Soft Error Benchmarking for L2 Cache with PARMA

Jinho Suh, Murali Annavaram, Michel Dubois

Soft Error Benchmarking for L2 Cache with PARMA

Jinho Suh, Murali Annavaram, and Michel Dubois
Ming Hsieh Department of Electrical Engineering
University of Southern California, Los Angeles, CA 90089-2560
{jinhosuh, annavara}@usc.edu, dubois@paris.usc.edu

Abstract

Low voltage operation and small device sizes reduce the critical charge stored in a SRAM cell making caches more susceptible to soft errors. To counter the problem, a plethora of protection schemes are being proposed to reduce the power and area overheads of cache protection. Accurate methods to measure and compare the effectiveness of such schemes are sorely needed.

This paper introduces a unified reliability benchmarking framework with PARMA (Precise Analytical Reliability Model for Architecture). PARMA is a rigorous analytical framework to measure the failure rate in the presence of soft errors under any protection scheme. PARMA continuously and accurately accounts for the increase of failure density due to soft errors affecting program behavior during an execution. We have implemented the PARMA framework on top of a cycle-accurate simulator to benchmark Level-2 cache failure rates for a set of CPU 2000 benchmarks. Three protection schemes are compared: parity, word-level 1*-bit ECC and block-level* 1*-bit ECC.*

1. Introduction

As on-die feature sizes continue to shrink the era of multi-billion transistors per chip has arrived. Roughly 50% of chip real estate is occupied by L2 or L3 caches. To reduce static power dissipation, caches are operated at low voltages using techniques such as drowsy caches and sub-threshold voltage operation.

Techniques to save power have the undesirable consequence of reducing reliability. Lower supply voltage during drowsy operation, for instance, reduces the amount of charge held in an SRAM cell thereby making it highly susceptible to single event upsets (SEUs) causing soft errors. For example, using Hazucha's [7] and Roche's [18] model, reducing Vdd from 1V to 0.8V in the 65nm high-performance model from ITRS2007 [21] could cause the SER (Single Error Rate) to increase from 1150 to 1472 upsets in 10^9 hours. The impact of SEUs is even more pronounced when chips are used in space electronic systems where cosmic ray intensity is much higher. For instance, the failure rate due to the higher cosmic ray flux increases by three orders of magnitude at 18km as compared to sea level [25]. In space, under the GEO solar flare model, the failure rate rises by 10 orders of magnitude as compared to sea-level [2]. Under extremely low operating voltages, erratic behavior causes the failure rate to increase by nine orders of magnitude [5].

1.1. Error Protection Approaches

Chip designers have been using mechanisms such as parity and error correcting codes (ECC) to guard against soft errors. As the soft error rate continues to rise designers are resorting to extreme solutions such as protecting each word in a cache line with per-word (or word-level) ECC. 1-bit ECC can fully recover from a single bit error but can only detect two bit errors. In general errors can be classified into three categories: Recoverable Errors (RE), Detected Unrecoverable Errors (DUE), and Silent Data Corruption errors (SDC). Recoverable errors do not impact system behavior and are ignored in this paper.

Not every SEU in an SRAM cell translates into an error. The impact of an SEU can be masked due to a variety of reasons such as: the cache line is invalid, the corrupted cache line is overwritten before it is read, the cache line is empty, or the block in the cache line is not referenced again. We adopt the usual classification of SDC, TRUE DUE and FALSE DUE [24] in this paper. A fault becomes an error at the architectural level as soon as the faulty instruction or the instruction loading the faulty data is *committed* by the processor. We do not track down the fault to the I/O level of program outcome. In the presence of error detection code, a DUE is raised when a corrupted block is delivered from L2 cache to L1 cache. If a word from a block containing corrupted bit(s) has been "consumed" (instruction fetch or data load by the committed instruction) by the processor, the DUE is TRUE, otherwise it is FALSE. If the delivery of the corrupted block has not been detected because detection code is not present, or because the number of corrupted bits is beyond the capability of detection

code, an SDC error is counted when a processor "consumes" the word containing any of the corrupted bit(s), otherwise the event is ignored.

