, the mean time to failure (MTTF) of a 32MB cache is around 155 days assuming 10^{-3}FIT/bit.

- Every read from the LLC requires error detection to ensure that no error has occurred. Error detection latency, therefore, is critical to application performance and error detection energy is important for overall power.

- 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, is unimportant as long as the write throughput can be supported. Additionally, write operations are typically less common than read operations and the complexity to compute error correction information can be higher.

- Given that SER is low, the latency and the complexity of error correction is not important. Even an error correction latency of 10ms is acceptable since it would occur once every several weeks or months on a given processor.

- 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, which can be done with lower cost than error correction.

We combine the observations above into an architecture that decouples error detection from error correction and that can store the redundant information required for correction in low-cost off-chip DRAM. This unique organization leads to several important contributions:

1. We develop an architecture that stores the encoded redundant information required for error correction within the memory hierarchy rather than in costly dedicated SRAM, reducing both the area and the energy overheads of error protection.

2. We apply low-cost error detection and/or correction uniformly to all LLC lines to ensure we can detect errors reliably and match or exceed the capabilities of traditional architectures. We do not require that correction be low latency or simple in all cases and utilize strong error-correcting codes to provide the desired protection level. We can afford much more complex and costly error correction because we do not dedicate on-chip storage resources to the redundant bits. This is a generalization of previously suggested decoupled detection/correction approaches [27, 18, 14].

3. We provide a single, extensible, and general method to store and manipulate error-correction data. We show how Memory Mapped ECC dynamically adapts the on-chip storage requirements for error correction to minimize the impact on application performance.

In this paper we focus on the overhead reductions enabled by Memory Mapped ECC and defer discussion of potentially improved protection to future work. We evaluate our architecture using a combination of full-system cycle-based simulation of challenging SPLASH2, PARSEC, and SPEC CPU 2006 benchmarks, and augment the simulation results using PIN-based emulation for large datasets. We show that Memory Mapped ECC outperforms prior techniques for reducing error-protection overheads on nearly every metric. We estimate 15% and 8% reductions in area and power-consumption of a last-level cache respectively, while performance is degraded by only 1.3% on average and by no more than 4%.

The rest of the paper is organized as follows: Section 2 describes related work, including circuit- and device-level techniques in addition to methods that utilize information redundancy; Section 3 details our architecture and discusses its benefits and requirements as well as comparing our method with prior work; Section 4 provides the evaluation of Memory Mapped ECC in the context of the LLC of an out-of-order processor and Chip Multi-Processors (CMPs); and Section 5 concludes the paper.

2 Related Work

In this section we briefly provide background on circuit and device level techniques for addressing the soft error problem, code-based error detection and correction mechanisms, and prior work on reducing the area and power impact of these mechanisms.

2.1 Circuit- and Device-Level Techniques

In this paper we focus on microarchitecture level mechanisms to reduce SER, but there has also been extensive work on SRAM reliability using process, layout, and circuit techniques, which typically have high overhead for significant SER reductions. The least intrusive process level method is deep N-well technology, which uses a triple-well process in which NMOS devices are constructed inside P-wells, which are themselves inside deep N-wells [6]. The Deep N-wells provide isolation from α-particles and neutron cosmic rays and reduce the failure rate of SRAMs by a factor of approximately $1.5 - 2.5$.

A more significant reduction in SER can be achieved by modifying the SRAM cell layout to increase Q_{crit}. This is done by adding poly-diffusion overlaps to the critical nodes [6], improving SER by a factor of 20 but with a 40% area overhead and a penalty of $6 - 8\%$ in latency.

On the circuit side, hardened SRAM cells have been suggested that use a larger number of transistors to achieve

Table 1: ECC storage array overheads [29].

	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%

reduction in soft-error propensity and more robust operation even at low V_{DD} values. For example, 8T cells [4], 10T cells [3], and Schmidt Trigger (ST) based 10T cells [16] can significantly improve reliability, but at the cost of increased access latency and SRAM cell area (8T - 30%, 10T - 60%, and ST 10T - 100% [36]).

2.2 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, then 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 [9] or Hsiao [10] 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 [19], single nibble error correcting and double nibble error detecting (SNC-DND) codes [5], and Reed Solomon (RS) codes [26] have also been proposed. They are, however, rarely used in cache memories because the overheads of ECC circuits and storage grow rapidly as correction capability is increased [13]. 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.

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 only provide error detection on the 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 [32] and IBM Power 4 [33] 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 [38] 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 [11] and Opteron [12] processors, as well as the DEC Alpha 21264 [7], 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.

2.3 Reducing Error Correction Overheads

In this subsection we describe work that is closely related to our architecture. We defer detailed comparison and discussion to later sections (Section 3.4 and Section 4.3.1 respectively). 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 [15] 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) [40]. 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 [27]. 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 [18], 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 [14] proposes a method to decrease area in addition to energy by trading-off performance. Their 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 4.3.1), 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* that utilizes replication to provide error protection. The R-Cache stores replicas of writes to a L1 cache [39], 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.

Finally, Lee et al. proposed eager write-back as a way to improve DRAM bandwidth utilization [17]. 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., [14, 18]), including ours, use eager write-back as a way to reduce the average number of dirty cache lines in the LLC.

Table 2: Possible T1EC/T2EC codes assuming 64B cache line.

T1EC		T2EC		T1EC	T1EC	T2EC
code	size	code	size	burst error correction	burst error detection	burst error correction
8 way parity	1B	8 way SEC-DED	8B	N/A	15 bits	8 bits
16 way parity	2B	4 way SNC-DND	8B	N/A	31 bits	4 nibbles
32 way parity	4B	8 bit symbol RS	8B	N/A	63 bits	4 bytes
8 way SEC-DED	8B	8 way DEC-TED	8B	8bits	16 bits	16 bits

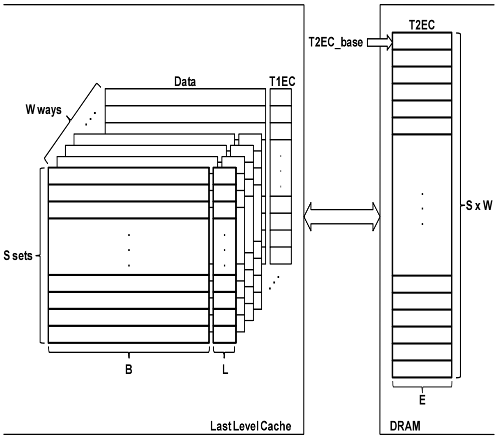


Figure 1: Memory Mapped ECC organization (see Table 3 for definitions of symbols).

