

# ICR: In-Cache Replication for Enhancing Data Cache Reliability \*

Wei Zhang    Sudhanva Gurusurthi    Mahmut Kandemir    Anand Sivasubramaniam  
Department of Computer Science & Engineering, The Pennsylvania State University  
University Park, PA 16802, USA  
{wzhang,gurumurt,kandemir,anand}@cse.psu.edu

## Abstract

*Processor caches already play a critical role in the performance of today's computer systems. At the same time, the data integrity of words coming out of the caches can have serious consequences on the ability of a program to execute correctly, or even to proceed. The integrity checks need to be performed in a time-sensitive manner to not slow down the execution when there are no errors as in the common case, and should not excessively increase the power budget of the caches which is already high. ECC and parity-based protection techniques in use today fall at either extremes in terms of compromising one criteria for another, i.e., reliability for performance or vice-versa. This paper proposes a novel solution to this problem by allowing in-cache replication, wherein reliability can be enhanced without excessively slowing down cache accesses or requiring significant area cost increases. The mechanism is fairly power efficient in comparison to other alternatives as well. In particular, the solution replicates data that is in active use within the cache itself while evicting those that may not be needed in the near future. Our experiments show that a large fraction of the data read from the cache have replicas available with this optimization.*

## 1 Introduction

Recent studies [9, 26, 22, 11] have shown that transient hardware errors caused by external factors such as alpha particles and cosmic ray strikes are responsible for a large percentage of system downtime. Denser processing technologies, high clock speeds and low supply voltages can worsen this problem. Consequently, transient-error conscious system design is becoming a necessity for reliable functioning of hardware.

While transient errors can affect all hardware circuits, the cache is particularly more susceptible of the on-chip components. In today's microprocessors, over 60% of the on-chip real estate is taken by caches, making them more vulnerable to cosmic ray strikes. Further, the high cache leakage power concern is leading to aggressive optimization techniques today that can increase errors. Errors in cache memories can lead to severe consequences on the execution, since they are very close to the main processing unit and can easily propagate to other components through the CPU and register file.

Currently the popular error checking schemes for caches use either byte-parity, which attaches one bit parity per eight-bit data, and SEC-DED (Single Error Correction, Double Error Detection). When we provision such schemes for transient error detection/recovery, there are three problems that arise. The first is the extra space that is needed for storing the redundant information. For instance, 12.5% extra overhead is incurred in storing an extra parity bit for each eight bit data, and the same overhead is incurred for maintaining an 8 bit SEC-DED for a 64-bit entity. The second, and perhaps more important, problem is the latency for cache operations. Typically, we want caches to serve a request within a processor cycle so that the pipeline of the datapath can be effectively utilized. While simple schemes such as parity may be able to meet such demands, more complex ECC based schemes may find it difficult to sustain such speeds (unless caches themselves are deeply pipelined and there is a steady stream of requests to the cache so that the bandwidth issue is more important than the latency). While this may not pose a problem in low-end (embedded) processors that are clocked at slow frequencies, it is certainly not feasible to provide single cycle latencies for caches of high-end processors clocked over 1GHz. As a result, there is an extra cycle incurred for reliability checks — particularly on a load where this may be in the critical path — which can be an overkill if error rates are not very high. Even if one very pessimistically assumes an error every million cycles [11], this is a high price to pay for the normal functioning. Another reason why the latency issue is more important than the space issue is because there have been recent studies that have shown effective techniques for compressing the redundant information. For example, Kim and Somani [11] propose a very small cache to store parity information or to duplicate recently used data and demonstrate very good hit rates for this cache with such a scheme. The third problem is the additional power consumption for maintaining informational redundancy and verifying data integrity, and this power is much higher for SEC-DED mechanisms as compared to parity [1].

This paper proposes a new mechanism for *data caches (dL1)* to enhance their reliability without compromising on performance. Note that error detection and correction is more critical for data caches (which can be written into), while detection may suffice for instruction caches which are mainly read-only. We refer to our mechanism as ICR (*in-cache replication*), wherein we use the existing cache space to hold replicas of a cache block. This mechanism can be used either with ECC or with parity based schemes. When we use such a mechanism together with ECC, we can get better performance and/or reliability than with a scheme that uses ECC uniformly for all cache lines. Similarly, this mechanism can also be used with parity, to provide higher detection and recovery capabilities without degrading per-

---

\*This research has been supported in part by NSF grants: 0103583, 0130143, and 0093082.

formance or power consumption significantly.

The basic idea for ICR is to perform replication without evicting blocks that may be needed in the near future so that there is no significant performance degradation (by cache misses). For this purpose, we use a dead block prediction strategy [10], wherein blocks that have not been used for a while are declared to be dead and their space in the L1 cache is recycled to maintain duplicates for the blocks in active use. Dead block prediction has been used in prior studies [10, 14, 7] for prefetching and cache leakage optimizations, and this is the first study to explore this idea for cache reliability. This paper uses this basic idea and shows several ways by which replication can be accomplished, both across the ways of a cache set, as well as across sets themselves. We investigate a large design space of when such replication should be done, how aggressive such replication should be, how many replicas to provide, how cache lookup should be done in the presence of replicas, and how we can check for errors in unreplicated blocks.

Transient errors are quite rare, and it is important that one does not pay a performance penalty or power cost in the common case when these errors do not occur. Even if the errors do occur, multi-bit errors are rare, in which case the scheme that we propose can catch single-bit errors using parity — without incurring performance penalties — and use the replicas within the cache to correct them. Further, it is conceivable that with replicas provided by our mechanism, one could possibly achieve even higher reliability than ECC in certain error situations.

The rest of this paper is organized as follows. The next section gives an overview of the dead block prediction strategy used here. The design space that is explored is given in Section 3. Section 4 gives the experimental setup. The results from simulations are presented in Section 5. Finally, Section 6 summarizes the contributions of this paper and identifies directions for future research.

## 2 Evicting Blocks Not in Use

When we are trying to replicate a block, we would like to put these replicas in the lines that contain data which are not likely to be used in the near future. This would help us alleviate any performance penalties, in terms of miss rates, from such replication. We refer to such lines as dead blocks.

Similar problems have been explored in the context of prefetching [14, 7], and cache leakage control [10]. The specific mechanism used in this work is similar to that proposed by Kaxiras et al [10], wherein a two-bit saturating counter is associated with each cache line. This counter is incremented at a timer tick and when it saturates, the block is declared to be dead. Whenever there is an access to this block in the meantime, the counter is reset. Note that we need only 2 bits per cache block (this works out to 0.39% space overhead for a 64 byte line size) which is quite small (and its power consumption has also been shown to be quite low [10]).