1.2. Challenges in Measuring Reliability

Challenges with Existing Metrics. From classical reliability theory, one can typically measure the reliability using three metrics:

(i) The probability of survival (called *survivability*). This probability (which is a function of time *t*) shows the chance of an object to survive up to and including time *t.*

(ii) The Mean Time To Failure (MTTF), which is obtained by calculating the first moment (expected value) of the *unconditional* failure density distribution. "Unconditional" means irrespective of the fact that the system survived up to time *t*.

(iii) The failure rate, obtained by counting failures within a time period. The failure rate is obtained by integrating the *conditional* instantaneous failure rate over the test time. FIT is a special case of failure rate where the test time is 10^9 hours. "Conditional" means that the system has survived until time *t*.

In the special case where the instantaneous failure rate is constant over time, as in the Poisson model which is commonly assumed due to its simplicity, all three metrics become interchangeable. However, in general cases where instantaneous failure rate is time varying, the above three metrics are *not* interchangeable. MTTF is unlikely to be precisely calculated unless someone can test until the whole population fails. Yet, survivability or failure rate measurements can still be meaningful, although limited by being valid only within the test time.

Researchers studying the impact of SEUs assume that soft errors resulting from SEUs are Poisson. While we model SEUs to follow Poisson, the soft errors may not follow Poisson, as previously noted in [6][11]. Some specific reasons include:

(i) accesses to bits are determined by program behavior or microarchitectural status (i.e. frequency of accesses is time varying),

(ii) faults are masked due to timing, logical, architectural, and microarchitectural factor and a corrupted bit may become "invisible", and

(iii) error protection codes, which can correct or detect errors in specific cases, change the constant Poisson rate into a time-varying rate.

Under these conditions, measuring the reliability impact due to soft errors using MTTF becomes challenging.

Challenges due to Test Time Explosion. One common approach used to model and estimate the vulnerability of various components, such as processors, memory subsystems or interconnects, is fault injection. In this approach faults are injected into the detailed RTL model or simulator of the system statistically to observe the vulnerability of the system [10][17]. While this framework is conceptually simple and can model any type of fault, the biggest drawback is that it requires vast numbers of experiments to obtain statistically meaningful results. For soft errors, given the current SEU rate in the order of 10^{-25} per bit, would require an unthinkable number of simulation cycles to observe a single fault in a simulation run. Actually no current benchmark is long enough to register one SEU during one of its execution. The number of runs of the benchmark to obtain a statistically meaningful estimate would be astronomical. Even if faults injections are accelerated by orders upon orders of magnitude the procedure still requires massive amounts of test time and only produces statistical estimates. Furthermore the artificial acceleration of faults distorts the results of the test, and accurately accounting for these distortions is a daunting problem.

The solution: PARMA. To meet these challenges, we have developed PARMA (Precise Analytical Reliability Model for Architecture), a new framework for accurate and realistic evaluation of the impact of soft errors. First, PARMA is accurate because it takes care of the effects of program behaviour, architectural masking and error protection codes, which all affect the Poisson assumption as mentioned above. Second, PARMA avoids the impossibly long test times and the distortions caused by accelerated injection rates by measuring reliability in a single simulation run with the actual SEU rate. It calculates on the fly the accurate probability of failure by modelling SEUs as a binomial process, as the simulation run progresses.

The rest of this paper is organized as follows. Section 2 provides a brief overview of the related work. Section 3 describes the PARMA framework in detail and applies it to L2 caches to calculate the FIT rates of various error types under various error protection schemes. Section 4 then describes our implementation of the framework on SimpleScalar. Results are analyzed in Section 5. We conclude in Section 6.

2. Related Work

Recognizing the drawbacks of fault injection test times, researchers have proposed mechanisms to

accelerate reliability estimation. Two approaches are of particular interest: SoftArch [12] and AVF [13].