3 Memory Mapped ECC

Our Memory Mapped ECC architecture minimizes the hardware energy and area costs beyond prior work while having minimal impact on performance. Our technique stores the ECC bits within the memory hierarchy itself rather than in dedicated storage. Further, we provide a generalization of the non-uniform techniques described above through a tiered error correction structure that is naturally implemented using our method. Memory Mapped ECC uses a two-tiered protection mechanism to ensure LLC data integrity. The *Tier-1 error code* (T1EC) is a low-cost code that can be decoded with low latency and energy because it must be accessed on every read. The *Tier-2 error code* (T2EC) provides the required strong error correction capability and is much more costly to compute and store. The T2EC, however, is only computed on a dirty write-back into the LLC. More importantly, it is only read and used on the rare event that the T1EC detects an error that it cannot correct. Therefore, we do not provide storage for the T2EC along with every cache line and instead write the T2EC bits into the memory system as addressable data. Thus the information can be cached in the LLC and eventually stored in low-cost off-chip DRAM. Figure 1 illustrates how T1ECs and T2ECs are organized in the LLC and DRAM.

Storing the T2EC bits as cacheable data, rather than in dedicated storage, offers several important advantages. First, we avoid dedicating costly SRAM to store ECC only, while at the same time, we do not limit the number of dirty lines in a cache set or the cache overall. By allowing the redundant information to be stored in DRAM and cached on chip, we transparently and dynamically adjust the on-chip T2EC storage to the number of dirty cache lines using the existing cache management mechanism. Thus we eliminate all SRAM leakage and area overheads of dedicated T2EC storage, improving on the savings suggested in the past [14, 18].

The second main advantage is the generality of our technique and the flexibility it provides in choosing the T2EC. In this paper we evaluate a T1EC/T2EC that uses 8 way interleaved parity codes for the first-tier T1EC and an 8 way interleaved SEC-DED second-tier code. Thus, the T1EC and T2EC are 1 byte and 8 bytes per 64 byte cache line respectively. The interleaved parity codes can detect multiple adjacent bit errors and minimize on-chip overheads and error detection latency. Once errors are detected, the off-chip, cached SEC-DED codes (T2EC) correct the errors. This configuration allows us to directly compare our architecture to 8-way interleaved Hamming SEC-DED codes commonly used in prior work. The combination of interleaved parity and SEC-DED codes as T1EC and T2EC provide the same bursty error correction capability (up to 8-bit bursts) as the conventional Hamming codes with reduced on-chip area. The number of detectable unrecoverable errors (DUEs), however, is different. The interleaved parity T1EC can detect bursts up to 15 bits long, whereas the conventional interleaved SEC-DED Hamming code 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, but such errors are much less common.

With our architecture, however, it is simple to accommodate stronger protection. We briefly summarize a few possible T1EC/T2EC combinations providing a range of different error protection capabilities in Table 2. Since all configurations have identical T2EC size (2B), the impact on performance will be roughly the same as the results that we present in Section 4. Hence, the tradeoff is between error protection capability and on-chip area, in which T1EC size represents on-chip area overhead. Encoding complexity also plays a role in cost, but we leave discussing this to future work. Note that the size of even the 32-way interleaved parity T1EC (4B) is still much less than the size of conventional interleaved Hamming codes (8B), while the combination of T1EC/T2EC can protect more errors. Each of the 8 interleaved codes of the T2EC of the last configuration in Table 2 is a DEC-TED code with separable SEC-DED property [25], in which the first 8 bits of the 16-bit DEC-TED code can provide SEC-DED capabilities; Thus, the T1EC can correct nearly all errors, those that have up to 8-bit bursts, with a very low latency overhead. The T2EC, which constructs the DEC-TED code from the 8 bits stored on-chip in the T1EC and the 8 bits that are handled by Memory Mapped ECC is used only in cases of a DUE in the T1EC, which are extremely rare.

We now describe how to integrate Memory Mapped ECC within the memory hierarchy including interfaces with the DRAM memory namespace and the LLC controller, followed by a discussion of the impact on area, energy, and performance.

3.1 T2EC in DRAM

Memory Mapped ECC requires a region of the memory namespace (the T2EC region) to be allocated for T2EC storage.

Table 3: Definitions and nominal parameters used for evaluation.

LLC parameters	C	LLC size in bytes	1MB
	W	LLC associativity	8
	S	number of LLC sets	2048
	B	LLC line size in bytes	64
ECC parameters	L	T1EC size in bytes (per LLC line)	1
	E	T2EC size in bytes (per LLC line)	8
Address mapping	*T2EC_addr(x)*	DRAM address in which the T2EC bits protecting address x are stored	
	T2EC_base	base pointer to T2EC in DRAM	
	way(x)	the way within a set in which address x is cached	
	set(x)	the set within the LLC in which address x is cached	
	T2EC line	multiple T2ECs that are mapped to a single LLC line	8

Our current implementation is to map this region to physical DRAM addresses and avoid address translation when reading and writing T2EC. An on-chip register, (*T2EC_base*), points to the start of the T2EC array, which is only $\frac{E\times C}{B}$ bytes in size, where E is the number of bytes of T2EC per cache line, C is the total cache size, and B is the number of bytes in a LLC line (Table 3). For example, using interleaved SEC-DED codes, the T2EC array is just 128KB of physical DRAM for every 1MB of a LLC with 64-byte lines. Each physical LLC line is associated with a T2EC in physical memory, as follows: (See Table 3 for notations)
$T2EC\_addr(x) = T2EC\_base + (way(x) \times S + set(x)) \times E$.
Using this mapping for the interleaved SEC-DED T2EC on 64-byte LLC lines, 8 T2EC entries are mapped into 64 bytes and form a *T2EC cache line*.

We define the T2EC region in DRAM to be cacheable in the LLC. This has two significant advantages over writing T2EC information directly out to DRAM. First, it reduces the DRAM bandwidth required to support Memory Mapped ECC by exploiting locality in T2EC addresses and accessing T2EC data in the cache when possible. We map LLC lines from consecutive sets to be adjacent in T2EC DRAM storage to increase the likelihood that accesses to arrays in the program will have their T2EC data placed in the same T2EC line. A T2EC cache line is fetched to the LLC, modified in the LLC, then evicted to DRAM like a normal cache line. In other words, the LLC is dynamically partitioned to data and T2EC cache lines and T2EC storage overheads are eventually amortized to DRAM.