## 3 Replication Design Space

### 3.1 Important Questions

**How aggressively should we mark blocks to be dead?** The dead block prediction mechanism described in the previous section provides a straightforward approach to make space for replicas. A very high counter frequency is more reliability-biased, i.e., it is going to clear more space for replicas, and a low counter frequency is more

performance-biased, i.e., it will try to hold on to blocks hoping for better temporal locality exploitation.

**When do we replicate?** In this work, we use two ways by which replicas can be created: (i) replicating data at the time it is brought from L2 (or memory) to the L1 if there is a suitable dead block at the replication site; and (ii) replicating the data at the time of a write to the block in L1 if possible, and updating both the original and the replicas.

Based on these two mechanisms, we consider two schemes. The first scheme uses both these mechanisms (replicating data at both L1 misses and at writes), while the second scheme uses only the second mechanism (replicating data at the time of a write). The reason that we include the second mechanism in both these schemes is because it is more important to provide higher reliability for blocks that are dirty (since in a write back dL1 these are not flushed to L2), while in the case an error occurs for a clean block, one could always go deeper in the memory hierarchy to bring the block [12].

The main difference between the two schemes is that the read-only data is not replicated in the second scheme, whereas it can be replicated in the first. Therefore, if an error occurs after the time data is brought from L2 to L1 but before it is written (updated) in L1, the first scheme can provide replica from L1, while the second scheme needs to visit L2 to fetch the data. Since error occurrence is not a very frequent event, paying L2 access latency when these rare events occur can be acceptable. On the other hand, the second scheme may be able to replicate a higher percentage of modified data (as only modified data are replicated). Therefore, one can expect better reliability and lower power consumption with the second scheme. Note that, in both the schemes, if an error occurs for a modified data without replica (and without ECC), recovery may not be possible. However, such cases are expected to be less frequent in the second scheme where a higher percentage of modified blocks would be replicated.

**Where do we replicate?** In this paper, we evaluate a class of schemes called “distance- $k$ ”, where parameter  $k$  is a non-negative integer, to search for dead blocks and to find replicas when the time comes. For example, assuming that original data is stored in set  $m$ , the distance-10 scheme places the replica in set  $(m+10) \bmod N$ , where  $N$  is the total number of sets in the cache. More specifically, the set  $(m+10) \bmod N$  is first checked if the replica can be placed there (discussed later). If it can, the replica is stored there (writing back the victim block if it is dirty). Otherwise, a fallback strategy (discussed later) is used. In our evaluation, we performed experiments with two values of parameter  $k$ :  $N/2$  and 0. In the rest of this paper, these two replication schemes are referred to as “vertical replication” (replication across sets) and “horizontal replication” (replication within the ways of a set), respectively. It should be noted that a bit needs to be associated with each block to indicate whether it is a replica or a primary copy. This is because the lookup for another block is that set may match the tags of the replica.

**How aggressively should we replicate?** As discussed above, in cases where we find that set  $(m+10) \bmod N$  does not contain a suitable block to evict, we resort to a fallback strategy. Maybe the simplest fallback strategy is “do nothing”, where we do not create a replica in dL1 in such cases. It is also possible to look for another place to put the replica. One such strategy is called “power-2” in this paper, and works as follows. We first try distance- $k$ , where  $k = 2^i$ . If this try is unsuccessful, we try distance- $2^{(i+1)}$  or distance- $2^{(i-1)}$ , depending on which direction we want to proceed towards. If these are also unsuccessful, we next try

distance- $2^{(i+2)}$  or distance- $2^{(i-2)}$ , and so on. We can stop trying after a certain number of attempts have been made.

**How many replicas do we need?** For higher reliability, one may want to create more replicas of a given block. An important question is then where to create these replicas. One option would be to follow the power-2 scheme summarized above, and replicate the block at each set visited; that is, the difference here is that we have multiple copies of the same block.

**How do we protect unreplicated cache blocks?** We consider two options — just maintaining a parity bit at byte granularity, and maintaining an 8-bit SEC-DED at 64-bit granularity (called ECC) — in our experiments. Note that the consequence of the first option is that when an error occurs to a dirty block that is not replicated, we cannot recover back the data (if we do not have a write-through cache). On the other hand, the second option can detect and correct such (single bit) errors.

**How do we protect replicated cache blocks?** As in the previous question, we could again use the same two options for protecting replicated blocks (i.e., both the primary and the replicas). However, since replicas automatically enhance reliability, and multi-bit errors are rare, we considered only the option of using parity to protect replicated blocks. Note that we could even have the option of dropping parity and simply use NMR techniques [20, 23] to detect and recover from errors based on the replicas. However, at least parity would be needed to protect non-replicated blocks, and this space should be allocated in any case for each cache line. Hence, we consider only the parity option.

**How do we place a primary copy in a set?** In the case of primary copies, we simply use the normal LRU mechanism to pick a victim regardless of whether it is a dead, replica or another primary block. In this regard, we are not any different from a normal cache placement.

**How do we place a replica in a set?** Here there are several options to find a victim, with the common characteristic being that they do not evict primary copies that are not dead (so that performance is not significantly affected). There are the following options:

- *dead-only*: We can perform LRU only amongst the dead blocks to find a victim. This approach tries to give reliability higher importance in not selecting candidates from replicas.
- *replica-only*: We can perform LRU only amongst the replicas to find a victim, giving performance more importance.
- *dead-first*: We can consider blocks that are dead or replicas for replacement, except that we check the dead blocks first.
- *replica-first*: We can consider blocks that are replicas or dead for replacement, except that we check the replicas first.

Of these, we do not feel that replica-only is very meaningful as it not only tries to eliminate the replicas, but it has also little scope for creating replicas when primary copies fill a set. Of the remaining, we mainly focus on dead-only and dead-first since our ICR mechanism is expected to leverage of the accuracy in dead-block predictions.

**What needs to be done upon a replacement?** When we replace a primary copy, we can either remove the replica or leave it there (for possible performance benefits). In the former case, an L1 load/store miss would directly go to L2. In the latter case, the L1 load/store miss could search for a replica either before going to L2, or could be done in parallel with L2 lookup. If we evict a replica, then we can simply throw it away.

