Replicating Tag Entries for Reliability Enhancement in Cache Tag Arrays

Shuai Wang, *Member, IEEE*, Jie Hu, *Senior Member, IEEE*, and Sotirios G. Ziavras, *Senior Member, IEEE*

***Abstract*—Protecting on-chip cache memories against soft errors has become an increasing challenge in designing new generation reliable microprocessors. Previous efforts have mainly focused on improving the reliability of the cache data arrays. Due to its crucial importance to the correctness of cache accesses, the tag array also demands high reliability against soft errors. Exploiting the address locality of memory accesses, we propose to duplicate most recently accessed tag entries in a small tag replication buffer (TRB) thus to protect the information integrity of the tag array in the data cache. Experimental results show that our proposed TRB scheme achieves a high 90% access-with-replica (AWR) rate with low performance (~0%), energy (16.3%), and area (19.9%) overheads. We also conduct a detailed design space exploration for the TRB design and propose a selective TRB scheme that achieves a higher AWR rate (97.4%) for the dirty cachelines with negligible overheads. To provide a comprehensive evaluation of the tag-array reliability, we further conduct an architectural vulnerability factor (AVF) analysis for the tag array in the data cache and propose a refined metric, detected-without-replica-AVF (DOR-AVF), which combines the AVF and AWR analysis. Based on our DOR-AVF analysis, a selective TRB scheme with early write-back (S-TRB-EWB) is proposed, which achieves a zero DOR-AVF and 100% AWR rate at a negligible performance overhead. Results from statistical fault/error injection experiment also confirm the effectiveness of our TRB schemes and the achieved reliability of the cache tag array that recovers 100% of detected errors.**

***Index Terms*—Cache tag array, reliability, soft error, tag replication buffer (TRB).**

I. Introduction

IONIZING radiation induced single-event upsets (SEUs), also known as soft errors, in semiconductor memories have been recognized for a long time as a major reliability issue in electronic systems [1], [2]. Due to their large share of the transistor budget and die area, on-chip caches suffer from a significantly higher soft-error rate (SER) than other on-chip components at the current and near future technologies [3]. An incorrect data value once read out from the cache may crash the subsequent computation/communication, external memory, or storage systems, leading to an overall system failure or program inaccuracy. As a critical requirement for reliable computing [4], protecting the information integrity in cache memories has captured a wealth of research efforts [4]–[13].

Recent work has made progress towards the cache vulnerability analysis and reliability optimization based on the analysis [5], [11], [13]–[17]. For example, early write-back schemes [8], [13], [15] were proposed to reduce the vulnerability factor (VF) of dirty cachelines in a write-back data cache. Information redundancy is another fundamental approach in building reliable memory structures. Various coding schemes, such as the parity and ECC codings, are used to enhance information integrity in latches, register files, and on-chip caches, providing different levels of reliability at different performance, energy, and hardware costs. Another form of information redundancy is to maintain redundant copies of the data items in the cache memories [9], [18]. While most of the previous work is targeting at improving the reliability of the data array in on-chip caches, few researchers have directed their attention to the reliability characterization and optimization of the cache tag array. Soft errors occurring in the tag array raise the possibilities of the *false hit* and *false miss* [17], [19]. *False hit* will cause an incorrect execution by loading data from or updating a wrong cacheline. *False miss* in a dirty cacheline will load stale data from the lower cache. Moreover, in a writeback cache, if the tag of a dirty cacheline is flipped by the soft error, the cacheline will be written back to a wrong location in the lower cache. Therefore, due to its crucial importance to the correctness of cache accesses, the tag array demands high reliability against soft errors. Furthermore, as the memory address space increases, e.g., from 32- to 64-bit, the die area of the tag array will also increase, which makes it more vulnerable to soft errors.

The parity coding scheme is widely used to protect the on-chip L1 caches in today's reliable microprocessors such as Intel Itanium [20], Itanium 2 [21], and IBM Power6 [22] processors due to its low cost. However, simple parity coding is only capable of detecting an odd number of bit errors without error recovery capability. On the other hand, error correcting codes (ECCs) typically provide single error correction and double error detection (SEC-DED). Nevertheless, the performance overhead and additional energy consumption due to ECC encoding/decoding make ECC a reluctant choice for high speed on-chip L1 caches [4]. Instead, ECC codings are widely adopted in L2/L3 caches that can tolerate longer access latencies [20]–[22]. The only exception is AMD Opteron server processors [23] in which the data array of the L1 data cache is ECC-protected and its tag array is still parity-protected. Note

Manuscript received May 18, 2010; revised September 12, 2010 and December 08, 2010; accepted January 15, 2011. A preliminary version of this work was presented at the IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2010).

S. Wang is with the Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China (e-mail: swang@nju.edu.cn).

J. Hu and S. G. Ziavras are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: jhu@njit.edu; ziavras@njit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2011.2111469

that soft errors occurring in tag entries of dirty cachelines in the L1 data cache cannot be recovered by the parity coding.

Exploiting the address locality of memory accesses, we propose in this paper a tag replication buffer (TRB), a small buffer that captures and maintains the replicas of frequently accessed tag entries, to enhance the reliability of the tag array in the L1 data cache. We perform a detailed design space exploration for the TRB implementation and propose several optimized schemes to improve the tag array reliability as well as to reduce the area and energy overheads of the TRB. To further improve the protection effectiveness and the provided reliability of the TRB, we then propose a selective TRB (S-TRB) scheme that only duplicates tag entries for dirty cachelines, which demand much higher reliability than clean cachelines. Note that the S-TRB exploits the fact that most latest high-performance microprocessors have L2/L3 caches (both data and tag arrays) protected by ECC codings and also assumes the single-bit error incident to a tag entry if any. Our experimental results show that the S-TRB scheme can achieve a 97.4% access-with-replica (AWR) rate with minimum overheads. In order to provide a comprehensive evaluation on the reliability of the cache tag array, we conduct a cache tag vulnerability factor analysis and propose a refined evaluation metric detected without replica (DOR) AVF that combines the AVF and AWR analysis. Based on our DOR-AVF analysis, a new TRB scheme with early write-back (S-TRB-EWB) triggered by a replacement in the TRB is proposed, which can achieve a 100% AWR rate and a zero DOR-AVF for the tag entries of dirty cachelines with a minimum performance and energy overhead. To further verify the effectiveness of the proposed TRB schemes and the achieved tag array reliability with a TRB, statistical fault/error injection experiment is performed to randomly inject a fixed number of soft errors into the tag array and the replication buffer during the simulation. Our experiment results show that: 1) all errors in accessed erroneous tags are detectable single-bit errors; 2) the base TRB recovers 37.0% and 18.9% of detected errors in clean and dirty cacheline tags; 3) the S-TRB improves the recovery rate to 42.9% for dirty cacheline tags; and 4) the S-TRB-EWB achieves a full recovery rate of 100% for detected errors, which confirms its DOR-AVF analysis.

The rest of this paper is organized as follows. Section II discusses some related work. Section III describes the proposed tag replication buffer (TRB) design. The design space of TRB is studied in Section IV. In Section V, several optimized schemes including the S-TRB are proposed. In Section VI, we analyze the tag array vulnerability to soft errors and propose our new reliability evaluation metric and the derived S-TRB-EWB scheme. The experimental setup and evaluation are presented in Section VII. Section VIII concludes this work.

II. Related Work

The fault behavior of the content addressable memory (CAM) tags has been studied and single-error tolerant solutions were provided in [24]. A functional level framework was also proposed in [25] for implementing a fault-tolerant/self-checking CAM architecture, with a focus on CAM cell designs. Compared to the hardware circuit solutions in [24] and [25], our scheme focuses on the microarchitecture design of the reliable cache. Biswas *et al.* [19] presented some initial efforts on vulnerability analysis of the tag array. Reference [17] proposed a detailed lifetime model to characterize the vulnerability of the tag array in both the L1 data and instruction caches. However, they did not provide any reliability optimization schemes specifically for the tag array. In this paper, we propose a TRB scheme to improve the reliability of the tag array in the data cache by exploiting the address locality of memory accesses. In [26], a caching address tag (CAT) scheme was proposed to reduce the area cost of the cache tag arrays. Due to the CAM implementation of the pointer part in the tag side, the CAT design will incur extremely high energy consumption caused by the CAM search operation in addition to its area overhead, if the tag cache is directly adopted as a replication buffer for reliability improvement. Zhang [27] proposed to add a replication cache (R-Cache) to enhance data cache reliability. R-Cache mainly focuses on the data array reliability improvement in the data cache while ours is on the tag array. R-Cache does not provide any explicit scheme for protecting the tag array, nor can it handle erroneous tag situations. Moreover, R-Cache needs to be updated on every cache write operation while our TRB is only accessed when the copy bit of the original tag entry accessed is zero. With very high access-with-replica rates (>90%), our scheme incurs significantly less frequent TRB accesses and is much more power-efficient.

