Mitigating Multi-bit Soft Errors in L1 Caches Using Last-Store Prediction

Brian T. Gold, Michael Ferdman, Babak Falsafi, and Ken Mai
Computer Architecture Laboratory (CALCM)
Carnegie Mellon University, Pittsburgh, PA 15213
http://www.ece.cmu.edu/~truss

***Abstract*—Recent studies suggest that the rate of spatial multi-bit soft errors will increase with future technology scaling. Unfortunately, multi-bit errors cannot be effectively mitigated with conventional techniques in L1 data caches (e.g., bit interleaving or stronger coding) due to high power and/or latency overheads. We propose the last-store predictor, a lightweight prediction mechanism that accurately determines when a cache block is written for the last time and writes the data back to the L2 cache where increased access latency permits more effective multi-bit error protection. Using a combination of commercial workloads and SPEC CPU2000 benchmarks, we show that, on average, write-back L1 data caches are 42% vulnerable to multi-bit soft errors. Where SECDED ECC fails to mitigate multi-bit errors, our mechanism reduces the multi-bit soft-error vulnerability to 12% on average.**

I. Introduction

Rising soft-error rates are a major concern for modern microprocessor designers. The reduction in charge stored in memory cells, a result of continued technology scaling, leaves on-chip SRAMs (e.g., caches, TLBs, register files) highly susceptible to soft errors. Coding techniques, such as SECDED ECC (single-error correct, double-error detect), are widely utilized for protecting on-chip SRAMs. For L1 data caches, however, where low access latencies are critical, the additional delay to correct ECC errors prohibits inline correction on a read. In the event an error is detected on a read, recent designs such as the AMD Opteron throw a machine check exception asynchronously, potentially halting the machine to prevent silent data corruption [1].

Further compounding problems, recent work suggests that spatial multi-bit errors, where a single cosmic particle strike upsets multiple neighboring memory cells, are increasingly likely at future technology nodes [9,13]. Bit interleaving, also called column multiplexing, is the conventional approach used to protect memory arrays from spatial multi-bit errors. In bit interleaving, bits belonging to multiple ECC check words are physically interleaved so that a spatial multi-bit error does not affect adjacent bits from a single check word. For SRAMs in a high-performance processor, however, our results indicate that interleaving beyond two-way is prohibitively expensive from a power perspective as a result of the additional precharging of bitlines from the interleaved data.

This work was funded in part by NSF awards ACI-0325802 and CCF-0347560, Intel Corp., the Center for Circuit and System Solutions (C2S2), the Carnegie Mellon CyLab, and fellowships from the Department of Defense and the Alfred P. Sloan Foundation.

Previous work [7,10] shows that a large fraction—as much as 80%—of cache frames contain "dead" data, where the last access to a block has occurred but replacement has not yet happened. For a write-back cache, dirty cache blocks that are dead—a last store has occurred for each block—remain vulnerable to multi-bit soft errors. Our results, which corroborate those in Biswas el al. [3], show that roughly 85% of the cache vulnerability is due to dirty blocks that are dead.

Based on these observations, we propose the last-store predictor (LSP), a lightweight prediction mechanism that accurately determines when a cache block is written for the last time and initiates an early write-back of the data to the L2 cache, where increased access latency permits multi-bit protection. The write-back operation writes the value to L2 while retaining the data and permissions in L1 to avoid additional cache misses.

As proposed, the LSP extends the class of program-counter (PC) trace-based predictors studied extensively in the literature [6]. Immediately after observing a trace of PCs that previously led to a last store, the predictor initiates the write-back for the block. Thus, the last-store predictor provides optimal reduction of vulnerability caused by dead time. Using a combination of commercial workloads and SPEC CPU2000 benchmarks, we show that, on average, write-back L1 data caches are 42% vulnerable to multi-bit soft errors. The LSP reduces the multi-bit soft-error vulnerability to 12% on average.

Paper Outline. The remainder of the paper is organized as follows. Section II covers necessary background for our work and our fault model. Section III presents the observations on dead times that enable prediction and early write-back. Section IV provides the details of the last-store predictor, which we evaluate in Section V. We discuss related work in Section VI and conclude in Section VII.

II. Background

A. Fault Model

As technology scales, lower supply voltages and smaller feature sizes continue to decrease the charge stored per SRAM cell [9,13]. The reduction in stored charge results in a lower Qcrit, the critical charge required to flip a bit. Hence, the soft-error rate increases. The two dominant physical sources of soft

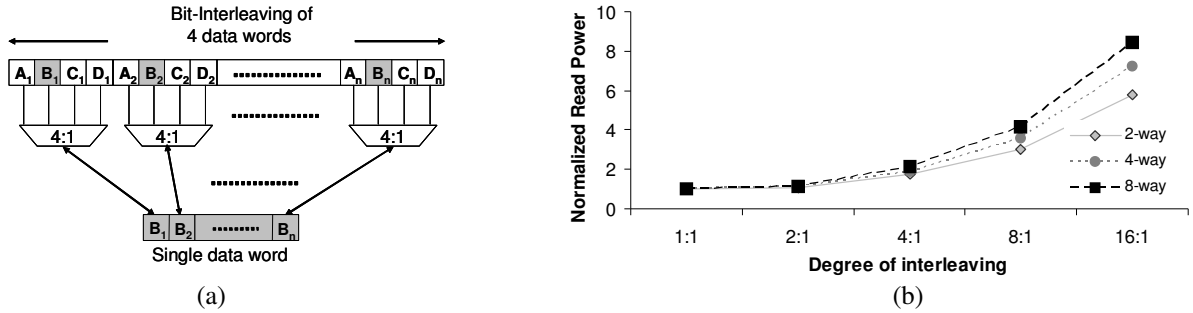

Fig. 1. Bit interleaving to protect against spatial multi-bit errors. (a) illustrates the structure of 4-way physical bit interleaving; (b) shows the dynamic power overheads as a function of the degree of interleaving, for different L1 cache associativities (all are 64kB, two ports, eight banks).