## 3.2 Schemes Under Consideration

In this work, we consider the following schemes:

- **BaseP**: This is a normal dL1 cache without dead block prediction, which does not perform any replication. All lines are protected only by parity, and load/stores are modeled to take 1 cycle in our simulations.
- **BaseECC**: This is similar to BaseP, except that all lines are protected by ECC. Stores still take 1 cycle (as the writes can be buffered), but loads take 2 cycles to account for the ECC verification. Note that in one set of later experiments, we consider speculative loads for BaseECC which complete in one cycle, i.e., ECC checks are done in the background.
- **ICR-P-PS (LS)**: This implements our replication mechanism with dead block prediction that is performed at both dL1 misses and dL1 writes. Parity is used for protecting both replicated and non-replicated lines, and to detect errors (single bit errors at 8 bit granularity). On the occurrence of such errors, in the case of non-replicated blocks, the block is loaded from L2 if it is not dirty, and the error is unrecoverable if the block is dirty. If the parity indicates an error for a replicated block, the parity of the replica is checked. If this also has an error (though this probability is much smaller), we default to the actions taken for an error in the non-replicated case. When an error does not occur, the load only needs to access the primary copy and incurs 1 cycle (since it is only parity computation). If an error occurs, and the replica is needed, we need an extra cycle, making the load take 2 cycles. Writes are always 1 cycle since they are buffered.
- **ICR-P-PS (S)**: This is similar to the previous scheme except that duplication is performed only at writes.
- **ICR-P-PP (LS)**: The difference between this and the ICR-P-PS (LS) scheme is that the replicas are searched in parallel and are both compared before the load returns. We conservatively assume this to take 2 cycles for loads, and stores taking 1 cycle as usual.
- **ICR-P-PP (S)**: This is similar to the previous scheme, except that replications are performed only on writes.
- **ICR-ECC-PS (LS), ICR-ECC-PS (S), ICR-ECC-PP (LS), ICR-ECC-PP (S)**: These schemes are similar to the corresponding ones described above, except that ECC is used to protect the non-replicated lines, thus enhancing their reliability. One could ask why not use the ECC for the replicated lines as well, since the space would anyway be available for those lines in these schemes. However, our goal here is performance savings for the loads to such lines since parity comparisons would be much more efficient (and are modeled as 1 cycle versus the 2 cycles needed for ECC checks). This is the case for ICR-ECC-PS (LS) and ICR-ECC-PS (S) where the replica is not looked up until parity fails. On the other hand, ICR-ECC-PP (LS) and ICR-ECC-PP

Configuration Parameter	Value
<b>Processor</b>	
Functional Units	4 integer ALUs 1 integer multiplier/divider 4 FP ALUs 1 FP multiplier/divider
LSQ Size	8 Instructions
RUU Size	16 Instructions
Issue Width	4 instructions/cycle
Cycle Time	1ns
<b>Cache and Memory Hierarchy</b>	
L1 Instruction Cache	16KB, 1-way, 32 byte blocks 1 cycle latency
L1 Data Cache	16KB, 4-way, 64 byte blocks 1 cycle latency
L2	256K unified, 4-way 64 byte blocks 6 cycle latency
Memory	100 cycle latency
<b>Branch Logic</b>	
Predictor	combined, bimodal 2KB table two-level 1KB table 8 bit history
BTB	512 entry, 4-way
Misprediction Penalty	3 cycles

**Table 1.** Configuration parameters and their values in our base configuration for a superscalar architecture. All caches are write-back.

(S) always look up the replica and involve 2 cycles (and the benefits of these are more in their reliability rather than performance compared to BaseECC).

## 4 Experimental Settings

We have evaluated all the different schemes discussed in Section 3.2 together with the different parameters outlined in the design space. In order to conduct detailed cycle level simulations, we use the SimpleScalar 3.0 [3] infrastructure with its sim-outorder mode to model a multiple issue superscalar processor. The default simulation values that are used in our experiments are given in Table 1. The costs for stores (which are always 1 cycle) and loads to dL1 for each scheme have already been discussed in Section 3.2. We use eight applications from the Spec2000 suite for this evaluation.

### 4.1 Evaluation Metrics

We use different metrics in this study to compare the performance of the schemes and their effectiveness:

- *Execution Cycles* is the time taken for the execution of 500 million application instructions.
- *Replication Ability* is the fraction of times that one is able to replicate a cache line upon a load miss or a store (depending on the scheme used).
- *Loads with Replica* is the fraction of read hits that also find a replica in the cache at the time of the read. A higher fraction indicates higher reliability for the cache data.
- *Energy* is the total dynamic energy incurred because of accesses to dL1 and L2 caches.

## 5 Results

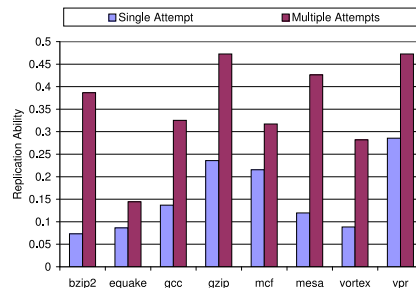
The results in Sections 5.1 and 5.2 use an aggressive dead block prediction strategy, wherein the block is immediately pronounced dead, as soon as the access for that block is complete making it a more promising candidate to hold replicas. We use the dead-only policy to choose the victim, which give higher bias towards reliability and the performance results that we obtain (in Section 5.2) will represent a pessimistic estimate of performance.

We then use the results of the impact of a larger decay window in Section 5.3 to evaluate the performance and reliability trade-offs in Sections 5.4 and 5.5, respectively. Section 5.6 explores the possibility of performance enhancements with choices upon replacement. Sections 5.7 summarizes the results from our sensitivity analysis. Finally, Sections 5.8 and 5.9 compare our schemes with other alternatives for addressing cache reliability and ECC performance problems.

### 5.1 Replication Mechanisms

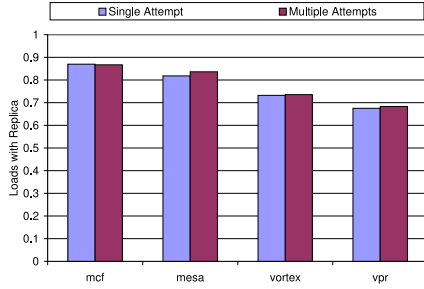
Before we present the results for all the applications and configurations, we first investigate what is the effect of aggressiveness in replicating cache lines.

Figure 1 shows the replication ability for our benchmarks, with a single attempt strategy (Distance-N/2) and another which considers multiple attempts (Distance-N/2 and N/4 in that order). Note that in either case we are attempting to create only one replica. As can be seen, the multiple attempt strategy does allow a higher probability of replicating cache lines. However, when we look at the loads with replica (Figure 2), we find negligible improvement from multiple attempts at replicating. This is because the loads are mainly to the data lines which were replicated even with the single attempt. In general, in these results we find the overall replication ability relatively low since we are using the dead-only strategy for finding candidate positions, and after a point the number of such positions may become less with high replication rates. Despite this fact, the loads with replicas are much higher and this is what really matters as far as reliability is concerned.