III. Microarchitecture of the TRB Designs

A. Basics of the TRB Design

Microprocessor issued memory accesses exhibit various localities. The address locality is a form of locality due to the spatial and temporal locality of memory accesses. It means that if a memory address is referenced at a particular time, the same address and its nearby memory addresses are very likely to be referenced in the near future. In other words, only a small set of the memory addresses are referenced during certain execution time intervals. Since the tag entry of a cacheline is the higher portion of the referenced address, it has a better locality property than the full memory address. We call this address locality cache tag locality (CTL). Exploiting the CTL, we propose to duplicate the tag entries in a small cache-like structure, called the tag replication buffer (TRB), to enhance the reliability of the tag array in the data cache.

Our TRB design is an information redundancy based reliable scheme. However, simply keeping two or more identical copies of the tag array is not area and energy efficient. By exploiting the CTL, a small TRB (e.g., 32 entries) can capture most of the tag references. Thus, by keeping the most recently accessed (MRA) tag entries in the small TRB, we can achieve a high AWR rate, providing a high reliability for the tag array. Notice that although the TRB design studied in this work is targeting at the data cache, it also applies well to the instruction cache since the instruction cache has a better locality than the data cache.

B. Microarchitecture of the TRB

One of the key issues in the TRB design is how to locate the replica entry in the TRB and how to identify the original tag with its replica. In [26], a similar caching scheme CAT was proposed to optimize the area of the tag array. Since the CAT

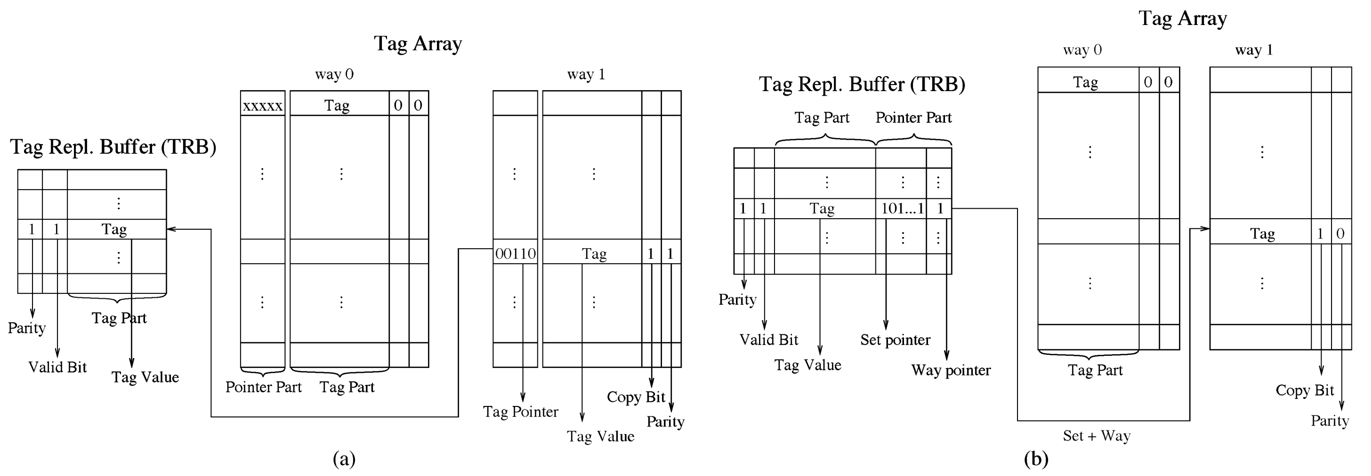


Fig. 1. Block diagrams of the TRB-CAT and TRB-PTR designs. (a) TRB-CAT microarchitecture. (b) TRB-PTR microarchitecture.

scheme aimed at area reduction, it replaces the original tag array with a CAM structure that only stores the pointers to the tag cache (TC) [26]. For our reliability designs, we adopt the pointer design and modify their scheme by putting back the original tag array and using the tag cache for replicating tag entries, as the TRB-CAT design given in Fig. 1(a). The pointer part is at the original tag array side. Each tag entry in the original tag array is associated with a pointer pointing to the location of its replica in the TRB. The TRB entry only contains the tag replica. An additional one bit (copy bit) is added for each original tag entry to indicate whether it has a replica in the TRB or not. In TRB-CAT, accessing the replica of a tag entry is very simple and efficient by directly following the pointer stored with the tag entry. Another advantage of the TRB-CAT design is that multiple entries in the tag array can share the same replica in the TRB, which has been discussed in [26]. To support that, the TRB also needs to be implemented as a CAM structure. When a new replica is to be added to the TRB, a CAM search needs to be performed to check whether the tag value is already in TRB or not. If it is, a pointer is returned to the tag entry in the tag array and this tag entry will share the replica with other tag entries. Otherwise, a TRB replacement shall be performed in case that all TRB entries are occupied. A replaced TRB entry will subsequently invoke a CAM search in the pointer part of the tag array in order to clear the copy bit of tag entries pointing to (sharing) this replica. Notice that the CAM implementation, especially of the pointers in the tag array may incur significantly high area and energy overheads.

To address the potential area and energy issues in the TRB-CAT design, we propose a second TRB-PTR design that moves the pointer part from the tag array to the much smaller TRB side, as shown in Fig. 1(b). In this TRB-PTR design, each entry in the TRB has an additional space to store a pointer. The pointer contains two parts: the set pointer and the way pointer. The set pointer indicates the set of the original tag entry and the way pointer indicates the way of the original tag entry in a set-associative cache. Notice that the way pointer is not needed in a directly-mapped cache. The copy bit in the original tag array is also needed to indicate whether the tag has a replica or not. For both designs, a valid bit is added to each entry in the tag replication buffer to indicate whether it is a valid or invalid tag replica. If a tag entry with replica (its copy bit is set) needs to access its replica in the TRB, its set index and way number are used to perform a CAM search within the pointer part in the small TRB, which shall be power/energy efficient due to the small size. Furthermore, the process of adding a replica to the TRB-PTR is dramatically simplified. If TRB has free (invalid) entries, a tag replica with the set and way pointers is directly created using a free TRB entry. Otherwise, a TRB replacement is required and the selected victim entry simply clears the copy bit in its original tag entry that is directly located by the victim's set and way pointers. It should be noted that TRB-PTR achieves its simplicity and efficiency by avoiding the sharing of a tag replica among multiple tag entries. On the other hand, the replication rate in TRB-PTR might not be as good as in TRB-CAT due to the 1-to-1 mapping (from the replica entry to the tag entry) in the TRB-PTR design. Nevertheless, our experimental results show that the TRB-CAT design incurs an extremely high area and energy overhead, of 39.1% and 172%, respectively, which makes it not a cost-effective TRB design. Thus, in the following sections, we focus on the TRB-PTR design and further explore its design space and optimizations. Note that the terms TRB and TRB-PTR are used interchangeably in the following discussion.

IV. Exploring the Design Space of the TRB

A. *How to Deal With Soft Errors*

In the TRB design, all the tag bits including the original ones and the replicas in the TRB are protected by parity coding. If the single-bit error model is assumed, all errors occurring in the tag array can be detected but not recovered with parity coding. When a tag entry is accessed, the parity checking is performed. If it passes the check, there is no error in the tag. The normal routine of the cache access will continue. If the parity checking fails, the copy bit is examined. If the copy bit is one, it means that this tag has a replica in the TRB. Then, the TRB is accessed and the replica is read out for error recovery. If the replica passes the parity checking, we can recover the original tag by copying back from the replica. If the copy bit is zero or the parity checking of the replica fails, the error in the original tag entry cannot be corrected by the TRB design. Since the probability of error occurring in both the original tag and its replica is extremely low, we use the AWR rate, which is the ratio of the tag accesses with a replica in the TRB over the total number of tag accesses, as one of the evaluation metrics for the TRB design. The higher AWR rate indicates higher reliability of the tag array. Notice that the pointer entry either in the tag array or in the TRB as well as the valid and copy bits are also protected by the parity coding.

B. *When to Duplicate*

A common issue in the information redundant strategy is when to make a redundant copy. In the TRB design, we explore two mechanisms to create replicas: 1) duplicating with a new cacheline (DNC)—making a replica in the TRB when a new tag entry is written into the tag array, i.e., at the time when a new cacheline is brought into the data cache from the L2 cache and 2) duplicating with a TRB miss (DTBM)—making a replica when the original cacheline is hit and there is no replica in the TRB for its tag entry, i.e., the copy bit in its tag entry is zero.

Based on these two mechanisms, we propose two duplication schemes. The first one is the DNC-only scheme in which we only make the replica with a new cacheline. The other one is the DNC+DTBM scheme in which we make the replica in both conditions, i.e., when a new cacheline is written into the data cache or when a hit cacheline does not have a tag replica in the TRB. Basically, if we choose the DNC+DTBM scheme, we can keep more recently accessed tag entries in the TRB to achieve a higher AWR rate. However, the DNC+DTBM scheme will incur more control overhead and energy consumption than the DNC-only scheme.