SoftArch is a reliability estimation tool, which obtains the MTTF of a microarchitectural structure. SoftArch calculates the average TTF in one benchmark run (called MTTF of finite program), by computing the probability distribution of each event on the object of interest. It then uses the notion of infinite program to calculate meaningful MTTF by extrapolating MTTF of finite program. This extrapolation is akin to Epstein's Maximum Likelihood Estimator of MTTF with censorship [4]. SoftArch simply assumes a Poisson arrival process for SEUs and calculates the probability of a soft error probability based on the length of time a bit is vulnerable. It is unclear how SoftArch accounts for complex failure states. For instance, the same bit hit twice is treated as two soft errors, instead of treating that case as no error. Temporal Multi-Bit Errors, where two SEUs flip two different bits within a byte or a word are treated as two different errors; obviously two bit flips in a single byte or word lead to a single system failure. The same weaknesses are true of AVF-based approaches as well.

Another weakness of SoftArch is that as described in [12] SoftArch requires completing one run of the benchmark before computing MTTF. Hence, SoftArch is an off-line tool and cannot be used by techniques that dynamically alter protection based on instantaneous reliability measure. PARMA in contrast, integrates instantaneous failure rates to measure reliability during test and on-line. PARMA can measure reliability even with time-varying rates due, for example, to environmental conditions. PARMA works even if the infinite program assumption does not hold, provided we do not need to extrapolate the PARMA's failure rate into FIT, but simply need to evaluate the failure rate of one finite-length benchmark.

The Architectural Vulnerability Factor (AVF) framework is widely accepted due to its simplicity. During program execution, AVF classifies a bit as either (micro)architecturally relevant or irrelevant. If an SEU strike on a bit propagates to architecturally visible state the bit is deemed architecturally relevant otherwise it is irrelevant. AVF then counts the duration (in number of clocks) of exposure to SEUs of the architecturally relevant bits. Studies in [1], [6] and [13] focus on how to generate the AVF of various components with different approaches by counting the ACE bits or bytes (required for Architecturally Correct Execution) in a cycle and on how to accumulate these counts over the workload execution. Studies in [1] and [23] developed a model based on AVF to calculate error rates directly. AVF methodology assumes that soft errors, not just SEUs, are Poisson distributed. However, prior studies showed that reliability under architectural masking varies with time [6][11]. As we will show in Section 3, accurately accounting for soft errors requires modelling them by a binomial process, which calculates the probabilities for any number of faulty bits. When a binomial process is approximated by a Poisson process, the mean (first moment) of probability distribution is the only correct moment. The probability distribution of a specific number of faulty bits calculated for a binomial process cannot be exactly modelled by a Poisson process. Therefore, AVF based methodology cannot calculate exact probability distribution of having errors in the presence of error detection/correction code.

By contrast, PARMA calculates the exact probability distribution of any number of faulty bits due to temporal MBEs, and hence can accurately measure reliability in the presence of error detection/correction code. Although PARMA is not the first methodology that can handle temporal MBEs (see [14][20]), to our best knowledge, PARMA is the first *unified* framework introduced to exactly calculate failure rate in any possible error state under any error protection code, except for spatial MBEs.

3. The PARMA Model

In every cycle, we measure the probability of any component fault in a system, assuming that no more than one fault of the same component happens in the same cycle. When a fault cannot be corrected or is not masked, we call it an *event* (failure). In a set of bits protected by any detection or correction code, a faulty bit inside the set may or may not be an event if it can be corrected or is architecturally masked. Take an example for a cache block. Bit faults in the block turn into an SDC error event if the error cannot be detected, and if an instruction fetched from the block or if the data that returns from the block by Load is committed by the processor; they turn into a TRUE/FALSE DUE event if 1) the bit faults are detected and 2) any/no instruction or data Load value containing the bit faults from the block is committed by the processor. Otherwise the bit faults are a non-event.