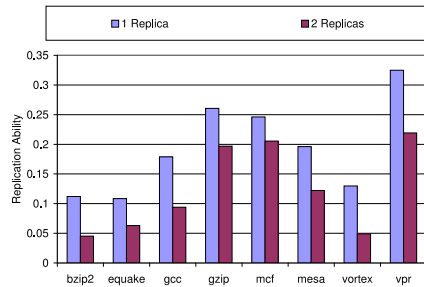
**Figure 1.** Replication ability for single and multiple attempt cases for ICR-P-PS (S). In either case, we are trying to create a single replica for the primary block. Multiple attempt case tries Distance-N/2 and Distance-N/4.

The replication ability in creating multiple copies with the attempts being done on writes alone is shown in Figure 3. We show the ability to create just one replica (the alternate location(s) were not available for replication) as well as the ability to successfully create two replicas (i.e., three copies of a block exist in the cache). We see that we are able to create more than one copy around 12% of the



**Figure 2.** Loads with replica for single and multiple attempt cases for ICR-P-PS (S). In either case, we are trying to create a single replica for the primary block. Multiple attempt case tries Distance-N/2 and Distance-N/4. Only the benchmarks that exhibit some difference are shown.

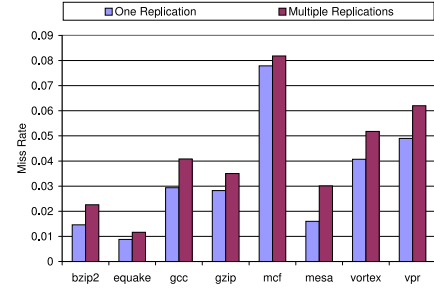
time (averaged across the applications). Several copies can provide higher degrees of reliability with schemes such as NMR being possible. However, the space taken by these multiple copies can evict more useful blocks thereby worsening the locality and increasing miss rates as is shown in Figure 4. It is not clear whether the added reliability warrants this deterioration in miss rates (in some cases such as mesa, the miss rate nearly doubles).



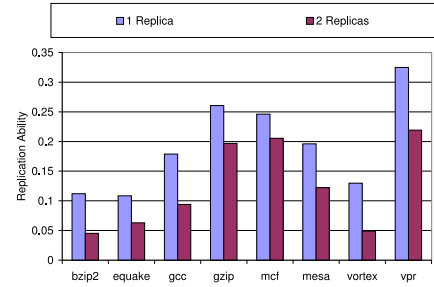
**Figure 3.** Replication ability for single replica and two replicas for ICR-P-PS (S). For the single replica case, we use Distance-N/2. For the multiple replica case, the first replica tries Distance-N/2, and the second one tries Distance-N/4.

We next examine how the Distance-N/2 and Distance-0 replication mechanisms affect the behavior in Figure 5. We can see that there is little difference between these schemes for our default cache configuration. A reason for this behavior is that the sets in the cache are more or less evenly balanced in terms of the live/dead data lines that they hold. We have also conducted experiments with Distance-7 (a prime number) and the results were not any different from those obtained in the Distance-N/2 case.

All these results give us directions on setting the replication mechanism for the next set of experiments. We set the number of replicas to at most one and make only a single attempt to replicate. This is also favorable from the hardware viewpoint in simplifying cache lookup, and requires only 1 extra bit per cache line to indicate whether it is a primary copy or a replica. This can also be more power efficient than having more replicas since each lookup expends dynamic power. We have also fixed the alternate location choice as



**Figure 4.** Miss rates when single replica and multiple (two) replicas are created ICR-P-PS (S).

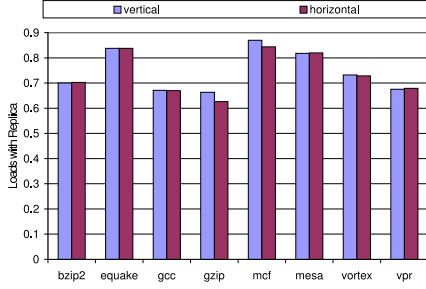


Distance-N/2 as our experiments with other distance values did not make any difference.

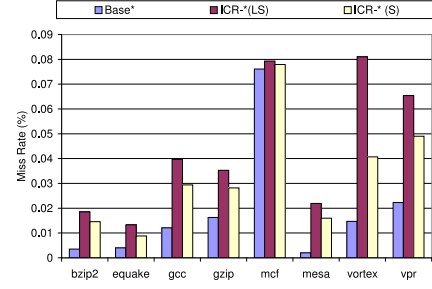
## 5.2 Comparing Performance — Aggressive Dead Block Prediction

Using the above settings for replication, we next compare the performance implications of the ten schemes described in Section 3.2. In these results, when the primary copy is evicted, all the replicas are evicted as well. Alternatives for these decisions will be explored later in the paper. Further, we are aggressively predicting a block to be dead as soon as its access completes, and the performance cycles are thus expected to be much worse than is possible with a more relaxed dead block prediction.

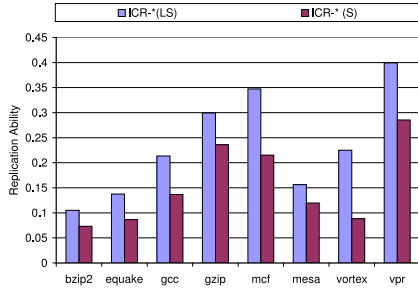
We begin by looking at the replication ability of our proposed ICR schemes based on the two opportunities when the replication can be attempted — upon a store alone (S), or at both stores and load misses (LS) — in Figure 6. As can be seen, LS allows higher replication abilities than S alone. The effect of the replication ability on subsequent load hits is shown in Figure 7. These results show very good promise for the ICR strategies since over 65% of read hits also find replicas in the cache across all applications. When we attempt to create replicas at both opportunities (LS), we find over 90% of read hits finding replicas (almost resulting in complete duplication in mcf). The results in Figures 6 and 7 show that even if opportunities for replication may not be very high, the chances of finding a replica when needed may be extremely good. This is a consequence of the *locality* of the program, wherein a few data items (that are also getting replicated because of this reason) are in high demand, i.e., hot data items are getting automatically replicated (we do not need a separate cache for achieving this compared to that needed by [11]). These results suggest that this approach can be very useful in enhancing the integrity of data. We next move on to evaluating the performance



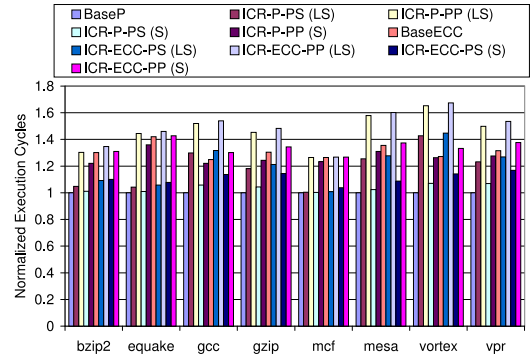
**Figure 5.** Loads with replica for vertical and horizontal replications for ICR-P-PS (S). The vertical replication uses Distance-N/2, while the horizontal replication uses Distance-0.