errors studied in the literature are alpha particles from trace radioactivity in packaging materials and heavy neutrons resulting from cosmic radiation [18]. This work considers soft errors from both physical phenomena by investigating the reduction in architectural vulnerability without regard to the underlying fault mechanism.

Multi-bit errors can be classified as either spatial or temporal. Spatial multi-bit errors are the result of a single event upset, while temporal errors result from multiple, independent upsets over time. In this work, we consider only spatial multi-bit errors, as the natural tendency of the cache to 'refresh' itself will mitigate temporal multi-bit errors [11]. Spatial multi-bit errors result when a high-energy cosmic particle strikes a silicon atom, and the resulting nuclear reaction gives off secondary ions capable of upsetting other, nearby memory cells [13].

Recent studies suggest that the spatial multi-bit error rate in SRAMs is increasing exponentially with scaling, leaving vendors without clear solutions to protect on-chip L1 caches. Furthermore, large contiguous bit flips (up to 5) were observed in accelerated beam testing [9], eliminating all practical code-based protection schemes for L1.

B. Cache Model

In this work, we consider protection mechanisms for write-back L1 data caches. Write-back caches are becoming the dominant paradigm for L1 caches in chip multiprocessors, where write-through L1s connected to a shared L2 would be bandwidth prohibitive.[1]

The conventional approach to protecting a write-back L1 is with SECDED ECC (e.g., AMD Opteron [1]). A common design point for ECC uses 8 check bits for every 64 data bits (the nominal L1 cache access size). Normal stores of 64-bit words can compute the ECC check word and store both data and ECC in parallel. To update the ECC bits on a partial write, the cache must issue a read-modify-write (RMW) operation.

For high-performance microprocessors, a major challenge of ECC in an L1 cache is the additional check latency on a read operation. The latency required to compute the ECC syndrome and then correct any errors is often higher than the cache access time itself. For the heavily integrated processor core and data cache, this overhead prohibits inline correction of erroneously-read data. Instead, the approach taken in the AMD Opteron is to assert a machine check exception, which can potentially halt the system [1].

To reduce the likelihood of a soft error resulting in a machine-check exception, the Opteron implements a form of cache *scrubbing*, where a small hardware component traverses the cache one frame at a time, checking for ECC errors and performing inline correction outside the processor core [1]. Essentially, the scrubber component races with normal program operation to reach an error first. If the scrubber wins, a single-bit error will be corrected; however, if the processor reaches the block first, a machine-check exception is thrown. The Opteron allows the user to select a scrubber period, thereby determining the tradeoffs of cache-port bandwidth, power overheads, and soft-error vulnerability protection. Little or no guidance is available to the user on the implications of these tradeoffs.

C. Interleaved ECC

The challenges presented by SECDED ECC also prevent stronger codes—DECTED (double-correct, triple-detect), TECQED (triple-correct, quad-detect), and the like—from solving the problem of multi-bit errors. First, many of these codes do not scale to the large number of contiguous errors observed in beam testing [9]. Second, power and area required for stronger codes prohibits their use even in small L1 caches.

The solution used in DRAMs and to some extent in SRAM arrays, is to physically interleave the data bits belonging to different ECC check words (illustrated in Figure 1(a)). With an N-way degree of interleaving, bits from N different check words are interleaved, and multi-bit errors up to N contiguous bits in size can be tolerated.

The major problem with bit interleaving is the additional power overhead resulting from precharging N times as many bitlines. Because bitlines must be precharged ahead of address decode, it is not possible for a latency-sensitive L1 cache to do selective precharging.

To quantify the power overhead from various degrees of bit interleaving, we modified CACTI 4.2 [15] to model bit-inter-

1. The notable exception at this time is the Sun UltraSPARC T1 (Niagara) [5], which can tolerate the resulting longer access time to L2 with its heavily multi-threaded microarchitecture.

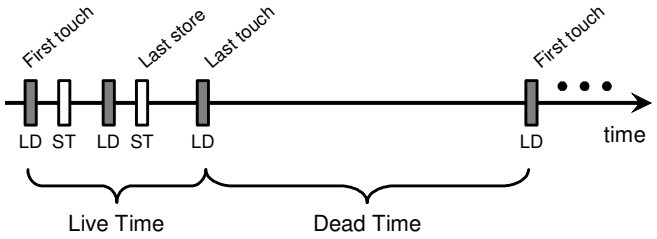

Fig. 2. Accesses to a cache block over time, illustrating the transition from live time to dead time.

leaved caches in which only sub-arrays containing the selected word are activated. Given the target degree of interleaving, CACTI explores the design space using wordline and bitline segmenting. Figure 1(b) shows the normalized dynamic power on a read operation as a function of the degree of interleaving. We plot varying associativities, with all results obtained for a 64kB cache with two ports and eight banks (one port per bank). We directed CACTI to optimize the cache design for delay, as typical of L1 caches.