The second advantage of caching T2EC data is in matching the T2EC access granularity to DRAM burst size. A typical T2EC will require a small number of bytes for every LLC line and writing in such fine granularity to modern DRAM systems is inefficient and wastes limited DRAM bandwidth. By caching T2EC data, the LLC acts as a write-combining buffer for T2EC bits and all writes to DRAM are performed in the granularity of one or more DRAM bursts.

3.2 Cache Operation

Data Access. We assume the cache level preceding the LLC (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 read from the LLC (fill into preceding level or write-back into DRAM), the T1EC is used to detect and potentially correct errors (Figure 2(a)). If the T1EC has error correction capability and the detected errors are correctable, then the errors are corrected immediately. Otherwise, the correction is performed by the T2EC as depicted in Figure 2(b). Clean cache lines do not require any T2EC access and are re-fetched from DRAM or are corrected by the T1EC if possible. Dirty cache lines, however, may require accessing T2EC data and decoding it. If the T2EC information is not already in the cache it is fetched from DRAM.

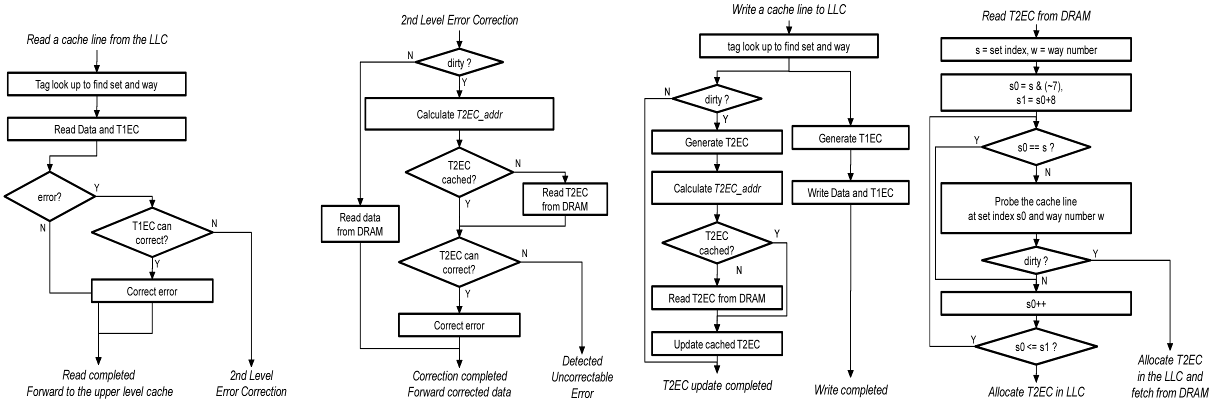
The procedure for writing a line into the LLC is summarized in Figure 2(c). In all write cases, the T1EC is computed and stored in the T1EC portion of the cache line. A T2EC, however, is only computed for dirty lines that are written back into the LLC from a previous-level cache (e.g., L1). This newly generated T2EC is mapped to a cacheable DRAM address, and if this T2EC address is already in the cache it is updated with the newly computed T2EC. If the T2EC address is a cache miss, a line is allocated for it in the LLC and populated with the relevant T2EC information. Note that a T2EC is generated only for a dirty cache line and T2EC encoding / caching is concurrent with the LLC data write.

T2EC Access. Our architecture provides an optimization for the allocation and population of T2EC lines in the LLC. Because a T2EC is just an ECC for a dirty cache line, we can often simply allocate a T2EC line in the LLC without first fetching its corresponding data from DRAM. If all the cache lines that are mapped to a single T2EC cache line are clean, the corresponding T2EC information in DRAM is stale and unnecessary and it is wasteful to read it. The optimized procedure for allocating and populating a T2EC line is illustrated in Figure 2(d). T2EC cache line fills occur only when there is at least one dirty cache line among the cache lines mapped to the T2EC cache line.

Filling a T2EC line from DRAM is similar to a regular data line fill and the read request is first placed in an MSHR. The MSHR is responsible for merging the newly computed T2EC with the T2EC data corresponding to other cache lines that is being fetched from DRAM. This merging capability is similar to that provided by the MSHRs between two cache levels where the granularity of writes is smaller than the cache line size (e.g., when the preceding level is a write-through cache). To accomplish the T2EC merging, each MSHR entry is extended with a bit-vector that indicates what portion of the T2EC line has just been calculated and what portion is being fetched from DRAM (Figure 3). This bit vector enables additional pending T2EC writes to the same T2EC line to proceed while the line is being fetched.

3.3 Impact on Area, Energy, and Performance

ECC Storage. Our two-tiered ECC scheme requires that all LLC lines be uniformly protected by the first-tier T1EC. Thus, the on-chip area of Memory Mapped ECC is equal to the total T1EC size, which is $\frac{L\times C}{B}$ bytes (Table 3). This area overhead, however, is lower than that of the conventional ECC since the T1EC is a low-cost code. For instance, using a simple parity-based error detecting code as the T1EC reduces on-chip overheads significantly compared to the conventional SEC-DED Hamming ECCs of today's



(a) Cache Read from LLC (b) 2nd Level Error Correction (c) Cache Write to LLC (d) Allocate a T2EC line

Figure 2: Operations in LLC with Memory Mapped ECC.

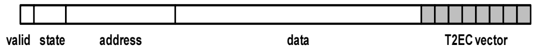


Figure 3: Extended MSHR entry. The bit vector length is equal to the number of T2EC entries mapped to a single LLC line.

processors. We also require a small amount of DRAM to be dedicated to T2EC storage, but as explained above, this area is much smaller than the LLC capacity and is insignificant when compared to DRAM capacity. Because the LLC is shared for both data and T2EC information, it is possible that application performance suffers when compared with the baseline organization. We evaluate this in detail in Section 4.

Error Detection. When properly designed, the error detection using a T1EC, which is the most common operation in error protection, is less complex than that of the conventional ECC. For instance, detecting a single bit error is much simpler using a parity based T1EC than with a SEC-DED Hamming code. We do not evaluate this advantage of our architecture in this paper and refer the reader to the PERC architecture [27], which uses decoupled error detection and error correction to improve the performance and energy of a first-level cache.

ECC Generation and Encoding. A T1EC will be computed and encoded for every LLC access, including both clean and dirty cache lines. Thus, we choose a T1EC with low encoding complexity. The strong protection is provided by the T2EC, and the complexity of computing and encoding the T2EC can be non trivial. T2EC data is rarely read, however, because errors that cannot be handled at the T1EC tier are rare. Therefore, the T2EC encoding circuits need only keep up with the throughput of LLC write operations and latency can be ignored. ECC generation does not impact performance and the required energy is no larger than in the baseline or related techniques. In fact, as in prior work, energy requirements are lower because the complex T2EC is only computed for dirty lines as explained in [18, 27].