**Figure 8.** Miss rates for Base\*, ICR-\*(LS), and ICR-\*(S) schemes. Both ICR-\*(LS) and ICR-\*(S) increase the number of dLI misses.



**Figure 6.** Replication ability for ICS-\*(LS) and ICS-\*(S) schemes. The results show that ICS-\*(LS) replicates more data than ICS-\*(S).

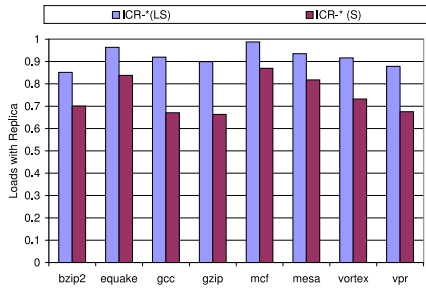


**Figure 9.** Normalized execution cycles for all schemes. Compared to the increases in the number of misses, the increase in execution cycles is not excessive in some of our schemes. This is because superscalar execution hides many cache misses. The average increases over BaseP due to ICR-P-PS(S) and ICR-ECC-PS(S) are 3.6% and 21.0%, respectively.

consequences of such enhancements.

We give the miss rates for the normal cache (BaseP or BaseECC) and compare them to the deterioration in miss rates due to replication with the S and LS strategies in Figure 8. We can see that while in applications like mcf, there is not a significant difference in miss rates (in this application, there is anyway very poor locality that evicting existing blocks to make way for replicas does not hurt performance), in some others there is a significant percentage change.

We next look at the performance impact of the replication strategies in Figure 9. All the bars are normalized for each application with respect to the bar for BaseP, which provides the best performance (no deterioration in miss



**Figure 7.** Loads with replica for ICS-\*(LS) and ICS-\*(S) schemes.

rates, and both loads/stores take 1 cycle). BaseECC, on the other hand, does not deteriorate miss rates, but incurs 2 cycles for each load hit, leading to approximately 30% degradation in performance on the average. The ICR-\*-PP schemes are comparable to ECC in general because it takes two cycles for each load hit to a block with a replica. This suggests that more than miss rate deterioration, the cost of the hit is much more important to optimize from the performance viewpoint. Even in ICR-P-PP, since most of the loads are to blocks with replicas (refer to Figure 7), the performance penalty of 2 cycle load latencies are more significant, making its behavior comparable to BaseECC or ICR-ECC-PP. The BaseECC and all ICR-\*-PP schemes are around 25-45% worse in terms of performance compared to BaseP.

The performance cycles of the ICR-\*-PS schemes do not directly obey the deterioration of miss rates shown earlier. This is because of the savings in load latencies for accesses to replicated blocks (needing only parity calculations and not ECC) which become more important than the miss rates. Of these schemes, the ICR-\*-PS (S) schemes fare better because of their lower probability of evicting useful data from the cache. Within this space, we find ICR-P-PS (S) doing slightly better than ICR-ECC-PS (S) as it is to be expected,

since it does not incur 2 cycle latencies for loads to non-replicated data. But, this difference is less than 6% on the average across applications.

The two most attractive schemes from these results are ICR-P-PS (S) and ICR-ECC-PS (S). The former is only 3.6% worse than BaseP on the average across applications, while providing better reliability for replicated data which as pointed out constitute a large portion of the data in active use. The latter, while 21% worse than BaseP, is better than BaseECC by 15% performance-wise, while still protecting unduplicated data by ECC and providing higher reliability than BaseP for replicated data. *It should be emphasized that we obtain such good results even with a very aggressive dead block prediction (with dead-only replacement) strategy.* It is important to understand the performance and reliability ramifications of a more lenient dead block prediction strategy which is investigated in the rest of this paper. We also change from the dead-only to dead-first to give more options to find replica sites while not deteriorating miss rates. In the interest of space and clarity, we mainly focus on ICR-P-PS(S) and ICR-ECC-PS(S) in the rest of the paper, comparing them to BaseP and BaseECC.

### 5.3 Aggressiveness in Predicting Dead Blocks

We vary the number of cycles after an access is made to predict a block to be dead and the results are shown in terms of the replication ability and loads with replica in Figure 10 for vpr. The corresponding normalized execution cycles taken by ICR-P-PS (S) and ICR-ECC-PS (S) are given in Figure 11. We can see that while the replication ability reduces with an increasing decay window size as is to be expected, the corresponding effect on the loads with replicas is negligible. This is because the program locality is good enough that only a few replicas (of the hot data items) are needed, and these are anyway available in the cache. On the other hand, dead block prediction aggressiveness can affect the miss rate of the program and have an effect on the execution cycles as is shown in Figure 11. Considering these trade-offs, we use a 1000 cycle decay window for the rest of the experiments in this paper.

### 5.4 Comparing Performance — Relaxed Dead Block Prediction

The execution cycles of the schemes with a 1000 cycle decay window is given in Figure 12. In fact, with this set of parameters, ICR-P-PS (S) and ICR-ECC-PS (S) are only 2.4% and 10.1% away, respectively, from the best performing BaseP on the average, while providing much better reliability. Further, the performance of ICR-ECC-PS (S) turns out to be 16.8% better than that of BaseECC. Figure 13 shows that the loads with replicas for a 1000 cycle dead block decay window are not significantly different from that by setting the window to 0 cycles (even if there are some differences in replication ability), suggesting that the reliability behavior is not going to be significantly compromised. However, an investigation of the reliability behavior with error conditions is needed as is undertaken in the next section.