We observe that dynamic power consumption increases roughly linearly with increasing degrees of bit interleaving. Beyond two- or four-way interleaving, the overhead exceeds twice the dynamic power of the baseline non-interleaved cache. We conclude that interleaving beyond these points is not practical due to power consumption, therefore requiring alternative solutions such as the work proposed in this paper.

III. Observations

A. Avoiding dead-time vulnerability

A cache block is resident within a cache frame between the time when a fill operation brings the block's contents into the cache and the time when that block is evicted through the cache's replacement policy. This timeframe can be broken down into two components, the block's live time and dead time. A block's live time begins when a cache fill places it into its corresponding cache frame, and the block's dead time begins at the last cache hit before the block's eventual eviction (see Figure 2). Due to temporal and spatial locality of data accesses, the contents of a cache block are likely to be accessed many times within the block's live time, whereas by definition, the block contents remain un-accessed during the dead time prior to the block's eviction.

The temporal characteristics of cache blocks in the L1 have been extensively studied in the literature [7,10]. Over a wide range of workloads and cache organizations, the average cache dead time is substantially longer than cache live time; consequently, as much as 80% of cache frames contain blocks in a "dead" state at any time [7,10]. We corroborate and extend these findings, showing that, on average, the dead time to live-time ratio is equally high for dirty cache blocks in a write-back cache.

The long duration of dirty-block dead times has significant implications for the architectural vulnerability of write-back caches in the presence of soft errors. A dirty cache block contains the only valid copy of data between the time a store

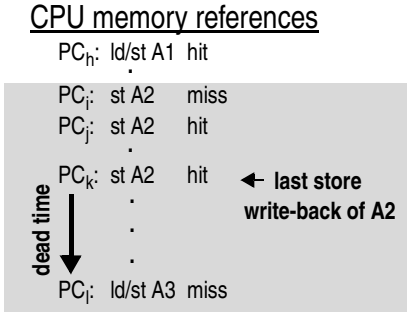

Fig. 3. Example of last-store prediction. Last store to A2 triggers a write-back of the block prior to the miss to A3.

operation is performed and the time the block's contents are written back to a lower level of the memory hierarchy. Because no stores are performed to the block for the duration of its dead time, there is no benefit to delaying write-back until a block's eviction from the cache. Conversely, retaining data in a dirty state during this time unnecessarily exposes the block to soft errors for the long duration of the block's dead time.

Dirty-block dead time constitutes the majority of time when cache contents are vulnerable to soft error. We therefore observe that performing write-back operations at the time of the last store can substantially reduce the system's architectural vulnerability. Furthermore, even if a cache is protected by SECDED ECC, performing write-back operations at the end of the block's live time reduces the probability of unrecoverable errors and silent data corruption due to spatial and temporal multi-bit errors during the block's dead time. By performing the write-back at the end of a block's live time, the architectural vulnerability approaches that of a write-through cache, without incurring additional write-back bandwidth.

B. Identifying block live time

This paper proposes the last-store predictor (LSP), a light-weight mechanism for detecting when a block can be written back to a lower level cache without increasing cache miss rates or incurring substantial bandwidth overheads. Repetitive program behavior enables the LSP to learn traces of store instructions leading up to cache block evictions and to later predict the last-store instruction upon detecting a store sequence that previously preceded an eviction. By correctly identifying last-store instructions, LSP can trigger a write-back precisely at the last-store, thereby minimizing the time when a cache block is vulnerable.

Figure 3 depicts an example of last-store prediction. For simplicity, we assume a direct-mapped L1 for this example. Cache blocks A1, A2, and A3 all map to the same cache frame. The predictor tracks all stores {PCi, PCj, PCk} to block A2 until the block is evicted. Upon a miss to block A3, block A2 is evicted, and the predictor records the instruction trace that led to the last store as a fixed-size last-store *signature*. The trace is represented by a hash of the program counter values that performed a store to A2 (e.g., {PCi, PCj, PCk}). On subsequent encounters of the same sequence of store instructions {PCi, PCj, PCk}, the predictor identifies PCk as the last store

prior to eviction and triggers a write-back of the corresponding cache block.

The LSP mechanism is decoupled from data addresses; it is not sensitive to data structure changes and exhibits short training time. A trace of store instructions leading to an eviction must be observed only once to enable the predictor to identify all last-store operations performed by the same sequence of instructions, regardless of the store addresses. For example, stores at multiple array indices are commonly implemented in loops. This data access pattern results in multiple cache frames containing dirty cache blocks, all written by identical sequences of store instructions. On the first iteration of a loop performing the store operation, LSP learns the sequence of store instructions that lead up to the eviction of modified blocks. LSP records only one fixed-size last-store signature for this sequence of stores. For all subsequent iterations of the loop, the same last-store signature is used to make a prediction of the last-store instruction and enable LSP to initiate write-back of the data at the time of the last store. This operation makes stored data invulnerable to soft error for the remaining time the block resides in the cache.