C. *Replacement Policies in the TRB*

In order to exploit the tag locality, the LRU (least recently used) policy is a good choice to select the victim entry in the TRB. However, the LRU information needs to be updated upon every tag access. Due to the highly-frequent updating operations and the implementation complexity, the LRU policy will incur high energy and area overheads. Therefore, we also study the first-in first-out (FIFO) and Random policies in the TRB, both of which are less expensive to implement. The FIFO policy can be implemented with a queue structure and a head pointer indicating the head of the queue. In the random policy, a random number is generated to determine the victim entry in the TRB.

D. *What to Do Upon Replacement*

In the TRB side, if we need to make a replica for a tag entry while all the entries in the TRB are occupied, a victim entry needs to be selected according to the replacement policy applied. In the TRB-PTR design, no pointer search operation is needed, since the original tag of the victim entry can be directly located by its pointers due to the 1-to-1 mapping property. Then, the copy bits in the located tag entry is cleared to zero. After that, the victim entry along with the pointer bits are replaced by the new value. In the cache side, if a cacheline is to be replaced due to a cache miss and the copy bit in its tag entry is one, the replica in the TRB needs to be located and the valid bit of the replica entry is reset to zero.

V. Optimizing the TRB Design

A. *Improving the Replacement Policy: LRU+ and FIFO+*

In the DNC duplication scheme, when a new cacheline is brought into the cache, a victim entry needs to be selected in the TRB for the replica. Normally, the victim entry is selected through the LRU or FIFO policy as we discussed in Section IV-C. However, most of the new cachelines will cause the replacement of another valid cacheline in the cache. If that cacheline has a tag replica in the TRB, the corresponding replica will be located and invalidated as we discussed in Section IV-D. In that case, it should be beneficial to use that invalidated entry as the victim entry instead of applying LRU or FIFO policy, which may replace valid replicas used by other tag entries. We call this new improved replacement policy LRU+ or FIFO+. To support the LRU+ or FIFO+ replacement policy, the DNC duplication logic is slightly modified to wait till the replacement in the cache side is completed and the victim tag replica in the TRB is located and invalidated if any. Then the new tag value and pointers will be written into the selected TRB entry without performing explicit LRU or FIFO replacement in the TRB.

B. *Tag Value Compression*

From our profiling results using the SPEC CPU2000 benchmark suite, we observed that only part of the entire set of tag bits is needed in order to resolve the tag conflict. These short tags have been identified as the *active tags* and studied for power optimization of tag arrays in [28]. In our simulated processor when running the benchmark suite, the leading (higher) 15 bits of the entire tag entry (33 bits) almost never change during the execution. Therefore, we propose to adopt tag value compression to improve the area and energy efficiencies of our TRB designs.

1) TRB Side Compression (TBSC): To reduce the area and energy overheads of the TRB, we propose our first tag value compression scheme TBSC, shown in Fig. 2. The higher 15 bits of the tag replica in the TRB, which remain unchanged during the execution, are stored in a special register called high tag register (HTR) protected by parity coding. The remaining 18 bits are stored in the TRB similar to the original TRB design. When there is a TRB write operation, we only write the lower 18 bits to the TRB. If we need to recover the error from the replica, the values in the HTR and TRB are read out simultaneously to form the entire tag entry. Since the bit size of the TRB is reduced in this scheme, the area and energy overhead of the TRB will be reduced.

Notice that in this tag value compression scheme, we assume that the higher 15 bits of the tag remain unchanged during the execution, which is based on our profiling results. Therefore, we only need to write the HTR once at the very beginning of the program execution. However, for other applications, the higher 15 bits may change. In that case, we can use compiler support to identify that particular code region similar to [28]. A special instruction can be inserted to disable the TRB scheme during the

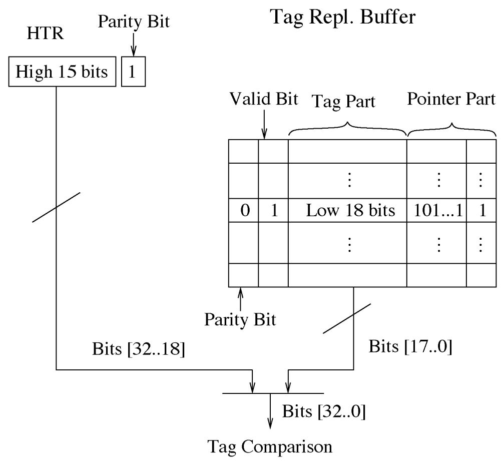


Fig. 2. Block diagram of the TBSC design.

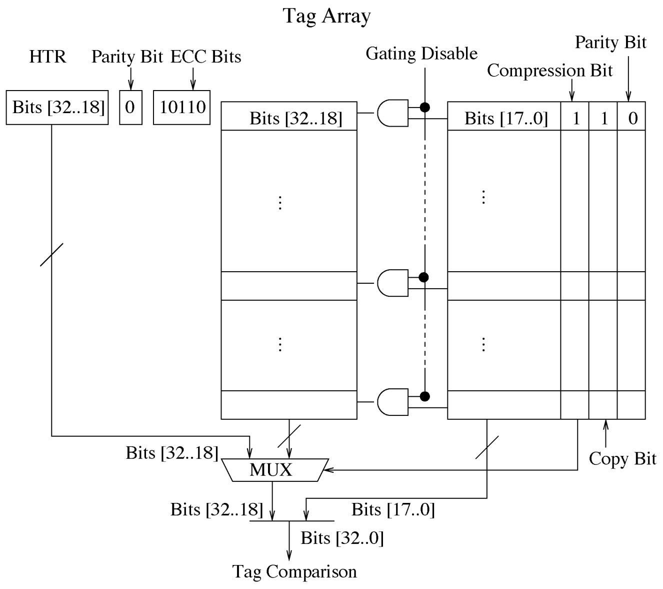


Fig. 3. Block diagram of the TASC design.

execution of that code region, which leaves the tag array unprotected by replicas. Fortunately, this situation rarely happens and the impact on the reliability of our TRB design is negligible.

2) Tag Array Side Compression (TASC): To further reduce the energy consumption in the tag array, we propose the second tag value compression scheme TASC, shown in Fig. 3. Different from TBSC, our TASC scheme moves the HTR from the TRB side to the tag array side. The higher 15 bits of the original tag array are gated for energy savings. The HTR is protected by both parity and ECC codes. During a normal access, the value in the HTR and the lower 18 bits in the tag array are read out, followed by the parity code checking. If the parity checking fails in the HTR, then the ECC code is used to recover from the error, since a single-bit error model is assumed throughout this work. The lower 18 bits or the entire 33 bits of the tag can be protected by the original TRB scheme. Note that if the execution enters the special code regions we discussed above, the gating to the higher 15-bit portion of the tag array is disengaged. In this ungated mode, if a compressed tag is accessed, the tag needs to be restored by using an additional compression bit in the tag entry that selects the readout either from the higher 15-bit tag portion (for uncompressed tags) or from the HTR register (for compressed tags), as shown in Fig. 3. Furthermore, if the TRB only replicates the lower 18 bits of the tag, the TRB has to be disabled for uncompressed tags. However, if the TRB keeps the replicas of the original 33-bit tags, the TRB can be still in active protection when the program enters the aforementioned special regions, which makes the tag compression an independent power/energy/area optimization. Therefore, in our TRB design with this TASC compression, the TRB duplicates the entire 33-bit tags.

C. Selective TRB Schemes for Improving Protection Effectiveness

Recent work [8], [12] claimed that dirty cachelines should have higher priority to be protected than the clean cachelines in a write-back data cache. The clean cachelines in the L1 data cache have their copies in the L2 cache, which can be used to recover from soft errors if the L2 cache is protected by some highly reliable error coding schemes (e.g., ECCs) and is error free, assuming a single bit error model. Unlike the clean cachelines, the dirty cachelines do not have replicas in the L2 cache. Therefore, we need schemes more reliable to protect the dirty cachelines. When it comes to the tag array, the consequences of an error-corrupted tag entry are different from that of a cacheline. If the tag of a clean cacheline was hit by soft errors and is detected by the parity checking during a cache access, this cacheline can be simply invalidated and possible reused to serve a later cache miss. On the other hand, if the tag of a dirty cacheline is soft-error corrupted, the latest data will be lost if the error in the tag is detected but however not recovered. Thus, in terms of reliability requirement, tags of dirty cachelines definitely demand a higher protection. Based on that, we propose a selective TRB (S-TRB) scheme that only duplicates tags of dirty cachelines. This S-TRB scheme is expected to have a better AWR rate compared to the original one, since it reduces the number of tag entries that need to be duplicated. Basically, the less dirty cachelines the data cache has during the execution, the better AWR rate the S-TRB can achieve.