### 5.5 Comparing the Schemes with Error Injection

We have considered several transient error models for the cache (direct, adjacent, column and random) described in [11]. Since the overall results are similar, we present the results specifically for the *random injection model*. In this model, an error is injected in a random bit of a random word present in the dL1 cache. Such errors are injected at each

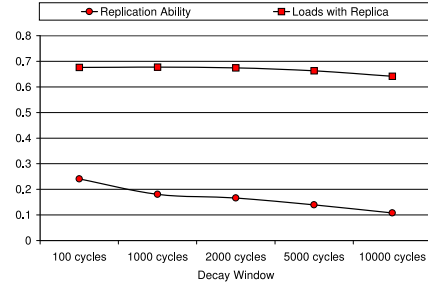


Figure 10. Replication ability and loads with replica for ICR-P-PS (S) with different decay window sizes (vpr).

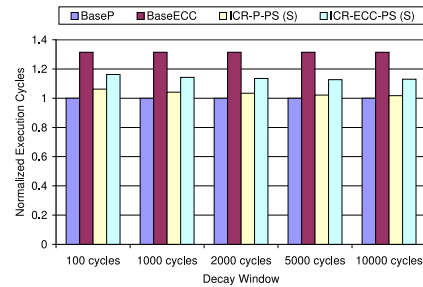


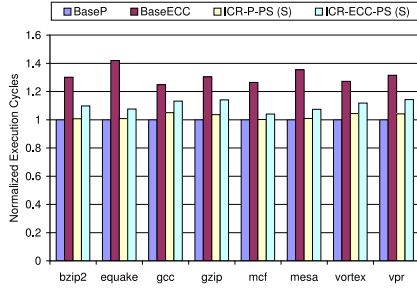
Figure 11. Normalized execution cycles with different decay window sizes (vpr). The increase in execution cycles due to ICR-P-PS (S) over BaseP is less than 4% for 1000 cycle window size and around 1.7% for 10000 cycle window size.

clock cycle based on a constant probability that is varied in these experiments.

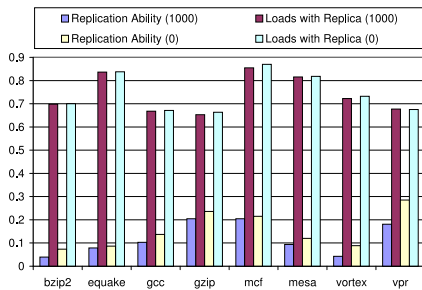
Figure 14 shows the fraction of loads that could not recover from a single-bit error when the data is loaded as a function of the probability of an error occurring in each cycle for BaseP, ICR-P-PS (S) and ICR-ECC-PS (S) in vortex. BaseECC can detect and correct all such 1-bit errors. We find that even at such high (and unrealistic) error rates, the ICR schemes exhibit much better error resilient behavior compared to BaseP. Note that the intention here is to mainly show the benefits of ICR under intense error behavior, and hence very high error rates are depicted. In a more realistic situation, error rates would be much lower (in fact, for 1/100000, the error rates even for BaseP tend to become zero for all of our applications). This reiterates the importance of improving performance in the more normal case of error-free operation.

### 5.6 Performance Improvements

So far we have been focusing on enhancing reliability without degrading performance. However, our replication strategy can be adapted to enhance performance in certain cases. For instance, when there is a need for replacing a primary copy, it is possible to leave the replica in dL1, and simply replace the primary copy in isolation. This can help fill certain primary copy misses at a much lower cost than going to L2 (and can thus make the cache appear to have higher associativity sometimes [18]). In such cases, upon



**Figure 12.** Normalized execution cycles with a decay window size of 1000 cycles. The average increases in execution cycles (over BaseP) due to BaseECC, ICR-P-PS (S), and ICR-ECC-PS (S) are 30.9%, 2.4%, and 10.2%, respectively.



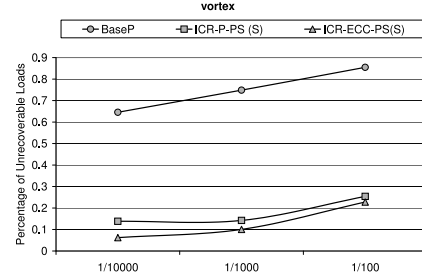
**Figure 13.** Replication ability and loads with decay window sizes of 1000 cycles and 0 cycles.

a primary copy miss in ICR-\*<sub>-PS</sub> schemes, there is only one extra cycle of load latency if the replica is present in dL1. This is much better than going to L2 (assumed to be 6 cycles in our experiments) that will be incurred in BaseP and BaseECC.

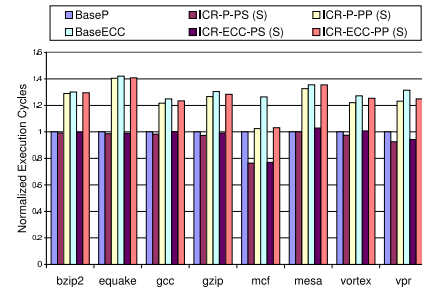
Figure 15 shows the performance results by leaving replicas in dL1 upon primary copy eviction for the ICR-\*<sub>-PS</sub> (S) schemes. We see that ICR-P-PS (S) and ICR-ECC-PS (S) are providing as good performance as BaseP (and much better than BaseECC) in nearly all cases, while providing much better reliability behavior as explained in the previous section. In fact, one could opt to even protect unreplicated blocks with ECC — which incurs higher load overheads — and still reap these benefits since most of the loads are to replicated blocks. We find that in mcf and vpr (and to a smaller extent in gcc, gzip and vortex), both these schemes outperform even BaseP (as much as 24% in one case), while providing better reliability characteristics. These results clearly demonstrate the benefits of in-cache replication for enhancing cache reliability without compromising on performance, and perhaps even enhancing performance in certain cases.

### 5.7 Sensitivity to Cache Parameters

We also conducted experiments with other cache sizes and associativities, and observed the overall results/trends to be similar. More specifically, the replication ability increases with increasing cache size since there are more replication sites available. However, the increase in the loads with replicas is not that significant. The other way of viewing this is that even in a small cache, we are replicating



**Figure 14.** Percentage of unrecoverable loads for vortex.



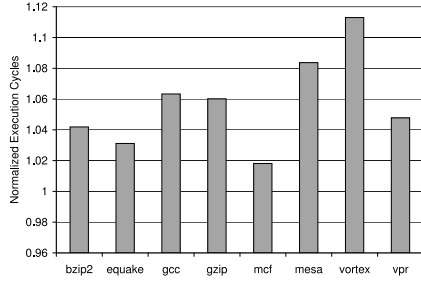
**Figure 15.** Normalized execution cycles when replicas are used for improving performance.

the data that is really the most in demand, and thereby providing higher reliability in the common case. This behavior is also observed when one keeps the cache size the same and varies the associativity. These results are not detailed here in the interest of space.