Error Correction. When the T1EC detects an error, the correction mechanism will attempt to correct the error using the T1EC if possible, which will be done with low energy and latency. If the chosen T1EC is not strong enough for correction, and the cache line is clean, error correcting simply involves re-fetching the line from DRAM with the corresponding energy and latency penalty. For dirty lines, the correction mechanism accesses the T2EC data and decodes the redundant information. Thus, the maximum penalty for correction is a single LLC line fill from DRAM and invoking T2EC decoding. Given the rare occurrence of an error on a dirty line that cannot be corrected by the T1EC, this overhead is entirely negligible.

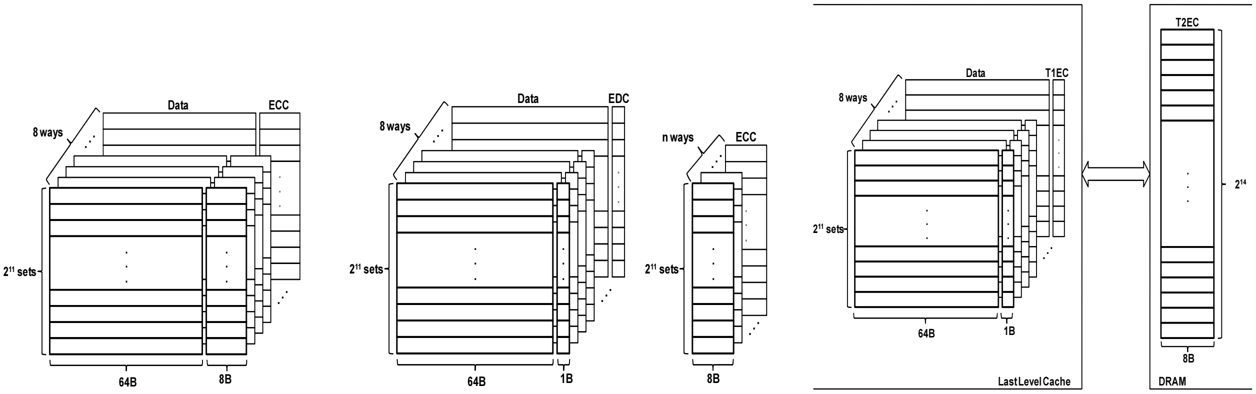
3.4 Comparison with Baseline and MAXn

Finally, we compare the organizations of the baseline, MAXn, and Memory Mapped ECC architectures (Figure 4). Note that all three configurations have roughly the same error protection capability.[1] Memory Mapped ECC only dedicates on-chip storage to the T1EC and significantly reduces the area and leakage energy overheads of strong ECC. We use CACTI [34] to measure the LLC and related ECC area. Table 4 compares the area overheads, leakage power, and energy per access in a 45nm process. Memory Mapped ECC's on-chip ECC overhead is only 2.4% of the LLC area, which is much smaller than the overhead of the conventional ECC (20.7%) and MAXn ($3.4\% - 6.5\%$ for $n = 1 - 4$). Note that Memory Mapped ECC reduces not only ECC area overheads but also leakage power and energy per access. These savings are achieved in a different way than in the energy-efficient scheme of [18], which power-gates the ECC portion of clean lines to curb leakage. In Memory Mapped ECC, no leakage is associated with T2EC because it is stored within the data array, whereas in the energy-efficient scheme, the amount of leakage is proportional to the total number of dirty lines in the LLC. The tradeoff we make is increasing dynamic energy because of a larger number of access to the LLC to manage T2EC storage. As we discuss in Section 4.3.2, however, static leakage power dominates dynamic power and validates our tradeoff decision.

4 Evaluation

In this section, we evaluate Memory Mapped ECC using the Simics full system simulator [21] with the GEMS [23] toolset. We use the OPAL out-of-order processor model of

[1] As explained in Section 3, the interleaved parity codes used by MAXn and the Memory Mapped ECC can detect burst errors of up to 15 bits as opposed to the 16 bit bursts detectable by the baseline (16bit burst error).



(a) Conventional ECC (b) MAXn (n<8) (c) Memory Mapped ECC

Figure 4: Implementations of baseline, MAXn, and Memory Mapped ECC.

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

	Baseline /wo ECC	**Baseline /w ECC**	**MAX1**	**MAX2**	**MAX4**	**Memory Mapped ECC**
Leakage Power (W)	1.4	1.6	1.5	1.5	1.6	1.4
Energy per Read (nJ)	2.0	2.4	2.1	2.1	2.1	2.1
Area (mm^2)	10.0	12.0	10.3	10.4	10.6	10.2
ECC area overhead (%)	-	20.7	3.4	4.2	6.5	2.4

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 [35] into GEMS. We use challenging applications from the SPEC CPU 2006, PARSEC, and SPLASH2 benchmark suites that stress the memory subsystem. Because accurate full-system simulation is very slow we augment the simulation results using PIN emulation [20] to study applications behavior with large datasets. We believe that simulating a single core with the given parameters proves the utility of Memory Mapped ECC and conclusions can be drawn on the performance and overheads if implemented within a multi-core processor, and we discuss Memory Mapped ECC in the context of CMPs in Section 4.5.

4.1 Baseline System Configuration

Table 5 describes the baseline system configuration. Because eager write-back [17] improves DRAM channel utilization (up to 26% performance improvement in our experiments) and reduces the number of dirty cache lines in the LLC, we included eager write-back in the baseline system. We set the eager write-back threshold to 10^6 cycles.[2] The baseline system uses uniform ECC with an 8-bit SEC-DED code associated with every 64-bit data word or, 8 bytes of 8-way interleaved SEC-DED for every 64-byte LLC line.

4.2 Workloads

We use a mix of the SPLASH2 [37], PARSEC [2], and SPEC CPU 2006 [30] 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 [8] to extract five representative regions of 200M instructions and report their weighted sums. We use much larger datasets and execute all applications to completion for our PIN-based evaluation (Section 4.4).