D. Performance Impact

For a regular cache access, tags will be read out and compared with the address tag field. Simultaneously, the parity codes are checked for errors. If there is no error in the tag, the normal routine of the cache access will continue. Meanwhile, in the DTBM scheme, the copy bit will be also checked. If it is 0, the duplication routine will be triggered. Since most of the accessed tags should have replicas and the duplication is not on the critical path of the pipeline, the performance impact due to the duplication is trivial. If the parity checking fails, the error recovery routine discussed above will be triggered, which may hurt the performance. However, due to the extremely low error rate in the real world, the recovery routine is rarely triggered and the performance degradation due to the error recovery should be negligible.

VI. AVF Analysis of Tag Arrays

A. *Lifetime of Tag Arrays*

In order to characterize and optimize the on-chip cache reliability against soft errors, recent papers [5], [11], [13]–[17] present some initial efforts towards cache vulnerability analysis. References [17] and [19] also propose some lifetime models to analyze the vulnerability of the tag array. To evaluate our TRB design, we conduct a similar vulnerability analysis on the tag array based on the lifetime model in [17]. The cache tag vulnerability factor is defined as the average rate of tag items in vulnerable phases over the total tag items that the cache can accommodate during execution. If the single-bit error model is assumed, a false hit [17], [19] will happen only when the tag has one single bit different from the incoming address tag and this particular bit is flipped by a soft error. However, if a tag entry whose original value matches the incoming address tag is hit by a soft error at any bit location, a false miss will happen. Depending on the state of the cacheline, the consequences of a false miss are different. If it is a clean cacheline, a false miss only incurs one extra cache miss without causing any correctness problem. On the other hand, a false miss to a dirty cacheline will cause the loss of all updates to the cacheline and reload a stale copy from the L2 cache, resulting in data loss and erroneous data. Therefore, based on this hamming-distance-one (HDO) analysis [19], the lifetime of a tag entry in a write-back cache can be divided into six phases: `RH`, `FWPL`, `RHFW`, `HFW`, `HPL`, and `Invalid`.

- `RH`: Lifetime phase between the first read and the last Hamming-distance-one (HDO) match of a clean cacheline.
- `FWPL`: Lifetime phase between the first write and the replacement of a dirty cacheline.
- `RHFW`: Lifetime phase between the first read and the last HDO match before the first write of a dirty cacheline.
- `HFW`: Lifetime phase between the last HDO match and the first write of a dirty cacheline.
- `HPL`: lifetime phase between the last HDO match and the replacement of a clean cacheline,
- `Invalid`: Lifetime phase in the invalid state.

Fig. 4 shows the correlation among the lifetime phases for typical tag activities. The `RH`, `FWPL`, and `RHFW` phases are vulnerable because errors occurring in the `RH` and `RHFW` phases will cause false hits, and errors occurring in the `FWPL` phase will cause incorrect writebacks to the L2 cache or erroneous data load. Note that all tag bits in the `FWPL` phase are vulnerable. Phases `HFW`, `HPL`, and *Invalid* are non-vulnerable because errors occurring in the `HFW` phase will only cause a false miss on the first write in a clean cacheline, and errors occurring in the `HPL` phase will be discarded on replacement. Notice that the vulnerability analysis based on this lifetime model is performed at the tag bit level.

B. *DOR AVF*

The above AVF analysis assumes no error protection schemes, such as parity or ECC codings. Note that our TRB design by default is protected by the parity coding. If the single-bit error model is assumed, in the parity-protected tag array, all single-bit errors will be detected by the employed parity coding, and thus the original silent data corruption

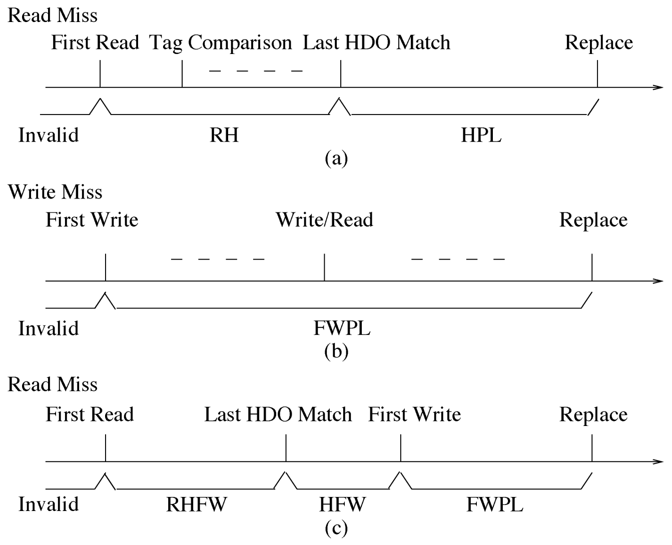


Fig. 4. Tag lifetime of a cacheline in the writeback cache. (a) Clean line. (b) Dirty line. (c) Dirty line.

(SDC) AVF is converted into detected unrecoverable error (DUE) AVF. However, for a clean cacheline with a detected error in its tag, if we assume that the L2 cache is protected by some means of ECC codings and is error free under the single-bit error model, this single-bit error can be recovered by invalidating the cacheline and reloading it from the L2 cache. Thus, these single-bit errors in tag entries of clean cachelines are detected recoverable errors (DREs). If an error hits the tag of a dirty cacheline, the updates to the cacheline will be lost since the tag cannot be recovered, which contributes to DUE. From work [17] and our simulation results, the `FWPL` phase contributes the most to DUE-AVF in the data cache tag array. Therefore, protecting the tag entry of a dirty cacheline should have a much higher priority than that of a clean cacheline.

To evaluate the reliability provided by our TRB design, we introduce a refined AVF metric that combines the vulnerability factor and AWR analysis. It converts the tag DUE-AVF of a dirty cacheline into two categories: DOR AVF and DWR AVF. The DUE-AVF portion of the DWR-AVF is contributed by the fact that the tag replica is also corrupted by soft errors. Since the situation that there are errors in both the tag entry and its replica rarely happens, lower DOR-AVF means higher reliability in the tag array. If the strong assumption is made that an error affected tag can always be recovered from its replica, the DWR-AVF will be converted into the DRE-AVF. A summary of this AVF classification in the context of our TRB schemes is given in Fig. 5.

C. *AWR Versus DOR-AVF*

From the previous discussion, our S-TRB should be a good choice to reduce DOR-AVF of the tag array in the data cache by providing a higher tag replication rate for dirty cachelines. Note that all the duplication and replacement policies studied so far in the TRB design are aiming at protecting the most recently accessed tags by exploiting the address locality. However, the major contributors to the tag AVF are the phases with no access activity [17]. In other words, our S-TRB scheme may not

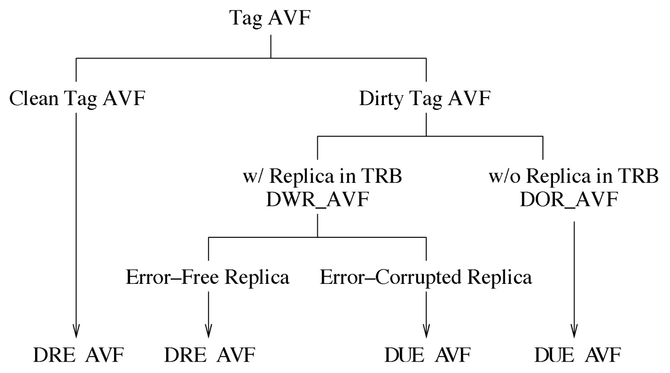


Fig. 5. AVF classification in the context of our TRB designs. A single-bit error model is assumed for both the tag array and the TRB.

have a high replace-with-replica (RWR) rate, which is the ratio of the tag replacements with a replica in the TRB over the total number of tag replacements. To summarize, in order to achieve a high AWR, we need to protect the frequently accessed tags. However, to significantly reduce the DOR-AVF, we may need to give priority to dirty cacheline tags with a long dead time. In other words, an optimized TRB scheme driven by AWR optimizations does not necessarily lead to an optimized DOR-AVF. Therefore, a TRB design that achieves both a high AWR rate (i.e., with priority of protecting most frequently accessed tags) and a low DOR-AVF number (i.e., with priority of protecting most vulnerable tags) is of paramount importance to the provision of high reliability to the tag array.

D. *TRB Replacement Triggered Early Write-Back in Data Cache*

A direct solution to reduce the DUE-AVF is to use write-through data caches. A non-protected write-through data cache has a very low AVF in the tag array. In a parity-protected write-through data cache, the tag array does not have the DUE-AVF since all single-bit errors in the tag array of the data cache can be recovered from the ECC-protected error-free L2 cache. However, due to the performance degradation and the significantly increased accesses to the L2 cache [13], [15], the write-through data cache is not preferred for applications that require high performance and low energy consumption.