IV. Design

We propose a cache protection scheme that provides detection of large-scale multi-bit errors and significantly reduces the rate of uncorrectable errors. To detect multi-bit errors, we use word-interleaved parity. While conventional parity stores one parity bit for every 64 bits of data, interleaved parity keeps N check bits (e.g., N=8) for every 64 bits and physically interleaves the data so that no N contiguous bits are part of the same parity group. Interleaving parity *within* a 64-bit data word avoids the additional power overheads associated with interleaved ECC because no additional bitlines must be precharged (compared with non-interleaved ECC).

Our design treats errors detected in a clean cache block as L1 misses, which results in the correct data being retrieved from the memory system. When combined with the LSP, this design removes the vulnerability for dirty blocks from the last store until the block's eviction. Following a write-back, subsequent read accesses do not incur performance overhead while subsequent writes (mispredictions) either mark the block as dirty (if the cache supports clean-exclusive state) or re-acquire write permission from L2.

A. LSP Structure

Figure 4 depicts the anatomy of a LSP implementation. LSP comprises two low-associativity SRAM structures. The history table, organized like the L1 tag array, maintains partial trace encodings of store instructions for each block in the L1 data cache. A signature table contains completed last-store signatures known to correspond to last stores. Both structures are latency insensitive because update and lookup operations in these structures are not on the critical path of the processor.

B. LSP training

Last-store signatures are constructed within the history table. The history table parallels the L1 tag array. For each

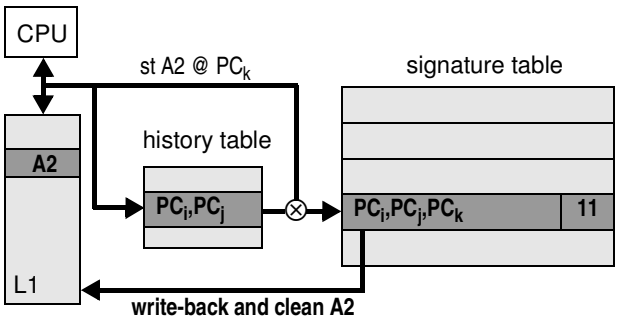

Fig. 4. Anatomy of a LSP implementation.

frame in the L1, a corresponding history table entry maintains an encoded trace of store instructions performed on that cache block since the preceding fill operation. The entry also contains a single *predicted* bit, to indicate that a last-store prediction was previously made on the corresponding cache block. The trace is incrementally constructed from program counter values by applying truncated addition of each new PC value to the previous trace encoding. The history entry is cleared whenever the corresponding L1 frame undergoes block eviction. Upon an eviction from L1, LSP extracts the completed last-store signature from the history table and updates the signature table. Each signature table entry consists of a two-bit saturating counter. The signature table update either replaces an old entry with the new signature and confidence value "2", or increments the confidence count if the new signature is already present in the table.

C. LSP prediction

After each update, the new trace encoding from the history table is used to perform a lookup in the signature table. Presence in the signature table with a high confidence counter value (greater than or equal to 2) indicates that after a previous encounter of the same store-instruction sequence the cache block was evicted. The latest instruction encoded in the trace is therefore predicted as a last store, implying that the corresponding block will not be written to again before it is evicted from the cache. Upon detecting that a last store was performed, LSP triggers a write-back on the corresponding cache block to prevent the dirty data from remaining vulnerable in the cache.

On every L1 write, the trace encoding stored in the history table is updated. Prior to update, the *predicted* bit is checked. If this bit indicates that an incorrect prediction was made (prediction was made but trace encoding was not cleared by a subsequent eviction), the signature table confidence value associated with the old trace encoding is decremented.

V. Evaluation

A. Methodology

We use architectural vulnerability factor (AVF) [12] as a measure of the reduction in soft-error rate achieved by various protection mechanisms. AVF analysis determines the fraction of time that, if an error had occurred, the error might have affected execution outcome. Thus, our results do not indicate failure rates, but rather show the average fraction of time

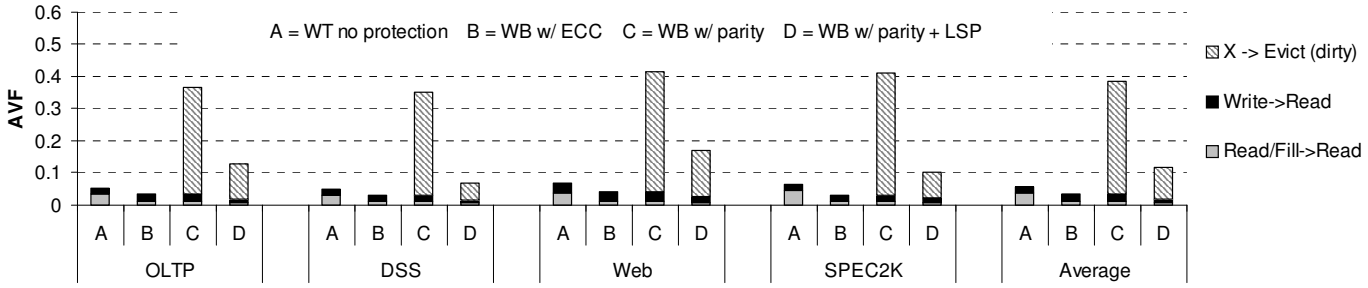

Fig. 5. AVF for a 64kB L1 data cache under a **single-bit** fault model, with various protection schemes. WT is a write-through cache and WB is a write-back cache.

where a structure (e.g., L1 cache) is architecturally vulnerable to soft error.