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 (10^6 cycle threshold) 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

4.3 Results and Analysis

We first show the overall performance results and compare the impact of Memory Mapped ECC to that of the MAXn schemes [14] (Section 2.3). We then analyze the results in depth and attribute performance degradation to one of two main sources: increased LLC miss rate and increased memory traffic. LLC miss rate can go up with Memory Mapped ECC because the LLC is shared between T2EC and regular data (Section 4.3.2). Memory traffic might be higher, because of the fetching and writing back of T2EC information. Note that the additional memory traffic only degrades per-

[2] Because eager write-back degrades performance in bzip2 and lbm, it is disabled in those applications. Also, different eager write-back thresholds are applied to fluidanimate and libquantum; 0.25M and 1.2M cycles, respectively, to optimize baseline performance.

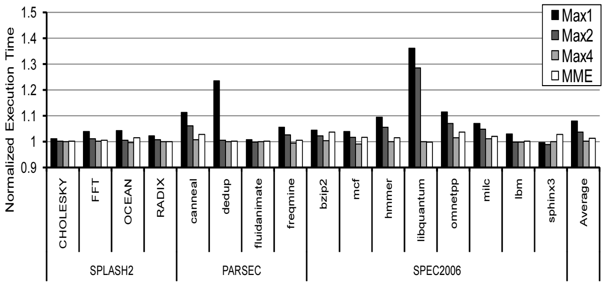


Figure 5: Normalized execution time.

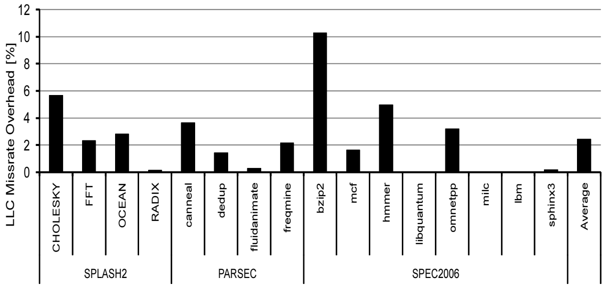


Figure 6: LLC miss rate overhead of Memory Mapped ECC normalized to the miss rate of the baseline.

formance if it competes with data traffic for DRAM bandwidth. It is therefore important to analyze both the overall traffic demands as well as the timing of the added DRAM reads and writes (Section 4.3.3). We also evaluate the impact on power dissipation and the partitioning of the LLC between data and T2EC.

4.3.1 Impact on Performance.

Figure 5 compares the execution times of MAXn and Memory Mapped ECC normalized to the execution of the baseline configuration, which dedicates ECC storage uniformly to all cache lines. The impact on application performance of Memory Mapped ECC is minimal, with an average performance penalty smaller than 1.3% and a maximum degradation of under 4%. These results are encouraging because of the benefits our scheme provides 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, which uses $0.1mm^2$ more area than Memory Mapped ECC, averages over 8% performance loss with several applications suffering $10 - 36\%$ degradation. Even with another 2% increase in area required by MAX2, Memory Mapped ECC still performs better. Only MAX4, which requires 4% more area than Memory Mapped ECC, slightly improves performance over our architecture.

The anomalous behavior of OCEAN, freqmine, mcf, and sphinx3, where the performance of MAX2 and MAX4 is slightly higher than baseline (less than 1%), is a result of increased early write-backs for lines that are being forced clean 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 Memory Mapped ECC, as well as with MAXn, and we do not report their results.

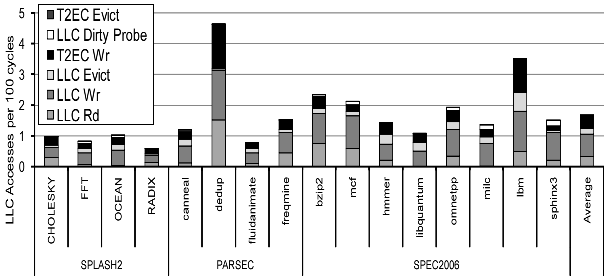


Figure 7: Breakdown of LLC accesses. *LLC Rd* and *LLC Wr* are read and write requests for data, and *LLC Evict* are write-backs of LLC lines to DRAM. *T2EC Wr*, *LLC Dirty Probe*, and *T2EC Evict* are the additional requests required by Memory Mapped ECC: *T2EC Wr* writes T2EC bits to a T2EC line in the LLC; *LLC Dirty Probe* checks whether a cache line is dirty; and *T2EC Evict* moves a T2EC line to DRAM.

4.3.2 Impact on LLC.

In this subsection we analyze the impact of Memory Mapped ECC on the behavior of the LLC, including data miss rate, LLC accesses, and power consumption.

Figure 6 shows that the sharing of LLC resources between T2EC information and data does not significantly impact the LLC data miss rate, which on average increases by only 2% and no more than 11% (bzip2). Storing T2EC information within the memory hierarchy, however, does require a significantly larger number of accesses to the LLC. As shown in Figure 7, Memory Mapped ECC increases the total number of LLC accesses by around 36% on average, of which 85% are writes of T2EC bits (*T2EC Wr*) and 14% are for testing whether a LLC line is dirty or clean as described in Figure 2(d) (*LLC Dirty Probe*). We accurately model contention on the LLC port in our simulations and demonstrate that this increased pressure does not significantly impact performance. Additionally, as we discuss below, the overall power dissipation of the LLC is significantly improved by Memory Mapped ECC even when accounting for the additional accesses.

Figure 8 compares LLC power consumption of the baseline, decoupled EDC/ECC [27], and Memory Mapped ECC estimated using CACTI 5.1 [34] for the cache parameters in Table 5 in 45nm technology. Decoupled EDC/ECC saves energy by separating EDC and ECC so that a read accesses only data and EDC bits, while a write also accesses ECC information. Decoupled EDC/ECC reduces dynamic power while performance and traffic are identical to the baseline. Memory Mapped ECC consumes less dynamic power than the baseline and slightly more dynamic power than decoupled EDC/ECC. The overall power, however, is significantly reduced by Memory Mapped ECC because of the reduction in the dominant leakage power. While we do not show the results in this paper, Memory Mapped ECC has an advantage over MAXn and the energy-efficient scheme because leakage dominates LLC power dissipation, as discussed in Section 3.4. The reasons why the impact on dynamic power is small even though the increase in number of accesses is large are twofold. First, the energy per LLC access of the Memory Mapped ECC architecture is over 14% lower because of the reduction in area (Table 4). Second, most of the additional accesses are T2EC writes and LLC dirty probes, which require only 8 bytes and 1 bit of data respectively.