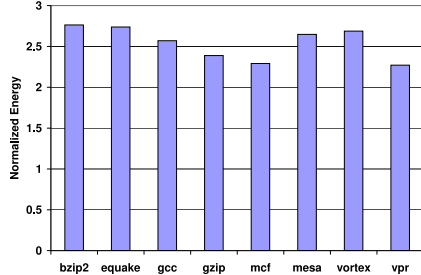
### 5.8 Comparison with Write-Through Data Cache

In addition (or in conjunction) to ECC, another way of enhancing the ability to recover data from errors, is to force data L1 writes to propagate to L2 as well. When a subsequent L1 read indicates an error (using parity), the value can be compared with that in L2 to possibly recover from the error(s). Write-through L1 is thus another approach to enhance L1 data integrity as is the case in IBM POWER4 [2]. However, this has both performance and power ramifications. In order to reduce stalls on writes, a write-buffer is typically introduced between L1 and L2 [24]. Stalls are still encountered when the write-buffer fills up. Figure 16(a) illustrates the effect of these stalls on the overall execution cycles by giving the execution cycles of BaseP with a write-through data L1 (using a coalescing write-buffer of 8 entries), normalized with respect to our ICR-P-PS (S) scheme with a write-back L1. We find that our solution is around 5.7% better in terms of performance, on the average, across the applications.

A write-through data L1 also increases the dynamic energy consumption by increasing the number of writes to L2. Figure 16(b) compares the energy expended in the L1 and L2 caches for BaseP (with write-through), normalized with respect to ICR-P-PS(S) (using write-back). The per access dynamic energy figures have been obtained using the CACTI 3.0 tool [4]. We find that despite the higher L1 power incurred by ICR-P-PS(S) due to the duplicate writes



(a) Normalized execution cycles.



(b) Normalized energy.

**Figure 16.** Performance and energy consumption (of L1 and L2 caches) with write-through dL1 for BaseP normalized with respect to ICR-P-PS(S) that uses a write-back dL1.

performed when there are replicas, the overall energy expended in the data cache hierarchy (L1 plus L2) is still less than half of that for BaseP with write-through dL1.

## 5.9 Comparison with BaseECC with Speculative Loads

One way of circumventing the ECC time cost is to perform the checks in the background while the data is loaded and the computation proceeds speculatively. If an error is detected, the computation can then be squashed. It is to be noted that this does make the hardware more complex and may not be very suitable for embedded systems. Such speculative load capabilities are available today in certain high-end processors. Even if a processor supports speculative loads, while the ECC computation may be performed in the background to not get in the critical path, it does not affect the power consumption of the ECC mechanism itself. As noted earlier, ECC computations can be much more power consuming than simple parity calculations [1]. Figure 17(a) gives the execution cycles of BaseECC (with 1-cycle latency speculative loads), normalized with respect to the performance-optimized ICR-P-PS(S) version that leaves replicas in place. One can observe that, except mcf, ICR-P-PS(S) is still 2.5% better on the average than BaseECC with speculative loads. In mcf, our scheme is 30.8% better than the speculative BaseECC scheme.

Figures 17(b) and (c) show the energy consumption of the speculative BaseECC mechanism (with 1 cycle load latency) in the cache hierarchy (L1 plus L2), normalized to the energy consumption (L1 plus L2) of the ICR-P-PS(S)

scheme for different values of energy consumption taken by parity and ECC computations. These values are given as a percentage of the cost taken by a normal L1 access. As can be seen from Figure 17(b), with a conservative assumption of ECC taking only twice the energy of a parity computation [1], we find ICR-P-PS(S) cache energy is more or less comparable to BaseECC (the difference is around 0.2% on the average across the applications) despite incurring overheads of performing two writes for replicated lines. On the other hand, if ECC computation turns out to be three times more costly than the parity computation from the energy perspective, the average cache energy increase of the speculative BaseECC mechanism over ICR-P-PS(S) is around 3.1% across these applications (see Figure 17(c)). It should be noted that though we have considered different energy costs for parity and ECC computations we are only giving representative results here in the interest of space.

## 6 Concluding Remarks and Future Work

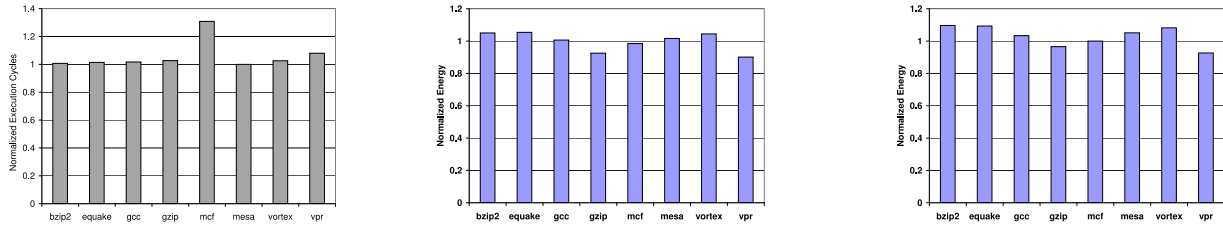
Transient errors in processor caches can lead to catastrophic consequences. Most high-end processors provide SEC-DED capabilities to protect cache data. However, this can delay cache accesses, which can stretch application execution times. Speculative loads to hide this cost can complicate processor design (making them less suitable for embedded systems) and does not still reduce the power expended in ECC computations. While parity based protection is fairly performance efficient, it fails to correct errors which can be as important as detecting them. This paper has presented several interesting design alternatives between these extremes to trade-off performance and reliability criteria. The basic idea is to leverage from the fact that not all blocks in the dL1 cache are really needed in the near future (they are dead), and one can reuse this space to keep replicas of the blocks that are in greater demand. Detection of dead blocks is achieved using simple counters, and this mechanism is already being used to enhance performance and power [10, 7, 14]. With this idea, we have evaluated several replication mechanisms and reliability enhancing schemes and have found two schemes — ICR-P-PS (S) and ICR-ECC-PS (S) — that can prove to be quite useful.

ICR-P-PS (S) is a much better alternative for situations where one may want simple parity protection (BaseP). It provides almost as good performance as parity-based protection, but can enhance reliability significantly by providing replicas for blocks that are in great demand. This scheme is much more performance-efficient than an ECC based protection of all cache words.

ICR-ECC-PS (S) can be a better alternative for situations where one may want ECC protection (BaseECC). It provides much better performance than BaseECC, without significantly compromising on reliability (and can in fact detect/correct multi-bit errors in certain situations). It has much better reliability behavior than a simple parity based protection.