We adopt the methodology of Biswas et al. [3] to measure AVF for the L1 cache. This technique marks events in the life of every *byte* in each cache block (e.g., fill, read, write, evict). Between any two events, we annotate the time as either vulnerable—required for architecturally correct execution (ACE)—or not vulnerable (unACE). The AVF for the data cache is then the average fraction of the cache in the ACE state over the entire execution.

Our experiments use traces of memory references from a variety of commercial workloads and the SPEC CPU2000 benchmark suite, as shown in Table 1. All traces are run for 1B instructions of warmup, and the AVF is measured for 2.5B instructions thereafter. We include a 500M instruction cooldown to resolve any remaining cache block lifetimes [3].

All traces are collected using FLEXUS, a full-system simulator that extends Virtutech Simics. Our simulations run Solaris 8 on the SPARC ISA. We model a 64kB L1 data cache with two read/write ports implemented with eight single-ported banks (as in AMD Opteron [1]). Other details of the simulated machine do not affect the AVF analysis.

Our workload traces include TPC-C v3.0 online transaction processing (OLTP) workload on *IBM DB2 v8 ESE*. We select three queries from the TPC-H decision support system (DSS), showing scan-dominated, join-dominated, and mixed behavior. All three DSS queries are run on *IBM DB2* as well. We evaluate web server performance with the SPECweb99 benchmark on *Apache HTTP Server v2.0*. We include 26 of the 28 SPEC CPU2000 benchmarks using the ref0 input sets. To report aggregate results, we average the four workload classes (OLTP, DSS, Web, and SPEC2K) with equal weight.

TABLE 1. Workload configurations.

Online Transaction Processing (TPC-C)	
DB2	100 warehouses (10 GB), 64 clients, 450 MB buffer pool
Web Server	
Apache	16K connections, FastCGI, worker threading model
Zeus	16K connections, FastCGI
Decision Support (TPC-H on DB2)	
Qry 2	Join-dominated, 450 MB buffer pool
Qry 6	Scan-dominated, 450 MB buffer pool
Qry 17	Balanced scan-join, 450 MB buffer pool
SPEC CPU2000	
	26 benchmarks, ref0 input sets

B. AVF with Single-bit Fault Model

We first investigate the AVF of the data cache using four alternative designs: write-through with no protection (design 'A'), write-back with non-interleaved SECDED ECC ('B'), write-back with interleaved parity ('C'), and write-back with interleaved parity and a LSP ('D'). The baseline LSP design uses 16-bit signatures and a direct-mapped, 4K entry signature table. We explore predictor sensitivity to signature storage size later in this section.

Figure 5 shows the resulting AVF under a single-bit fault model. For write-through with no protection, the AVF reported corresponds to silent data corruption (SDC) AVF. For the write-back caches, all AVFs reported are for detectable but uncorrectable errors (DUE). We include the write-through design simply to measure the "live" portion of the cache, rather than advocate this as a practical design point or attempt to compare SDC AVF with DUE AVF.[2]

The different categories (shown in the legend) correspond to the AVF contribution of various events. The area shaded in light gray is due to the time between a cache line fill or read and a subsequent read with no intervening events. The category 'X -> Evict (dirty)' corresponds to any event ('X') that precedes an eviction of a dirty block (e.g., write-back to L2). Note that design A, write-through with no protection, has no 'X -> Evict (dirty)' time.

From Figure 5, we first observe that write-back with ECC (design 'B') outperforms the alternatives in terms of single-bit AVF. What little vulnerability remains is due to the live time of a dirty cache block, where this model of SECDED ECC cannot perform inline correction. For clean blocks where an error is detected, we assume the cache can obtain the correct value from L2 or the rest of the memory system (e.g., treat the access as an L1 miss). Hence, write-back with ECC reduces the AVF over write-through with no protection by protecting read-only blocks.

Write-back with interleaved parity (but no LSP) has over an order-of-magnitude greater AVF than write-back with ECC

2. We omit write-through with interleaved parity because the AVF is always zero under our fault model.

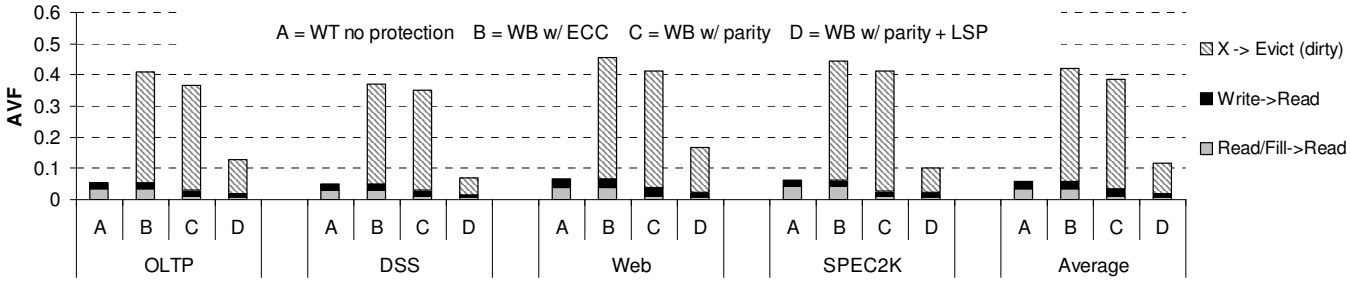

Fig. 6. AVF for a 64kB L1 data cache under a **multi-bit** fault model, with various protection schemes.

entirely due to the contribution of dead-block time. This corroborates the findings of previous studies [7,10], but also shows the dead time of *dirty* blocks greatly exceeds live time.