In order to achieve both a high AWR and a low vulnerability factor, we propose a new TRB scheme with early write-back (EWB) in the data cache that is triggered by TRB entry replacement. In this S-TRB-EWB scheme, we only duplicate tags of dirty cachelines, which is similar to the S-TRB scheme. When a replica entry in the TRB is replaced in the S-TRB-EWB scheme, its corresponding dirty cacheline will be forced to write back to the L2 cache. Therefore, all the tags of dirty cachelines have their replicas in the TRB and those dirty cachelines that are to lose their replicas in the TRB will become clean due to the early write-back. Since the replacement in the TRB does not occur frequently with a high AWR rate, the S-TRB-EWB scheme incurs much less L2 cache accesses than the write-through scheme. Notice that our early write-back scheme in the data cache is deterministic and not prediction based, which is significantly different from the dead-time based early write-back schemes in [13], [15]. Most importantly, our S-TRB-EWB scheme achieves a 100% AWR rate for dirty cachelines and reduces the DOR-AVF to zero.

TABLE I
PARAMETERS OF THE SIMULATED PROCESSOR

Processor Core	
Datapath Width	4 inst. per cycle
Int Issue Queue	20 entries
FP Issue Queue	15 entries
Load/Store Queue	64 entries
Active list (ACL)	80 entries
Int Register File	80 registers
FP Register File	72 registers
Function Units	4 IALU, 2 IMULT/IDIV 2 FALU, 1 FMULT/FDIV/FSQRT 2 MemPorts
Branch Predictor	
Branch Predictor	Alpha 21264 tournament predictor 32-entry RAS
BTB	2048-entry 2-way
Memory Hierarchy	
L1 I/DCache	64KB, 2 ways, 64B blocks, 2 cycles
L2 UCache	4MB, 8 ways, 128B blocks, 12 cycles
Memory	225 cycles first chunk, 12 cycles rest
TLB	Fully-assoc., 128 entries

VII. EVALUATION

A. *Experimental Setup*

We derive our simulators from SimpleScalar V3.0 [29] to model a contemporary high-performance microprocessor similar to Alpha 21364. In the new simulator, the original register update unit (RUU) structure is replaced by a separated integer issue queue, a floating-point issue queue, an integer register file, a floating-point register file, and the active list (a.k.a., the reorder buffer). A MIPS R10000 style register renaming scheme is adopted in our implementation. The data cache is virtually tagged with a 48-bit address and uses the write-back policy by default. Table I gives the detailed configuration of the simulated microprocessor. Cacti 6.5 [30] is used for area estimation and energy profiling (at 32-nm technology) during the simulation.

For experimental evaluation, we use the SPEC CPU2000 benchmark suite compiled for the Alpha instruction set architecture using the "-arch ev6-non_shared" option with "peak" tuning. We use the reference input sets for this study. Each benchmark is first fast-forwarded to its early single simulation point (*gap* and *ammp* use the standard single simulation point instead of the very large early single simulation point) specified by SimPoint [31]. We use the last 100 million instructions during the fast-forwarding phase to warm-up the caches if the number of skipped instructions is more than 100 million. Then, we simulate the next 100 million instructions in detail.

B. *Design Space Exploration*

1) TRB Duplication Policies: DNC-Only Versus DNC+DTBM: In our TRB design, we have two duplication policies. One is DNC-Only policy, which performs the duplication only when a new cacheline is written into the data cache. The other is DNC+DTBM policy, which makes the duplication not only when a new cacheline is written into the data cache but also when a hit cacheline does not have

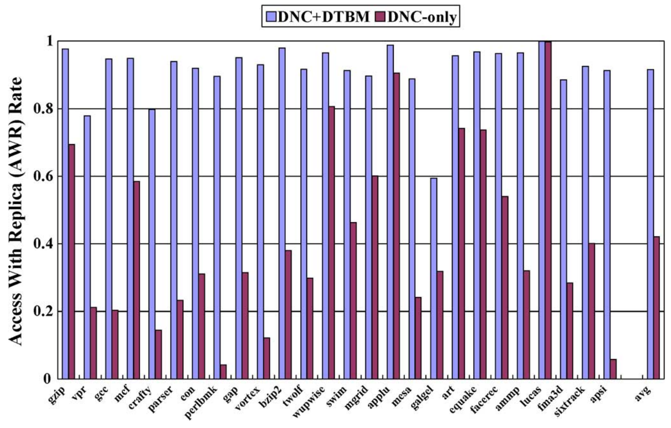


Fig. 6. AWR rate comparison of the TRB with different duplication policies.

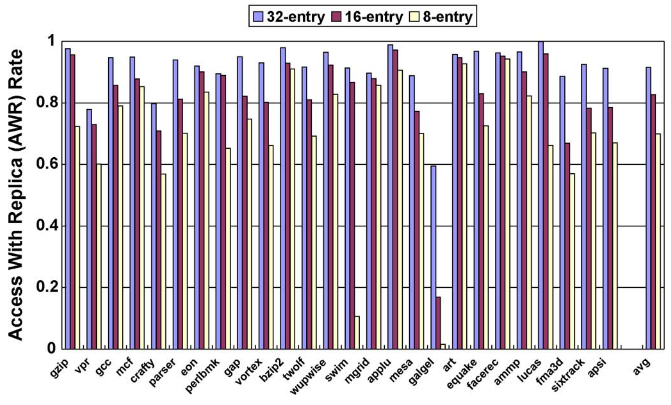


Fig. 7. AWR rate comparison of the TRB with different sizes.

a tag replica in the TRB. To evaluate these two duplication policies, we use the LRU replacement policy in a 32-entry TRB. Fig. 6 shows that the average AWR rate of the TRB with the DNC+DTBM policy is 91.5% compared to 42.1% with the DNC-Only policy. Therefore, to achieve a high AWR rate, the DNC+DTBM policy is preferred. We use DNC+DTBM as the default policy in the following study.

2) TRB Sizes: Fig. 7 shows the AWR rates for different TRB sizes. An 8-entry TRB only has a 69.9% AWR rate and the AWR rate of a 16-entry TRB is increased to 82.7%. In comparison, a 32-entry TRB achieves a very high AWR rate, 91.5% on the average. If we further increase the size of the TRB, the area overhead of the TRB and the energy consumption in TRB accesses will dramatically increase. Therefore, to balance the achieved AWR rate and the incurred overheads, we choose the 32-entry TRB in our TRB design.

3) TRB Replacement Policies: In the previous sections, in order to study different design schemes and duplication policies in the TRB, we assume the LRU as the default TRB replacement policy. Although the LRU policy is good at exploiting the property of locality, we need a more cost-effective policy due to LRU's implementation complexity. Therefore, we compare three replacement policies, LRU, FIFO, and Random. The results in Fig. 8 show that the LRU policy achieves the highest AWR rate, 91.5%, while the AWR rate of the FIFO policy is 90.0%. The Random policy has the lowest AWR rate, 88.1%.

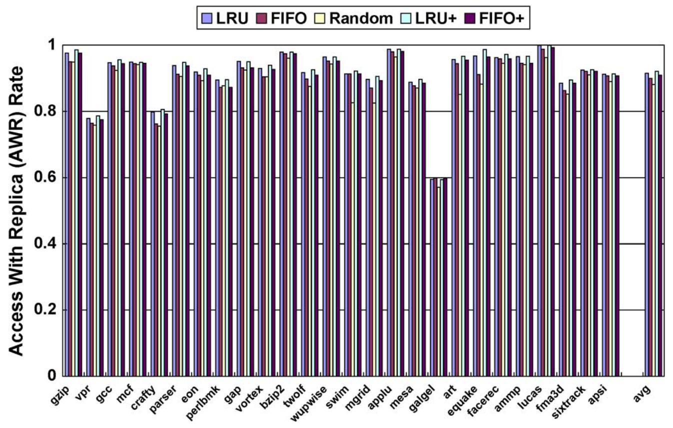


Fig. 8. AWR rate comparison of the TRB with different TRB replacement policies.

TABLE II
COMPARISON OF THE ECC, FD, TRB-CAT, AND TRB SCHEMES

	ECC	FD	TRB-CAT	TRB
Performance Loss	2%	0%	0%	0%
Area Overhead	21%	106%	39.1%	16.3%
Energy Overhead	118%	44.2%	172%	19.9%
AWR Rate	-	100%	92.5%	90.0%

The LRU information needs to be updated upon every tag access. Therefore, in a cost-effective design, the FIFO policy is preferred. We use the FIFO as the default TRB replacement policy in our following study.