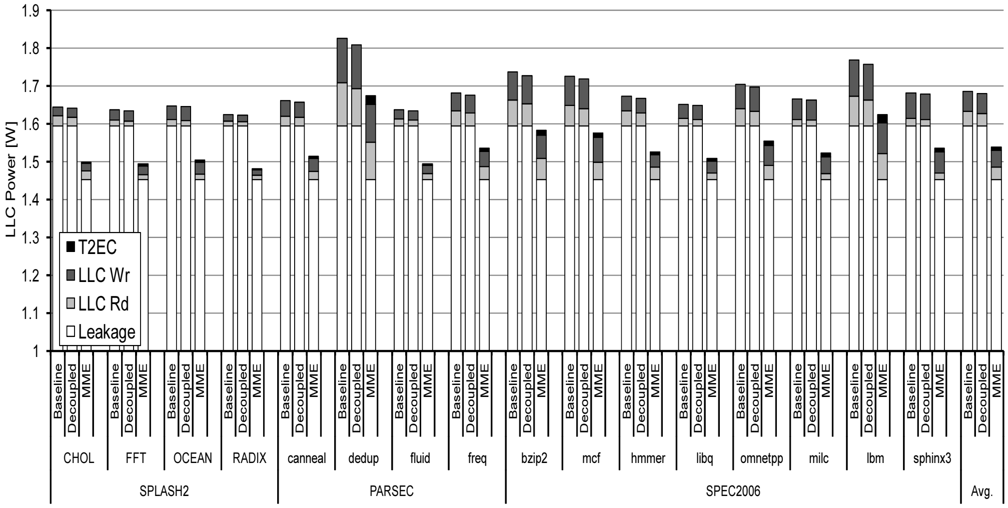


Figure 8: LLC power consumption estimated with CACTI for a 1MB, 8-way, 64B cache line LLC in 45nm technology.

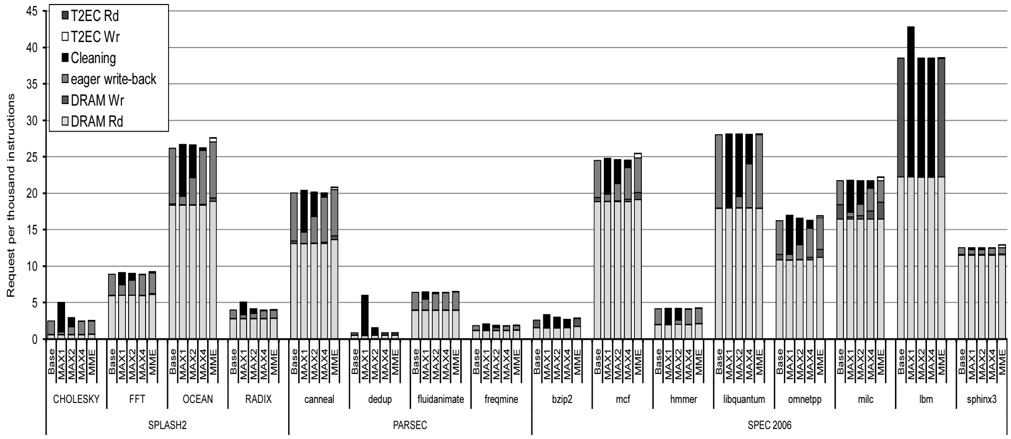


Figure 9: DRAM traffic comparison.

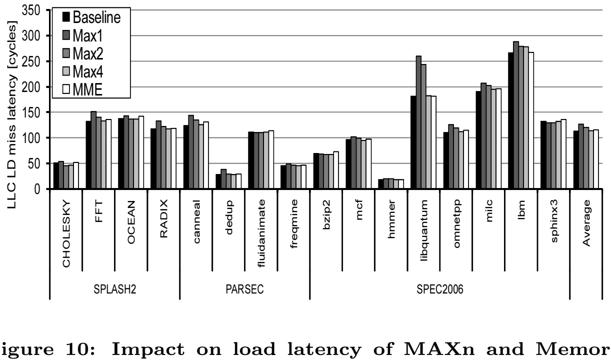
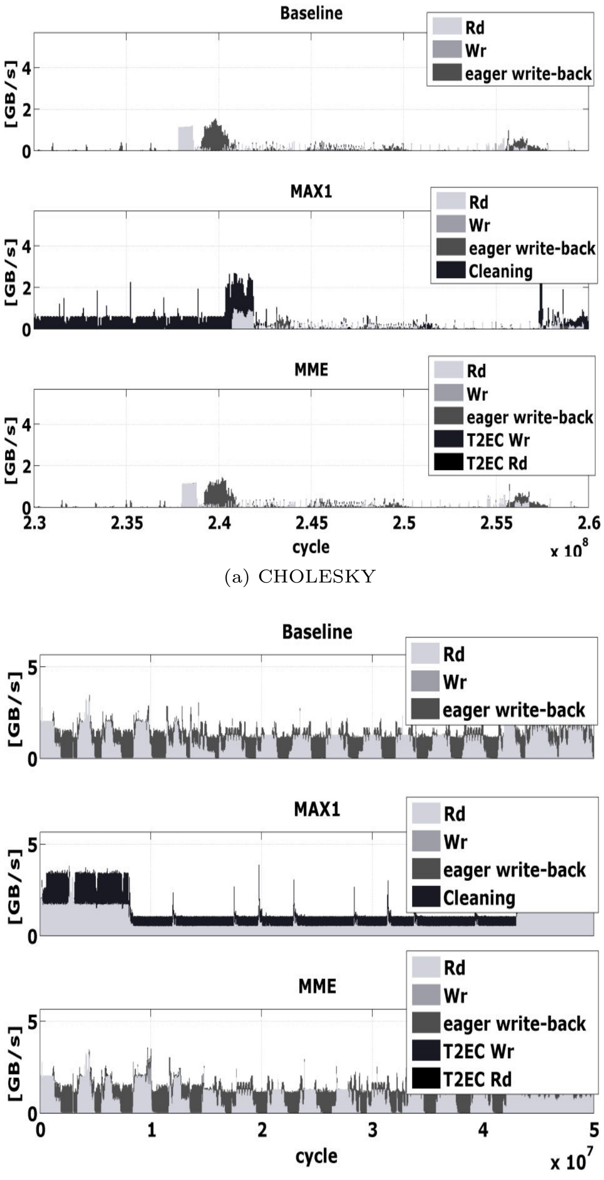


Figure 10: Impact on load latency of MAXn and Memory Mapped ECC.