The addition of a last-store predictor reduces the AVF contributed by dead blocks from 35% down to less than 12% on average. This reduction corresponds to an average of 70% coverage in the predictor, meaning that 70% of all last-stores are correctly predicted.

C. Multi-Bit Fault Model

Figure 6 shows the AVF of the four designs described previously when subjected to a multi-bit fault model. We assume the number of bit flips may be larger than two, and are thus not always detectable by SECDED ECC. We assume that the interleaved parity of designs 'C' and 'D'—write-back with parity and write-back with parity and LSP, respectively—is sufficient to detect any multi-bit error in this fault model.

The AVF for write-through with no protection (design 'A') is unchanged, as the silent-data corruption remains undetected independent of the number of bit flips. Design 'B'—write-back with SECDED ECC, which we assume has no interleaved parity—cannot detect the multi-bit errors and has an SDC AVF exceeding 40%. Note that this result does not account for software-level *derating*, where the flipped data bits may be unused or trigger an error in the OS/application and hence become DUE. The multi-bit AVF for SECDED ECC is not 100% because blocks are sparsely accessed. For read-only blocks, only the bytes read by instructions contribute to AVF.

Nevertheless, it is clear from Figure 6 that for a write-back cache, a technique such as LSP must be used to reduce the vulnerability to multi-bit soft errors. Because the prediction mechanism is independent of the number of bits flipped, the design with LSP (design 'D') maintains the low AVF of the previous section (12% on average).

D. LSP Design Sensitivity

We examine the sensitivity to LSP organization along three dimensions: signature size, signature table size, and associativity. In all cases, we report the coverage and over-predictions of the LSP. Coverage represents the fraction of last stores that are correctly predicted by LSP. Over-predictions represent blocks that are written back but then written again by the program prior to eviction. Over-predictions waste L2 bandwidth but do not otherwise affect execution.

Figure 7 shows the sensitivity to signature size—the number of PC bits used in the signature—where 30 bits is the maximum possible size (uniquely determining the PC in SPARC). These results use an infinite-sized, fully-associative signature table to isolate the aliasing effects due to signature collisions. For all the workloads, the general trend is not only to reduce the coverage as the number of signature bits decreases, but also to increase the over-prediction rate. This trend is due to a *toggling* of signatures where the trace leading to a last store (raising the confidence counter) later results in an over-prediction because of aliasing. From these results, we choose 16-bit signatures as a compromise between coverage and LSP storage size.

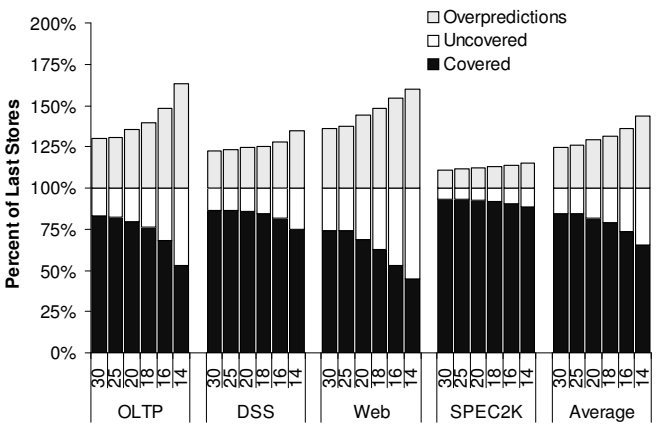

Fig. 7. LSP sensitivity to signature size (# PC bits).

Having fixed the signatures at 16 bits, Figure 8 shows the predictor sensitivity to signature-table size, while maintaining a fully associative structure to isolate the effect of capacity misses in the signature table. The shape observed for OLTP and Web workloads is quite interesting, as over-predictions initially increase as the table size shrinks, but eventually peak around 8K entries and then decline. The signature-size results showed that significant aliasing occurs with 16-bit signatures. Thus, as the table size increases from very small (1K), the number of potential aliases increases. Eventually, the table becomes large enough (e.g., 16K entries) to reduce the aliasing and keep sufficient history of the traces that should not lead to predictions, without affecting coverage.