ICR-P-PS (S) and ICR-ECC-PS (S) can even outperform the BaseP performance-optimized version in many cases if we can leave the replicas that they create in the cache when their primary counterparts are evicted. Thus, in-cache replication can serve the dual purpose of not only enhancing reliability, but also to improve performance.

There are several options for addressing the problems studied in this paper. First, one could think of provisioning a cache with both parity and ECC at the same time, and use the parity for loads (incurring lower latency) and using ECC when errors are detected by the parity. However, this doubles the space needed to store such auxiliary information (affecting dynamic and leakage power) for error detec-



(a) Norm. execution cycles (b) Norm. energy(Parity:ECC=15:30%) (c) Norm. energy(Parity:ECC=10:30%)

**Figure 17.** Performance and energy consumption (of L1 and L2 caches) for BaseECC assuming speculative (1-cycle) loads normalized with respect to ICR-P-PS(S), where replicas are left in place when primary copies are evicted. Representative results for Parity and ECC computations taking (b) 15% and 30% and (c) 10% and 30% respectively of the L1 access energy are given.

tion/correction, while our schemes do not require additional storage. Secondly, one could opt for a write-through dL1 as in the IBM POWER4 to ensure writes are not lost because of errors. We have shown that our mechanisms are both performance and power efficient than this alternative. Finally, one could use BaseECC with speculative loads to hide ECC calculation latencies. However, this still does not address the power consumption of ECC calculations, and we have shown that our techniques are better in this respect. With performance, power, space and reliability all becoming important optimization criteria (and often inter-dependent), we believe that the schemes proposed in this paper provide interesting options for cache design from all these perspectives.

In our future work, we plan to explore controlling replication using software mechanisms that can direct how many replicas are needed for each line, when such replication should be initiated, and what blocks should not be replicated.

## References

- [1] D. Bertozzi, L. Benini, and G. De Micheli. Low power error resilient encoding for on-chip data buses. *Proceedings of Design, Automation, and Test in Europe Conference*, Paris, France, March, 2002.
- [2] D. C. Bossen, J. M. Tendler, and K. Reick. POWER4 system design for high reliability. *IEEE Micro*, 22(2):16–24, March 2002.
- [3] D. C. Burger and T. M. Austin. The SimpleScalar tool-set, Version 2.0, *Technical Report 1342*, Dept. of Computer Science, UW, June, 1997.
- [4] CACTI 3.0. <http://research.compaq.com/wrl/people/jouppi/CACTI.html>
- [5] M. Hamada and E. Fujiwara. A class of error control codes for byte organized memory systems-SbEC-(Sb+S)EDcodes. *IEEE Transactions on Computers*, 46(1):105–110, January 1997.
- [6] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, CA, 1996.
- [7] Z. Hu, S. Kaxiras, and M. Martonosi. Timekeeping in the memory system: predicting and optimizing memory behavior. *Proceedings of the 29th International Symposium on Computer Architecture*, May 2002.
- [8] H. Imai. *Essentials of Error-Control Coding Techniques*. Academic Press, San Diego, CA, 1990.
- [9] J. Karlsson, P. Ledan, P. Dahlgren, and R. Johansson. Using heavy-ion radiation to validate fault handling mechanisms. *IEEE Micro*, 14(1):8–23, February 1994.
- [10] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. *Proceedings of the 28th International Symposium on Computer Architecture*, June 2001.
- [11] S. Kim and A. Somani. Area efficient architectures for information integrity checking in cache memories. *Proceedings of International Symposium on Computer Architecture*, May 1999, pp. 246–256.
- [12] S. Kim and A. K. Somani. An adaptive write error detection technique in on-chip caches of multi-level caching systems. *Journal of Microprocessors and Microsystems*, 22(9):561–570, March 1999.
- [13] S. Kim and A. K. Somani. An affordable transient fault tolerance for super-scalar processors. *Fast Abstract of IEEE DSN-2001*, July 2001.
- [14] A-C. Lai, C. Fide, and B. Falsafi. Dead-block prediction and dead-block correlating prefetchers. *Proceedings of International Symposium on Computer Architecture*, July 2001.
- [15] J. M. Mulder, N. T. Quach, and M. J. Flynn. An area model for on-chip memories and its application. *IEEE J. Solid-State Circuits*, 26:98–106, February 1991.
- [16] S. Park and B. Bose. Burst asymmetric/unidirectional error correcting/detecting codes. *Proceedings of International Symposium on Fault-Tolerant Computing*, pages 273–280, June 1990.
- [17] D. Patterson, T. Anderson, N. Cardwell, R. Formm, K. Keeton, K. Kozyrakis, R. Thomas, and K. Yelick. Intelligent RAM (IRAM): chips that remember and compute. *Proceedings of International Symposium on Solid-State Circuits*, pages 224–225, February 1997.
- [18] J-K. Peir, Y. Lee, and W.W. Hsu. Capturing dynamic memory reference behavior with adaptive cache topology. *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 240–250, 1998.
- [19] J. C. Pickel and J. T. B. Jr. Cosmic ray induced error in MOS memory cells. *IEEE Transactions on Nuclear Science*, NS-25:1166–1171, December 1978.
- [20] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package*. Kluwer Academic Publishers, 1996.
- [21] A. M. Saleh. Reliability of scrubbing recovery-techniques for memory systems. *IEEE Trans. Reliability*, 30(1):114–122, April 1990.
- [22] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic. *Proceedings of the International Conference on Dependable Systems and Networks*, June, 2002.
- [23] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems: Design and Evaluation*. Digital Press, Bedford, MA, 1992.
- [24] K. Skadron and D. W. Clark. Design issues and tradeoffs for write buffers. *Proceedings of the Third International Symposium on High-Performance Computer Architecture*, pp. 144–55, February 1997.
- [25] A. K. Somani and K. Trivedi. A cache error propagation model. *Proceedings of International Symposium on Pacific Rim Fault Tolerant Computing*, pages 15–21, December 1997.
- [26] J. Sosnowski. Transient fault tolerance in digital systems. *IEEE Micro*, 14(1):24–35, February 1994.
- [27] P. Sweazey. SRAM organization, control, and speed, and their effect on cache memory design. *Proceedings of Midcon/87*, pages 434–437, September 1987. <http://www.specbench.org>.
- [28] K.-L. Wu, W. K. Fuchs, and J. H. Patel. Error recovery in shared memory multiprocessors using private caches. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, Number 2, April 1990, pp. 231–240.