Let's index each processor cycle by j (where $1 \leq j \leq T_{exe}$). Then $h(j)$, the discrete time failure probability mass at j^{th} cycle is defined as:

$$h(j) = \Pr(\text{fail at } j \mid \text{survived until } j) \quad (1)$$

Then the total failure rate observed during the benchmark execution time T_{exe} is:

$$H(T_{exe}) = \sum_{j=1}^{T_{exe}} h(j) = E[ERR] \quad (2)$$

$H(T_{exe})$ is the failure rate of the target benchmark observed during execution time T_{exe}. It is also equivalent to $E[ERR]$, the expected number of failures of type *ERR* (*ERR* can be SDC, TRUE DUE or FALSE DUE) within this timeframe. If we assume that the target benchmark repeats its execution indefinitely as in SoftArch, we can extrapolate the observed failure rate to the more familiar FIT rate, simply by scaling time. Then, the average FIT rate for a set of applications running one after another, independently on a processor is calculated as:

$$\overline{FIT_{ERR}} = \sum_{\forall \text{ benchmark } i} f_i \times FIT_{i,ERR} \quad (3)$$

where f_i is the fraction of time taken by the execution of benchmark i. $FIT_{i,ERR}$ is the FIT rate extrapolated from $E[ERR]$ per T_{exe}.

3.1. Physical Model

The system considered in this paper is shown in Figure 1. The L2 cache in a dotted frame is the only component vulnerable to soft errors. All other system components are assumed to be totally fault-free or bullet-proof. An L2 cache block ceases to be vulnerable when it is written back to main memory.

3.2. SEU Model

PARMA uses one bit of data and one clock cycle as the minimum units of space and time. The probability that one bit can be upset in one cycle is denoted p. We make two basic assumptions about the physical fault model.

(i) All clock cycles are independent: whether a bit is struck or not at clock cycle i is independent of whether or not it was hit in cycles k =1,...,i-1 and

(ii) All cache bits are independent: there is no correlation between SEUs affecting any two cache bits.

The first assumption allows us to consider SEUs as a renewal process, so that the probability distribution of SEUs is always the same after every cycle. The second assumption may not hold if one particle strike may cause multiple bit upsets. The current PARMA model can be refined to include spatial MBEs provided their probability of occurrence can be established. We will include spatial MBEs in future implementations of PARMA.

3.3. Vulnerability Clock

In between two consecutive accesses to a memory block in the L2 cache, the block is vulnerable to SEUs. Memory is byte-addressable and therefore the model must track the integrity of a unit of memory no less than a byte. To this effect, in PARMA, each byte of memory is associated with a *vulnerability clock*

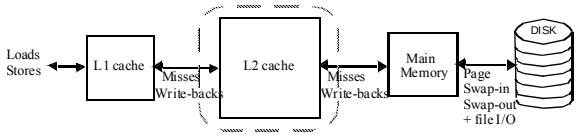

Figure 1. Memory Hierarchy Model

which is initialized to zero and which keeps track of the time the byte resides in L2 and is subject to SEUs.

As soon as a block of data is brought from memory into L2 all the bytes in this block become vulnerable and the vulnerability clock of each byte ticks up in every clock cycle, while the block resides in the L2 cache. When a block is loaded in L1 two copies of the same block coexist, one in L1 and one in L2. The copy in L1 is bullet-proof to any strike, but the copy in L2 is still vulnerable. Thus we must keep track of two cases: (i) the block copy in L1 is modified, in which case it will be written back and the vulnerability clocks of every byte of the L2 copy will be reset to their value at the time it was loaded in L1, or (ii) the block copy in L1 is not modified, in which case the block will not be written back to L2. In this latter case, the block copy in L2 remains vulnerable during the entire stay of the block in L1 and the vulnerability clocks of its bytes will not be reset when the block is replaced in L1.