From Figure 8, we observe that 4K entries is approximately equal in coverage to infinite size and choose this size for the remaining experiment: associativity. Figure 9 shows that with 16-bit signatures and a direct-mapped 4K-entry table, the cov-

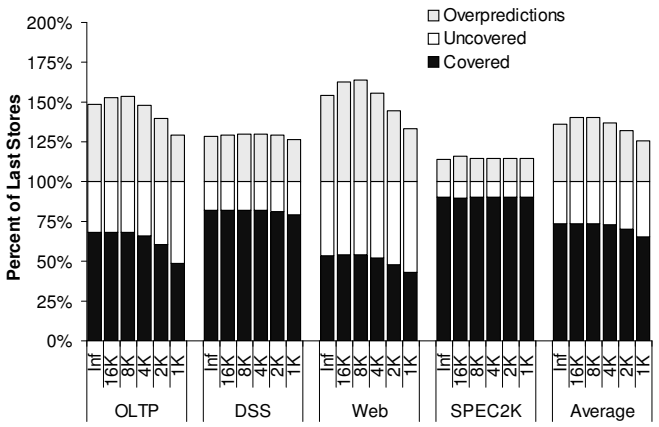

Fig. 8. Sensitivity to the number of entries in the LSP signature table. A fully-associative signature table is used for these experiments to isolate the effect of capacity misses.

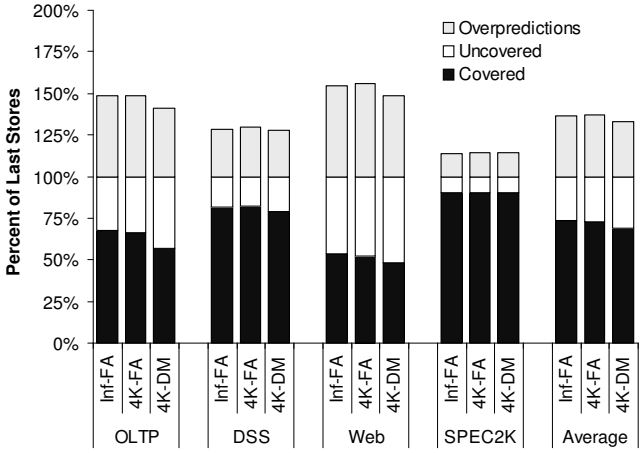

Fig. 9. LSP sensitivity to associativity.

erage and over-predictions are sufficiently close to a fully-associative organization. At this size and structure, the signature table needs four tag bits per entry and thus uses 3kB of total storage (including the 2-bit confidence counters).

E. Area and Power Overheads

The proposed LSP implementation augments the processor microarchitecture with two additional structures. In this section, we quantify the area and power overheads of LSP. We note that both additional structures are substantially simpler than a typical cache. The signature table is a direct-mapped structure that accesses a single two-bit counter value without word-selection logic. As a result, the signature table resembles a cache tag array. Conversely, each history table entry corresponds to a cache frame in the L1 data cache; therefore, this structure only requires storage for its data, without need for a serial or parallel tag lookup.

The LSP history table must store one encoded trace of program counter values per cache block. The area overhead is therefore limited by the number of cache frames in the cache. Furthermore, the area overhead is insignificant when considered in comparison to ECC bits for a cache line. SECDED ECC calls for 12.5% overhead (8 bits of ECC for every 64 bits of data), whereas the fixed-size PC trace encodings and *predicted* bit introduce approximately 3.3% overhead (17 bits per 64 byte block).

To get a sense for the relative contribution of LSP compared to the L1 of our architecture, we used CACTI 4.2 [15] to estimate the energy of the history table and signature table structures and a 64kB L1 cache in a 70nm technology. Despite lookup on every L1 access, the narrow width of the history table and lack of a separate data lookup in the LSP allows for substantially lower read and write energy compared to the L1. CACTI estimates 49pJ dynamic energy for accesses to the two-ported L1 cache. For each L1 access, the last-store predictor requires up to two accesses to the history table and up to two accesses to the signature table. CACTI estimates a total of 3.4pJ dynamic energy for all four accesses to structures sized according to results of the previous section.

CACTI estimates for leakage power of the LSP structures are similarly small compared to the L1 cache. The data cache will leak approximately 250mW, while the LSP structures will leak only about 15mW as a result of significantly smaller size. It should also be noted that lookup in the LSP structures does not require low latency and is not on the critical path, enabling a pipelined design using high-Vt and/or long channel length transistors, which may further reduce leakage compared to the highly latency-sensitive L1 cache.

VI. Related Work

The mechanisms proposed in this paper fall under the broader class of *early-write-back* techniques that reduce vulnerability of L1 caches [2,8,16]. This paper differs from prior work on early-write-back mechanisms by (1) evaluating the AVF reduction for multi-bit errors and (2) studying the use of trace-based prediction (LSP) as the mechanism that triggers early write-back.

Asadi et al. [2] propose a periodic write-back of dirty blocks where a circuit examines cache frames sequentially, writing back the data to L2 if the block is dirty. Much like hardware scrubbing used in the AMD Opteron [1], the periodic write-back hardware consumes an L1 port for every cycle in which it examines the tag array for a dirty block. The LSP technique proposed in this paper does not consume additional L1 cache-port bandwidth.

Li et al. [8] propose early write-back to reduce the energy consumption of single-bit error protection mechanisms. In contrast, we propose a form of early write-back (with LSP) to protect against multi-bit errors. As we showed in Section V, the additional power required for the LSP arrays is significantly smaller than the power used by the L1 cache**.**

Vera et al. [16] use cache decay, originally proposed for leakage energy reduction, which evicts a block if it has been dead for a period of time (called the decay interval). Their evaluation differs from this paper in that [16] models a write-through cache only and does not report multi-bit AVF reductions. Furthermore, the mechanisms in [16] invalidate the

blocks after the decay interval, which increases the miss rate in L1 (on over-predictions) and degrades performance.