4.3.3 Impact on DRAM Traffic.

Figure 9 shows the total DRAM traffic broken down into components of data reads and writes (of LLC lines), eager write-backs of cache lines, MAXn writes for cleaning lines, and Memory Mapped ECC T2EC reads and writes. Memory Mapped ECC requires additional memory operations, with T2EC reads and writes contributing an additional 0.1% and 1.4% respectively to total DRAM traffic. These extra DRAM requests, however, are well behaved and as shown earlier do not have a large impact on application performance. MAXn, on the other hand, does not increase overall memory traffic much in most cases, but the type of traffic and its timing differ significantly from both the baseline and Memory Mapped ECC. As shown in Figure 9, MAX1 and MAX2 essentially replace eager write-backs with DRAM writes for cleaning cache lines to avoid more dirty lines in a set than supported. Unlike eager write-backs, which are scheduled during idle DRAM periods [17], cleaning writes compete with demand fetches and degrade performance.

We demonstrate this detrimental effect of replacing controlled eager write-backs with cleaning writes in Figure 10 and Figure 11, which show the average load latency across all applications and snapshots of traffic traces from CHOLESKY



(a) CHOLESKY

(b) OCEAN

Figure 11: DRAM traffic over time.

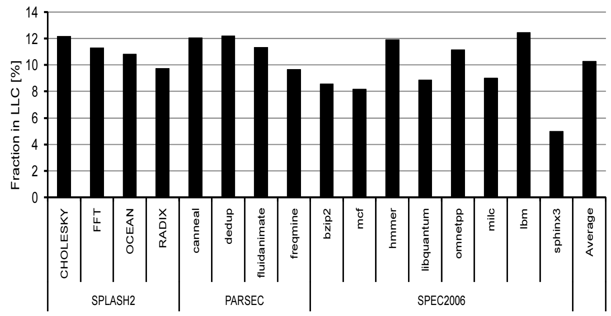
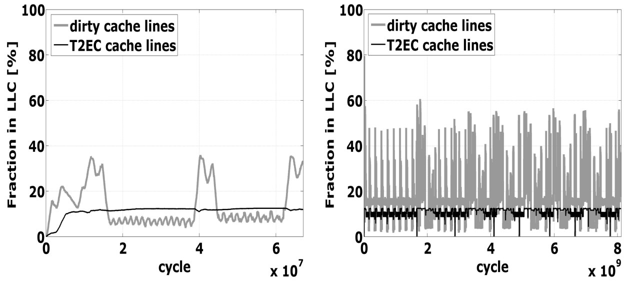


Figure 12: Average fraction of T2EC cache lines in the LLC.



(a) FFT

(b) OCEAN

Figure 13: Fraction of dirty and T2EC cache lines over time.

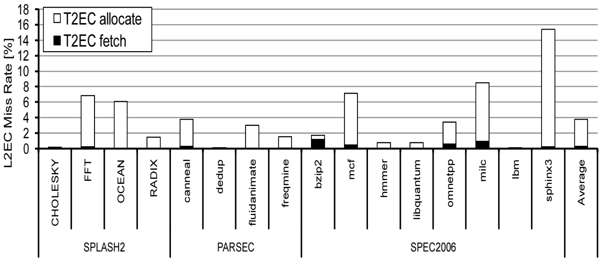


Figure 14: T2EC miss rate.

and OCEAN. Memory Mapped ECC increases latency by under 6% and only 2.5% on average because the T2EC read and write DRAM traffic is evenly distributed and does not interfere with demand fetches. MAX1, on the other hand, shows very bursty cleaning write traffic and degrades load latency by up to 42% and over 11% on average.

4.3.4 LLC T2EC Information.

Figure 12 shows that the average fraction of the LLC that is occupied by T2EC lines is roughly 10%. It is possible that we can reduce the total number of T2EC lines by changing the cache policy for T2EC lines, such as, inserting T2EC lines in the least recently used (LRU) position. We did not pursue such techniques in this paper because the performance overheads of Memory Mapped ECC are already low. Figure 13 illustrates the dynamic adjustment of T2EC lines to the number of LLC dirty lines in the FFT and OCEAN applications. In FFT, we can see the spatial locality of placing multiple T2EC words in a single cache line. Even as the number of dirty lines changes, there is no need to allocate additional cache lines for T2EC information. OCEAN demonstrates how Memory Mapped ECC transparently adjusts to the rapid changes in the number of dirty lines in the LLC.

Figure 14 presents the *T2EC miss rate*: the fraction of accesses to T2EC data not found in the LLC. *T2EC fetch* is the rate of T2EC fetches from DRAM and *T2EC allocate* represents the fraction of fetches avoided when all addresses that map to the T2EC line are clean (see Figure 2(d)). Note that nearly all T2EC fetches from DRAM are avoided on average. The low actual T2EC miss rate (T2EC fetch), less than 0.5% on average, indicates that T2EC caching effectively captures the locality of T2EC accesses.

4.4 PIN-Based Emulation

Full-system cycle-based simulations can provide accurate performance prediction and measurements, but are very slow,

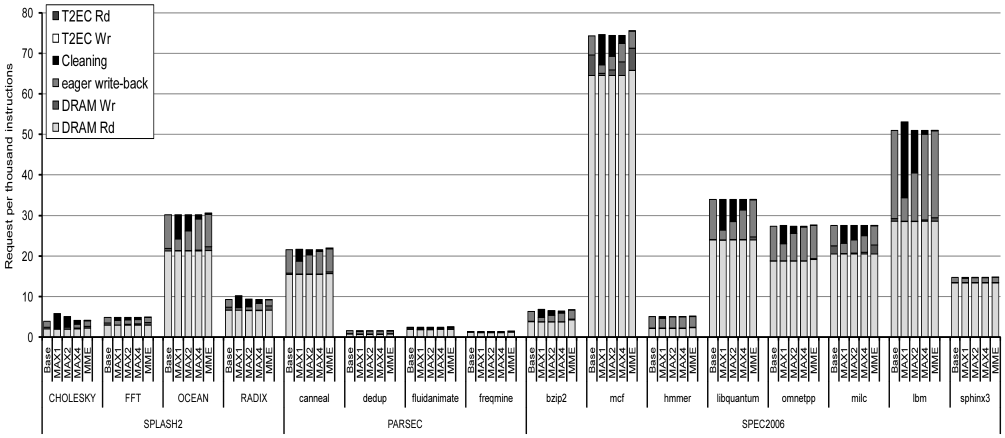


Figure 15: Total traffic with PIN-based emulation.