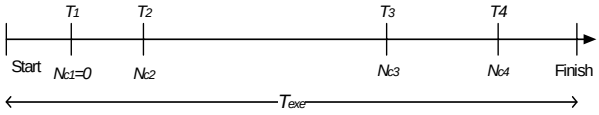

Figure 2. L2 Block Accesses during Execution

When a block is replaced in L2, it is either replaced silently, i.e. the block is not written back to main memory since the block is clean, or it is written back to main memory when the block is dirty. If it is replaced silently, then the vulnerability clocks for all the bytes in the block remain unchanged in the main memory as if they had never been loaded in L2 before. If it is written back to main memory because it was modified, then each byte in the block carries its corresponding vulnerability clock with it but the clocks stop ticking and will restart later on when the block is reloaded in the L2 cache on a miss.

Rather than a naïve implementation that calculates the faults on each byte in every clock cycle, PARMA updates vulnerability clocks and the fault rates only when the block is touched in L2, on an L1 miss or an L1/L2 block is written back to L2/memory. We timestamp each byte when the block is accessed.

Figure 2 shows the timeline for the execution of a benchmark, from Start to Finish where we track a byte in a memory block. In this figure the T_i's are the times in the execution (in processor cycles) when the block is accessed in L2 and the N_{ci}'s are the readings of the vulnerability clock (in processor cycles) for a given

byte in the block at these times. The block is loaded for the first time in L2 at time T_1. The vulnerability clock of the byte is reset. Then the block is accessed three more times. In between two accesses the block may move in and out of the L2 cache and, at times, the vulnerability clock of the byte stops, keeps ticking up, or is reset. After T_4 the block is dead and will be replaced or will simply sit in cache until the end of the execution.

Resetting the vulnerability clocks is key for PARMA to calculate discrete failure probability masses given by (2). Note that discrete failure probability mass (which is a dual of continuous time instantaneous failure rate) is a conditional probability for a failure at this moment, given that a system has survived up to this moment. PARMA resets the vulnerability clock in order to represent that the block survived up to that moment, satisfying the 'given' condition of (1).

We update fault metrics every time a cache line is accessed. At these times, we calculate the conditional probabilities of various error types, called discrete failure probability mass, based on the vulnerability clock in each byte of the block. The first step towards that goal is to calculate the distribution of bit faults in a set of bytes.

3.4. Distribution of Faults in a Set of Bytes

Whenever the vulnerability clock cycles (N_c) is reset, any probability as a function of N_c automatically becomes a conditional probability failure mass.

We first calculate the Probability of i SEUs on one cache bit in N_c vulnerability clock cycles.

$$P_i(N_c) = \binom{N_c}{i} p^i (1-p)^{N_c - i}, i = 0,...,N_c \quad (4)$$

This result is based on the fact that within any cycle, one SEU can cause a bit to flip, with probability p. The value of p depends on the technology, geological location of experiment, etc. A bit can flip i times up to N_c times in N_c vulnerability cycles, and a bit flip may happen in any one set of i cycles amongst the N_c vulnerability cycles. With this equation we can calculate the Probability q that a given cache bit will be faulty after N_c vulnerability cycles, knowing that a bit is faulty if it is flipped an odd number of times during N_c cycles.

$$q(N_c) = \sum_{i=0}^{\lfloor N_c/2 \rfloor} P_{2i+1}(N_c) \quad (5)$$

We can now apply similar formulas spatially to all bits in a byte as we just did temporally for each bit. Given the value of $q(N_c)$ we calculate the probabilities $q_b(k)$ that a byte residing in an L2 cache line will experience k bit errors in N_c vulnerability cycles, where $k = 0, ..., 8$.

$$q_b(k) = \binom{8}{k} q(N_c)^k (1 - q(N_c))^{8-k}, k = 0,...,8 \quad (6)$$

$q_b(k)$ is an implicit function of N_c. We call a set of bytes that are accessed together an *access domain* (denoted as D), and a set of bytes that are protected together by the same code under any specific protection scheme a *protection domain* (denoted as S). Access domain and protection domain can be the same or one can be a subset of the other. For example the cache block is the access domain when the L2 cache writes back to main memory since the transfer happens at a block level. But the protection domain is a word if a word-level SECDED code is used; within a word one faulty bit will be corrected and two faulty bits will be detected. The protection domain can be a simple word, a block, or a set of bytes that should be accessed together as in interleaved ECC [14]. In the following we assume that the access domain is an L2 cache block and the protection domain is a subset of the access domain.