Recent work has also examined *in-cache protection* as an alternative to early-write-back techniques. Kim and Somani [4] selectively protect the most-frequently- or most-recently-used cache blocks. These mechanisms target a fault model where blocks are prone to errors during their access as a result of electrical noise or cross-coupling [4] and do not address a particle-radiation fault model.

Sadler and Sorin [14] decouple error detection from correction by integrating fast error-detection codes (EDCs) with the data array in L1 and keep a separate structure for holding error correcting codes (ECCs). The chosen codes could detect and correct multi-bit faults. However, a large area overhead is required for multi-bit correction codes. In contrast, the techniques proposed in this paper do not rely on coding in L1 to correct multi-bit errors and avoid the large area overheads.

Zhang et al. [17] propose the replication cache, a small, fully associative structure that holds duplicate copies of recently-written blocks. When evicting from the replication cache, the data is written back to L2 (as in early-write-back mechanisms). While in the replication cache, blocks enjoy full redundancy and hence large-scale multi-bit fault protection. However, as Sadler and Sorin [14] show, the fixed-size replication cache incurs significant performance overheads in several SPECfp applications because of thrashing in the replication cache and the resulting bandwidth demands on L2.

VII. Conclusions

Increasing soft-error rates, particularly multi-bit error rates, require alternatives to the interleaved ECC used in previous L1 data cache designs. We corroborate and extend prior findings that the majority of time that a dirty cache block is present in the cache is "dead" time; that is, after the last store to the block. We leverage this observation to construct a lightweight predictor for last stores that triggers early write-backs to reduce the AVF of dirty blocks in L1 data caches. We show that, on average, write-back L1 data caches are 42% vulnerable to multi-bit soft errors. Where SECDED ECC fails to mitigate multi-bit errors, our mechanism reduces the multi-bit soft-error vulnerability to 12% on average.

References

[1] AMD. BIOS and kernel developer's guide for AMD Athlon 64 and AMD Opteron processors. Technical Report Pub. 26094, AMD, 2006.

[2] G.-H. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli. Balancing performance and reliability in the memory hierarchy. In *Proc. of 2005 Intl. Symp. on the Perf. Analysis of Syst. and Software (ISPASS 2005)*, March 2005.

[3] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan. Computing the architectural vulnerability factor for address-based structures. In *Proc. of 32nd Intl. Symp. on Comp. Arch. (ISCA-32)*, June 2005.

[4] S. Kim and A. K. Somani. Area efficient architectures for information integrity in cache memories. In *Proc. of 26th Intl. Symp. on Comp. Arch. (ISCA-26)*, pages 246–255, 1999.

[5] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multi-threaded SPARC processor. *IEEE Micro*, 25(2):21–29, Mar-Apr 2005.

[6] A.-C. Lai and B. Falsafi. Selective, accurate, and timely self-invalidation using last-touch prediction. In *Proc. of 27th Intl. Symp. on Comp. Arch. (ISCA-27)*, June 2000.

[7] A.-C. Lai and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Proc. of 28th Intl. Symp. on Comp. Arch. (ISCA-28)*, July 2001.

[8] L. Li, V. Degalahal, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Soft error and energy consumption interactions: a data cache perspective. In *Proc. of 2004 Intl. Symp. on Low Power Electronics and Design (ISLPED)*, 2004.

[9] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong. Characterization of multi-bit soft error events in advanced SRAMs. In *Electron Devices Meeting (IEDM'03)*, pages 21.4.1–21.4.4, 2003.

[10] A. Mendelson, D. Thiebaut, and D. Pradhan. Modeling live and dead lines in cache memory systems. Technical Report TR-90-CSE-14, Department of Electrical and Computer Engineering, University of Massachusetts, 1990.

[11] S. Mukherjee, J. Emer, T. Fossum, and S. Reinhardt. Cache scrubbing in microprocessors: Myth or necessity? In *10th IEEE Pacific Rim International Symposium on Dependable Computing*, March 2004.

[12] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proc. of 36th IEEE/ACM Intl. Symp. on Microarch. (MICRO 36)*, Dec 2003.

[13] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara. SRAM immunity to cosmic-ray-induced multierrors based on analysis of an induced parasitic bipolar effect. *IEEE Journal of Solid-State Circuits*, 39(5):827–833, May 2004.

[14] N. N. Sadler and D. J. Sorin. Choosing an error protection scheme for a microprocessor's l1 data cache. In *Proc. of 2006 Intl. Conference on Computer Design (ICCD)*, 2006.

[15] D. Tarjan, S. Thoziyoor, and N. Jouppi. CACTI 4.0. http://www.hpl.hp.com/techreports/2006/HPL-2006-86.pdf, 2006.

[16] X. Vera, J. Abella, A. Gonzalez, and R. Ronen. Reducing soft error vulnerability of data caches. In *Workshop on System Effects of Logic Soft Errors (SELSE-3)*, 2007.

[17] W. Zhang. Enhancing data cache reliability by the addition of a small fully-associative replication cache. In *Proc. of 2004 Intl. Conf. on Supercomputing*, pages 12–19, 2004.

[18] J. F. Ziegler, et al. IBM's experiments in soft fails in computer. *IBM Journal of Research and Development*, 40(1), 1996.