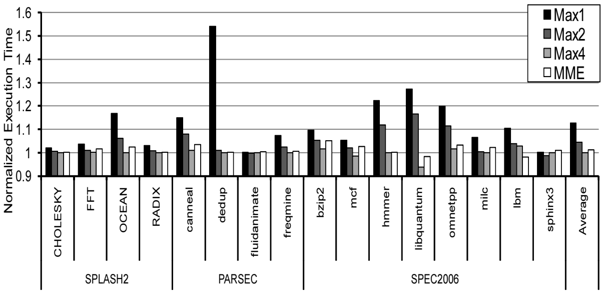


Figure 16: Normalized execution time (2.667 GB/s DRAM bandwidth).

limiting the number and lengths of experiments. Hence, we simulated small to medium problem sets with the SPLASH2 and PARSEC benchmarks, and only representative regions of SPEC applications. To understand how larger datasets affect Memory Mapped ECC, we used PIN [20] to qualitatively evaluate LLC behavior for: tk29.O for CHOLESKY; 4M samples for FFT; a 1026 × 1026 grid for OCEAN; 8M samples for RADIX; the simlarge inputs for all the PARSEC applications; and the complete SPEC CPU 2006 application runs. We implemented eager write-back, MAXn, and Memory Mapped ECC schemes in a two level cache hierarchy PIN tool. Note that our PIN based emulation is only used to collect statistics on LLC behavior and does not utilize a timing model, an instruction cache, stall and re-try due to finite resources such as MSHRs, or a DRAM model. The results are presented in Figure 15 and follow similar trends to those seen with full simulation (Figure 9). The most significant difference is that the traffic overhead of Memory Mapped ECC is smaller when larger datasets are used and we expect to have an even smaller performance degradation than the 1.3% average experienced with small datasets (Section 4.3.1). We repeated the experiments while changing the LLC size between $1 - 16$MB and the results and trends are similar to those reported in the paper.

4.5 Multi-Core Considerations

We have so far discussed and evaluated Memory Mapped ECC in the context of a single core and now briefly discuss multi-core and multi-processor implications. Memory Mapped ECC is easily integrated with any cache coherence mechanism because the T2EC region is fixed and inherently private; the mapping is between physical cache lines and physical memory. A potential problem with Memory Mapped ECC is that increased DRAM traffic for the T2EC will hurt performance given the relatively lower memory bandwidth per core of a multi-core processor. To assess this sensitivity to bandwidth, we evaluated a system with low DRAM bandwidth of only 2.667 GB/s, which is half the bandwidth of the baseline system. As Figure 16 shows, Memory Mapped ECC is not sensitive to DRAM bandwidth and the relative performance is almost unchanged in the limited DRAM bandwidth system. An anomaly is lbm where Memory Mapped ECC slightly improves performance. Most of the memory accesses of lbm are stores, making the application very sensitive to write-back traffic. Our analysis, not shown in this paper, determined that Memory Mapped ECC reduced the effective size of the LLC leading to the minor perturbation in performance.

5 Conclusions and Future Work

This paper presents a novel architecture that provides strong error protection for a last-level cache with minimal hardware and performance overheads. Memory Mapped ECC minimizes the need to dedicate SRAM resources to maintain ECC information by placing it within the memory hierarchy and re-using existing cache storage and control. The savings in area and power dissipation are significant, 15% and 8% respectively, while performance is degraded by only 1.3% on average and no more than 4%. These results are achieved with a single storage mechanism and no additional hardware, and outperform prior techniques on both hardware resource reductions and performance metrics.

The Memory Mapped ECC architecture also enables general tiered error protection techniques and offers great design flexibility by mapping the required redundant information to memory. In future work, we will explore how to exploit this flexibility by supporting multiple T2EC mechanisms that can be tailored to different applications, data types, and access patterns. Such mechanisms have the potential to increase both protection capability and performance, but are difficult and costly to implement when storage resources are dedicated and designed for a specific code. We also believe that our memory-mapped mechanism is general enough to support other meta-data that needs to be associated with physical cache lines.

6 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 *Proceedings of 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 University, January 2008.

[3] B. H. Calhoun and A. P. Chandrakasan. A 256kb Sub-threshold SRAM in 65nm CMOS. In *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, February 2006.

[4] 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 Symposium on VLSI Technology*, June 2005.

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

[6] N. Derhacobian, V. A. Vardanian, and Y. Zorian. Embedded Memory Reliability: The SER Challenge. In *Proceedings of the Records of the 2004 International Workshop on Memory Technology, Design, and Testing*, August 2004.

[7] Digital Equipment Corp. *Alpha 21264 Microprocessor Hardware Reference Manual*, July 1999.

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

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

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

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

[12] 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.

[13] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe. Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding. In *Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2007.

[14] S. Kim. Area-Efficient Error Protection for Caches. In *Proceedings of the Conference on Design Automation and Test in Europe (DATE)*, March 2006.

[15] S. Kim and A. K. Somani. Area Efficient Architectures for Information Integrity in Cache Memories. In *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA)*, May 1999.

[16] J. P. Kulkarni, K. Kim, and K. roy. A 160mV Robust Schmitt Trigger Based Subthreshold SRAM. *IEEE Journal of Solid-State Circuits*, 42(10):2303–2313, October 2007.

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

[18] L. Li, V. S. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Soft Error and Energy Consumption Interactions: A Data Cache Perspective. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, August 2004.

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

[20] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 2005.

[21] 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.

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

[23] 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. *Computer Architecture News (CAN)*, 33:92–99, November 2005.

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

[25] A. M. Patel and M. Y. Hsiao. An Adaptive Error Correction Scheme for Computer Memory System. In *Proceedings of the fall joint computer conference, part I*, December 1972.

[26] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of Society for Industrial and Applied Mathematics*, 8:300–304, June 1960.

[27] N. N. Sadler and D. J. Sorin. Choosing an Error Protection Scheme for a Microprocessor's L1 Data Cache. In *Proceedings of International Conference on Computer Design (ICCD)*, October 2006.

[28] 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 *Proceedings of IEEE Workshop on Silicon Errors in Logic - System Effects (SELSE)*, April 2007.

[29] C. Slayman. Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations. *IEEE Transactions on Device and Materials Reliability*, 5:397– 404, September 2005.

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

[31] 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.

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

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

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

[35] 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.

[36] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu. Trading Off Cache Capacity for Reliability to Enable Low Voltage Operation. In *Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA)*, June 2008.

[37] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA)*, June 1995.

[38] 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 *Proceedings of the International Solid-State Circuits Conference (ISSCC)*, February 2005.

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

[40] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam. ICR: In-Cache Replication for Enhancing Data Cache Reliability. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, June 2003.