C. Comparison to Related Work

To provide a comprehensive analysis of our TRB (i.e., TRB-PTR) design, we compare it with the ECC, TRB-CAT, and full duplication (FD) schemes. The FD scheme maintains an identical copy of the tag array. All the tag values in TRB-CAT, TRB, and FD are protected by the parity coding. A 32-entry TRB with the FIFO and DNC+DTBM policy is used in our TRB designs. Table II shows the comparison of these four schemes in terms of performance, area, and energy overheads as well as the AWR rate. We assume that the parity checking can be overlapped with the tag comparison and will not cause additional delay to the cache access. For ECC coding, we use a (33, 40) coding scheme for each tag entry (33 bits) and optimistically assume one additional cycle delay in cache access. The energy number for parity and ECC coding is scaled from [9]. The area overhead is estimated with a modified Cacti 6.5 [30]. For the CAM estimation of the pointer part in the TRB-CAT and TRB designs, we utilize the implementation of the tag matching in a fully-associative cache from Cacti.

Table II shows that because of the performance degradation (2%) and the high energy overhead (118%), ECC is not a good choice for protecting the on-chip L1 cache in high-performance processors. The 106% area overhead in the full duplication (FD) scheme also makes it an inefficient design. Although compared to the TRB scheme, the entry sharing property in the TRB-CAT scheme results in a higher AWR rate (92.5%), the high energy (172%) and area (39.1%) overheads are still not acceptable. In

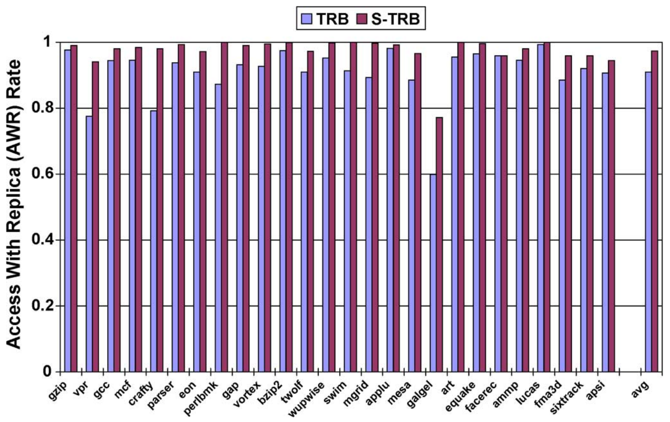


Fig. 9. AWR rate comparison between the original TRB and S-TRB schemes.

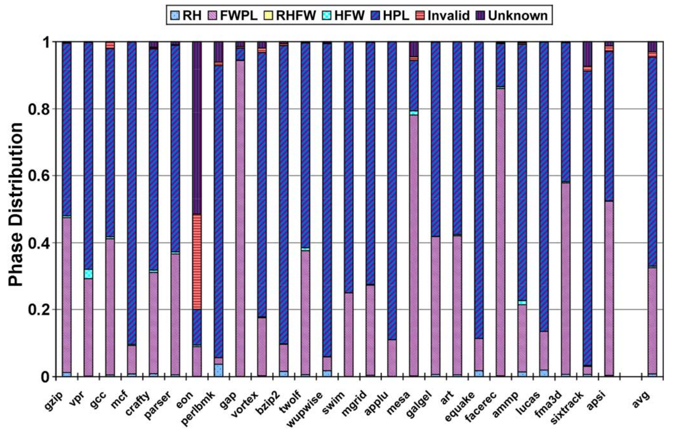


Fig. 10. Lifetime distribution for the tag array in the write-back data cache.

conclusion, our TRB design achieves the lowest area and energy overheads among these four schemes with an AWR rate of 90.0%. Note that in the ECC scheme, although the achieved reliability cannot be measured by the AWR rate, all the single-bit errors occurring in the tag entry can be recovered.

D. TRB Optimization Schemes

1) LRU+ and FIFO+: To improve the TRB replacement policy, we propose the LRU+ and FIFO+ in Section V-A. Experimental results show that the LRU+ and FIFO+ can improve the AWR by 0.6% and 1.0% on the average (shown in Fig. 8) without noticeable overheads. Therefore, we will use the FIFO+ policy in the following study.

2) TBSC and TASC: The TRB side compression (TBSC) scheme targets at reducing the area and energy overheads in the TRB. Experimental results show that TBSC reduces the area and energy overheads to 13.1% and 16.9%, compared to 16.3% and 19.9% in the original TRB scheme. Tag array side compression (TASC) scheme targets at reducing the energy overhead in the tag array. From our experimental results, the energy consumption in the tag array access is reduced by 17% due to the gating scheme. If we consider the overall energy consumption in the TASC scheme, it will remain almost the same compared to a conventional cache, which offsets the energy overhead incurred by the TRB. For the following experiments, our TRB design by default employs the TASC scheme.

3) S-TRB: In order to study the effectiveness of our S-TRB, we first conduct a profiling on the cacheline distribution in a write-back data cache. Our results show that on the average only 33% of the cachelines in the data cache are dirty during program execution. The clean cachelines account for 66% and the rest are invalid (not in use). By duplicating and maintaining the tags of only dirty cachelines in the TRB, the S-TRB should deliver a much higher AWR rate due to the virtually tripled TRB size. Fig. 9 shows that the AWR rate of our selective scheme is increased to 97.4% compared to the 91.0% in the original scheme. Notice that the results are based on the 32-entry TRB with the DNC+DTBM and FIFO+ policy.

4) Tag Array AVF Analysis: Fig. 10 shows the phase distribution of the tag array in a write-back data cache. About 0.76% of the tag lifetime is in the `RH` and `RHFW` phases. The `FWPL` phase contributes about 31.7%. Therefore, the total AVF of the tag array is about 32.5%. Notice that we use the bit-level analysis for tag AVF characterization without any tag protection schemes.

5) TRB With Early Write-Back (S-TRB-EWB): To evaluate the reliability of our TRB design, we have introduced a new metric, DOR-AVF. Fig. 11 shows that the TRB and S-TRB schemes have reduced the tag DOR-AVF of dirty cachelines from 31.7% to 22.6% and 16.7%, respectively. This moderate improvement of the DOR-AVF is mainly due to the fact that TRB is not directly optimizing the long `FWPL` phase. As shown in Fig. 12, the RWR rates stay low, 18.3% for the TRB and 51.4% for the S-TRB. However, if we use a write-through cache, the performance will degrade 3.7% and the energy consumption of the L2 cache will be more than doubled compared to that in a write-back cache, as shown in Figs. 13 and 14. By early writing back dirty cachelines triggered by the TRB replacement, S-TRB-EWB achieves a 100% AWR rate and a 100% RWR rate, consequently delivering a zero DOR-AVF for the tags of dirty cachelines. As shown in Fig. 13, S-TRB-EWB with a 32-entry TRB incurs a negligible performance loss (<0.01%). Notice that reducing the TRB size will cause more early write-back operations to the L2 cache. To further study the energy consumption in our S-TRB-EWB design, we conduct the simulation with different TRB sizes. Fig. 14 shows that the S-TRB-EWB with a 32-entry TRB only incurs a 9.7% energy increase in the L2 cache. If we further decrease the TRB size to 16-entry or 8-entry, the energy consumption will increase by 18.8% or 27.9% compared to that in the write-back cache.

E. Statistical Error Injection

In order to evaluate our refined AVF metric and the TRB design, we also conduct statistical error injection during the execution-driven simulation. The soft error injection flips one bit or multiple bits in the selected tag entries in the tag array or the TRB. For each benchmark, 100 errors are randomly injected into tag entries or replicas during the execution. Notice that errors injected in our simulation are accelerated and with higher rate than real ones. As a general way to perform architectural-level error injection [8], accelerated error numbers are assumed in order to expose the error behavior and evaluate the reliability of the system. Although the accelerated error number

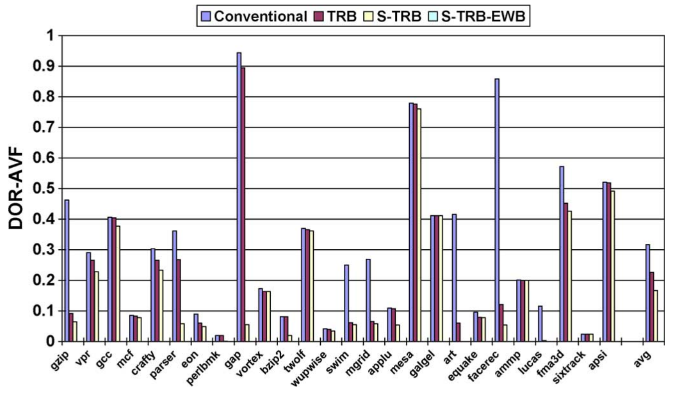


Fig. 11. DOR-AVF comparison for the parity-protected tags of the cache-lines with write operations among conventional, original TRB, S-TRB, and S-TRB-EWB data caches.

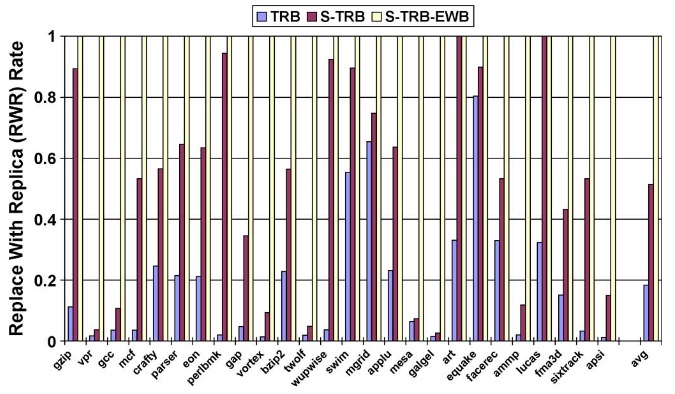


Fig. 12. RWR rate of the original TRB, S-TRB, and S-TRB-EWB schemes.

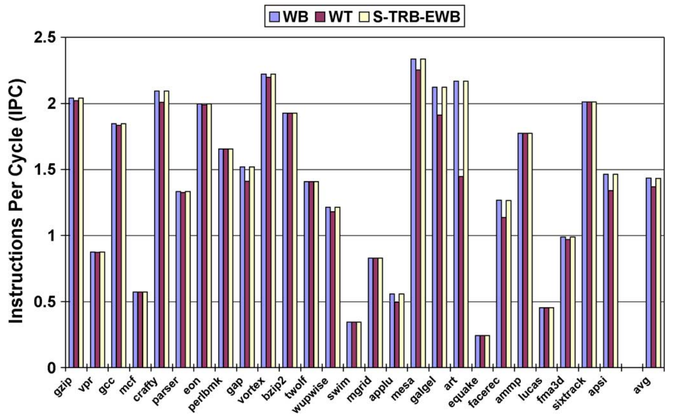


Fig. 13. Comparison of performance among conventional write-back (WB), write-through (WT), and S-TRB-EWB data caches.

is adopted, there are no double/multiple-bit errors occurring in our simulation.

First, we conduct the error injection for a conventional write-back data cache to verify our AVF analysis of the tag array. Table III shows that on average 31 (out of 100) error are injected into the `FWPL` phase that is the major contributor to the tag AVF and 63.5 (out of 100) errors are injected into the `HPL` phase that does not contribute to the tag AVF. The results are consistent with our AVF analysis presented in Fig. 10. Then, we conduct the error injection for our original TRB, S-TRB,

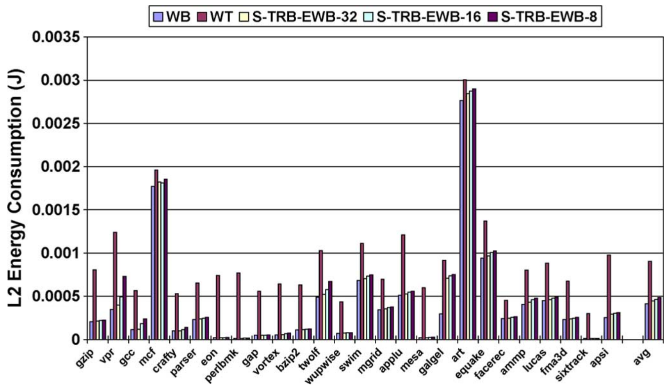


Fig. 14. Comparison of L2 cache energy consumption among conventional write-back (WB), write-through (WT), and S-TRB-EWB data caches.

TABLE III
DISTRIBUTION OF INJECTED ERRORS AMONG DIFFERENT LIFETIME PHASES

Bechmarks	RH	FWPL	RHFW	HFW	HPL	Invalid	Unknown
gzip	1	42	0	1	56	0	0
vpr	0	27	0	2	71	0	0
gcc	0	37	0	1	60	2	0
mcf	1	7	0	0	92	0	0
crafty	1	28	0	1	68	0	2
parser	0	37	0	1	61	0	1
eon	0	9	0	0	11	29	51
perlbmk	4	2	0	0	87	1	6
gap	0	90	0	0	9	0	1
vortex	0	18	0	0	79	1	2
bzip2	1	10	0	0	87	1	1
twolf	1	37	0	1	61	0	0
wupwise	2	5	0	0	93	0	0
swim	0	21	0	0	79	0	0
mgrid	0	27	0	0	73	0	0
applu	0	11	0	0	89	0	0
mesa	0	77	0	1	17	1	4
galgel	1	41	0	0	58	0	0
art	0	42	0	0	58	0	0
equake	2	10	0	0	88	0	0
facerec	0	86	0	1	13	0	0
ammp	1	19	0	1	79	0	0
lucas	2	12	0	0	86	0	0
fma3d	1	57	0	0	42	0	0
sixtrack	1	3	0	0	88	1	7
apsi	0	51	0	0	46	2	1
avg	0.731	31	0	0.385	63.5	1.46	2.92

and S-TRB-EWB schemes. The results are shown in Table IV. For the original TRB, the error recovery rate is 37.0% for the clean-cacheline tags and 18.9% for the dirty-cacheline tags. The error recovery rate is defined as the total number of errors that can be recovered by the TRB over the total number of detected errors. The error recovery rate in S-TRB increases to 42.9% and the S-TRB-EWB has a 100% recovery rate, which also confirm the effectiveness of our design and the achieved reliability. Notice that the recovery rate in S-TRB and S-TRB-EWB are only for tags of dirty cachelines.

VIII. CONCLUSION

In this paper, we proposed a TRB design by exploiting the memory address locality to protect the tag array of the on-chip data cache against soft errors. Several optimized schemes such

TABLE IV
COMPARISON OF DETECTED AND RECOVERED ERRORS AMONG ORIGINAL TRB, S-TRB AND S-TRB-EWB SCHEMES

Benchmarks	TRB						S-TRB			S-TRB-EWR		
	Clean Tag			Dirty Tag			Dirty Tag			Dirty Tag		
	Detected	Recovered	Rate	Detected	Recovered	Rate	Detected	Recovered	Rate	Detected	Recovered	Rate
gzip	12	5	41.8%	48	6	12.5%	43	18	41.9%	51	51	100%
vpr	10	6	60.0%	32	1	3.13%	32	1	3.13%	6	6	100%
gcc	13	4	30.8%	35	2	5.71%	34	4	11.8%	36	36	100%
mcf	2	1	50.0%	16	1	6.25%	7	3	42.9%	10	10	100%
crafty	47	19	40.4%	29	7	24.1%	29	16	55.2%	27	27	100%
parser	21	14	66.7%	34	7	20.6%	34	18	52.9%	33	33	100%
eon	36	15	41.8%	39	5	12.8%	38	5	13.2%	36	36	100%
perlbmk	6	2	33.3%	10	0	0.00%	3	3	100%	5	5	100%
gap	1	0	0.00%	97	6	6.19%	93	22	23.7%	96	96	100%
vortex	16	5	31.3%	17	0	0.00%	24	4	16.7%	12	12	100%
bzip2	11	5	45.5%	10	2	20.0%	9	5	55.6%	6	6	100%
twolf	15	6	40.0%	35	1	2.86%	41	3	7.32%	17	17	100%
wupwise	7	3	42.9%	8	1	12.5%	5	5	100%	2	2	100%
swim	3	1	33.3%	28	15	53.6%	23	16	69.6%	23	23	100%
mgrid	6	2	33.3%	26	17	65.4%	19	13	68.4%	25	25	100%
applu	1	0	0.00%	18	7	38.9%	7	4	57.1%	10	10	100%
mesa	7	3	42.9%	82	5	6.10%	76	8	10.5%	72	72	100%
galgel	17	7	41.2%	39	1	2.56%	32	2	6.25%	6	6	100%
art	2	1	50.0%	46	15	32.6%	17	16	94.1%	48	48	100%
equake	3	1	33.3%	13	10	76.9%	10	9	90.0%	11	11	100%
facerec	3	1	33.3%	83	29	34.9%	88	24	27.3%	90	90	100%
ammp	11	3	27.3%	19	1	5.26%	25	3	12.0%	8	8	100%
lucas	1	0	0.00%	10	3	30.0%	8	7	87.5%	9	9	100%
fma3d	6	3	50.0%	57	9	15.8%	36	15	41.7%	70	70	100%
sixtrack	4	2	50.0%	8	0	0.00%	8	1	12.5%	7	7	100%
apsi	9	4	44.4%	53	1	1.89%	49	7	14.3%	39	39	100%
avg	10.4	4.35	37%	34.3	5.85	18.9%	30.4	8.92	42.9%	29.0	29.0	100%

as TBSC, TASC, and S-TRB are proposed to further improve the reliability and reduce the energy and area overheads in our TRB designs. Our simulation results show that the S-TRB scheme with the DNC+DTBM duplication and FIFO+ replacement policies achieves a high AWR rate of 97.4% for the tags of dirty cachelines, at a moderate hardware overhead. To further characterize and optimize the reliability of the cache tag array, we conducted the AVF analysis and proposed a refined evaluation metric DOR-AVF that combines the vulnerability factor and AWR analysis. Based on our DOR-AVF analysis, we proposed the S-TRB-EWB scheme to further improve the tag array reliability by optimizing the contradicting AVF and AWR simultaneously. Our experimental evaluation shows that the S-TRB-EWB scheme achieves a 100% AWR rate and a zero DOR-AVF for the tags of dirty cachelines at a negligible performance loss and a minor energy overhead. Furthermore, our statistical-error-injection experiment results reassure us the AVF number from our lifetime based tag vulnerability analysis and the achieved tag reliability by the TRB schemes. These results also confirm that our TRB schemes can be an effective solution to protect the tag arrays of on-chip caches for high-performance reliable microprocessors.

REFERENCES

[1] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt, "Techniques to reduce the soft errors rate in a high-performance microprocessor," in *Proc. 31st Annu. Int. Symp. Comput. Arch.*, 2004, pp. 264–275.

[2] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, M. Nicewicz, C. A. Russell, W. Y. Wang, L. B. Freeman, P. Hosier, L. E. LaFave, J. L. Walsh, J. M. Orro, G. J. Unger, J. M. Ross, T. J. O'Gorman, B. Messina, T. D. Sullivan, A. J. Sykes, H. Yourke, T. A. Enger, V. Tolat, T. S. Scott, A. H. Taber, R. J. Sussman, W. A. Klein, and C. W. Wahaus, "IBM experiments in soft fails in computer electronics (1978–1994)," *IBM J. Res. Develop.*, vol. 40, no. 1, pp. 3–18, Jan. 1996.

[3] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. Int. Conf. Depend. Syst. Netw.*, Jun. 2002, pp. 389–398.

[4] S. Kim and A. Somani, "Area efficient architectures for information integrity checking in cache memories," in *Proc. Int. Symp. Comput. Arch.*, May 1999, pp. 246–255.

[5] H. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Vulnerability analysis of L2 cache elements to single event upsets," in *Proc. Des., Autom., Test Eur.*, Mar. 2006, pp. 1–6.

[6] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe, "Multi-bit error tolerant caches using two-dimensional error coding," in *Proc. 40th IEEE/ACM Int. Symp. Microarch.*, Dec. 2007, pp. 197–209.

[7] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin, "Soft errors issues in low-power caches," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 10, pp. 1157–1166, Oct. 2005.

[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. Int. Symp. Low Power Electron. Des.*, 2004, pp. 132–137.

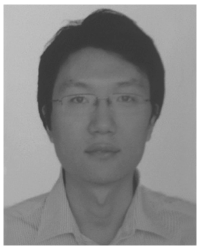
[9] R. Phelan, "Addressing soft errors in ARM core-based SOC," ARM Ltd., San Jose, CA, ARM White Paper, Dec. 2003.

[10] N. N. Sadler and D. J. Sorin, "Choosing an error protection scheme for a microprocessor L1 data cache," in *Proc. Int. Conf. Comput. Des.*, Oct. 2006, pp. 499–505.

[11] V. Sridharan, H. Asadi, M. B. Tahoori, and D. Kaeli, "Reducing data cache susceptibility to soft errors," *IEEE Trans. Depend. Secure Comput.*, vol. 3, no. 4, pp. 353–364, Oct. 2006.

[12] S. Wang, J. Hu, and S. G. Ziavras, "Self-adaptive data caches for soft-error reliability," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 8, pp. 1503–1507, Aug. 2008.

[13] W. Zhang, "Computing cache vulnerability to transient errors and its implication," in *Proc. 20th IEEE Int. Symp. Defect Fault Toler. VLSI Syst.*, Oct. 2005, pp. 427–435.
[14] G. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Balancing reliability and performance in the memory hierarchy," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Mar. 2005, pp. 269–279.
[15] S. Wang, J. Hu, and S. G. Ziavras, "On the characterization of data cache vulnerability in high-performance embedded microprocessors," in *Proc. 6th Int. Conf. Embed. Comput. Syst.: Arch., Model., Simulation (SAMOS VI)*, Jul. 2006, pp. 14–20.
[16] J. Yan and W. Zhang, "Evaluating instruction cache vulnerability to transient errors," *ACM SIGARCH Comput. Arch. News*, vol. 35, no. 4, pp. 21–28, Sep. 2007.
[17] S. Wang, J. Hu, and S. G. Ziavras, "On the characterization and optimization of on-chip cache reliability against soft errors," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1171–1184, Sep. 2009.
[18] W. Zhang, S. Gurumurthi, M. Kandemir, and A. Sivasubramaniam, "ICR: In-cache replication for enhancing data cache reliability," in *Proc. Int. Conf. Depend. Syst. Netw.*, 2003, pp. 291–300.
[19] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan, "Computing architectural vulnerability factors for address-based structures," in *Proc. IEEE Int. Symp. Comput. Arch.*, Jun. 2005, pp. 532–543.
[20] N. Quach, "High availability and reliability in the itanium processor," *IEEE Micro*, vol. 20, no. 5, pp. 61–69, 2000.
[21] C. McNairy and D. Soltis, "Itanium 2 processor microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 44–55, 2003.
[22] K. Reick, P. N. Sanda, S. Swaney, J. W. Kellington, M. J. Mack, M. S. Floyd, and D. Henderson, "Fault-tolerant design of the IBM power6 microprocessor," *IEEE Micro*, vol. 278, no. 2, pp. 30–38, Mar.–Apr. 2008.
[23] AMD, Sunnyvale, CA, "Bios and Kernel Developer's Guide for Amd Athlon 64 and Amd Opteron Processors," 2006.
[24] J.-C. Lo, "Fault-tolerant content addressable memory," in *Proc. Int. Conf. Comput. Des.*, 1993, pp. 193–196.
[25] F. Salice, M. Sami, and R. Stefanelli, "Fault-tolerant cam architectures: A design framework," in *Proc. 17th IEEE Int. Symp. Defect Fault-Toler. VLSI Syst.*, 2002, pp. 233–244.
[26] H. Wang, T. Sun, and Q. Yang, "Cat—Caching address tags: A technique for reducing area cost of on-chip caches," in *Proc. 22nd Annu. Int. Symp. Comput. Arch.*, 1995, pp. 381–390.
[27] W. Zhang, "Enhancing data cache reliability by the addition of a small fully-associative replication cache," in *Proc. Int. Conf. Supercomput.*, 2004, pp. 12–19.
[28] P. Petrov and A. Orailoglu, "Tag compression for low power in dynamically customizable embedded processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 7, pp. 1031–1047, Jul. 2004.
[29] D. Burger and T. M. Austin, "The Simplescalar Tool Set, Version 2.0," Comput. Sci. Dept., Univ. Wisconsin, Madison, Tech. Rep. 1342, 1997.
[30] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," HP Laboratories, 2009.
[31] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proc. 10th Int. Conf. Arch. Support for Program. Lang. Operat. Syst.*, Oct. 2002, pp. 45–57.



Shuai Wang (S'07–M'10) received the B.S. degree in computer science from Nanjing University, Nanjing, China, in 2003, and the Ph.D. degree in computer engineering from New Jersey Institute of Technology, Newark, in 2010.

He is currently an Assistant Professor with the Department of Computer Science and Technology, Nanjing University. His research interests include computer architecture, reliable circuits and systems, power/thermal-aware systems design, reconfigurable computing architectures, embedded systems, and on-chip networks.

Dr. Wang is a member of the IEEE Computer Society, ACM, and ACM SIGARCH.

Jie Hu (S'02–M'04–SM'11) received the B.E. degree in computer science and engineering from Beijing University of Aeronautics and Astronautics, Beijing, China, in 1997, the M.E. degree in signal and information processing from Peking University, Beijing, China, in 2000, and the Ph.D. degree in computer science and engineering from the Pennsylvania State University, University Park, in 2004.

He has been an Assistant Professor with the Electrical and Computer Engineering Department, New Jersey Institute of Technology, since 2004. His research interests include the areas of computer architecture, power-aware systems design, power-efficient memory hierarchy, high-performance microprocessors, complexity-effective processor microarchitecture, power-efficient reliable systems, compiler optimizations for performance and power consumption, and reconfigurable computing architecture. He is a member of the ACM.

Sotirios G. Ziavras (S'83–M'90–SM'96) received the Diploma in electrical engineering from the National Technical University of Athens, Greece, in 1984, the M.Sc. in electrical and computer engineering from Ohio University, Athens, in 1985, and the D.Sc. degree in computer science from George Washington University, Washington, DC, in 1990.

He was with the Center for Automation Research, University of Maryland, College Park, from 1988 to 1989. He was a visiting Professor with George Mason University in 1990. He is currently a Professor the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark. He has published over 150 research papers. He is listed, among others, in the Marquis Who's Who in Science and Engineering, Who's Who in America, Who's Who in the World, and Who's Who in the East. His main research interests include advanced computer architecture, reconfigurable computing, embedded computing systems, and parallel and distributed computing.