We can now calculate the Probability distribution of k bit faults ($k = 0, ..., 8N_S$) in a protection domain S of N_S bytes. In order to calculate the Probability of k faulty bits in a protection domain, we index each byte by $j = 1, ..., N_S$. For simplicity, we designate a byte in S by index j and use the notation $\{j\} \in S$ to sweep all bytes in domain S. A vulnerability clocks value N_c is associated with each byte and is implicit in the computation of $q_{b,j}(k)$. This is the reason why we must consider each byte in the protection domain separately. The Probability that there are k faulty bits within the protection domain can be expressed as a function of all $q_{b,j}(k)$'s as follows:

$${}^{S}Q_m(k) = \sum_{(\sum l_j = k)} \prod_{\{j\} \in S} q_{b,j}(l_j) \quad (7)$$

where m is an optional index of a protection domain in the access domain. In (7), we choose all the cases that S has k faulty bits among N_S bytes and compose the Probability accordingly. The summation in this equation is also known as k-fold discrete convolution. m can be omitted in the notation if the access domain holds only one protection domain. The above equation restricts the computation of Probability distribution to a protection domain. We do not need to calculate Probability distributions at the granularity of an access domain, as error types such as DUE or SDC are calculated for each protection domain. Because errors in different protection domains are independent of each other, they can simply be added.

Now, let's look at the L1 cache. Even if a corrupted L2 cache block reaches the first level cache, the computation may still be correct. Indeed, if no loaded data or I-Fetch of a committed instruction returns a faulty value during the sojourn of the block

Table 1. Errors as a Function of k and N_S Shown in this Study

	No parity	1-bit Parity		1-bit ECC			
	SDC	TRUE DUE	SDC	TRUE DUE		SDC	
				word-level	block-level	word-level	block-level
Access Domain	Block	Block	Block	Blk containing M words	Block	Blk containing M words	Block
Protection Domain	N/A	Block	Block	Word	Block	Word	Block
Faulty bits	≥1 in C	∀odd in S, ≥1 in C	∀even in S, ≥1 in C	2 in any M S_ms, ≥1 in any C_m	2 in S, ≥1 in C	≥3 in any M S_ms, ≥1 in any C_m	≥3 in S, ≥1 in C
Notation	${}^{B}E^{NP,SDC}$	${}^{B}E^{P1b,TRUE/DUE}$	${}^{B}E^{P1b,SDC}$	${}^{W}E^{ECCw,TRUE/DUE}$	${}^{B}E^{ECCb,TRUE/DUE}$	${}^{W}E^{ECCw,SDC}$	${}^{B}E^{ECCb,SDC}$

in L1, then an L2 cache block failure becomes no error, or a FALSE DUE. Every time the loaded data or I-Fetch of a committed instruction returns a faulty value from the block residing in L1 for the first time (we do not count the same error more than once), it is considered either a SDC error or a TRUE DUE. If the processor has stored a value first in an L1 byte before it is read then that byte is error-free. In the PARMA model, we use two bits, a *first_read* bit and a *dirty* bit per byte in an L1 cache block to accumulate the expected number of TRUE errors accurately. A Store sets the dirty bit to one and leaves the *first_read* bit untouched. A Load sets the *first_read* bit only if the dirty bit is not set. If the *first_read* bit is unset then either the processor never read that byte or a Store first modified the byte before any Load. In essence, the *first_read* bit indicates if the processor read a vulnerable byte at least once.

Let's call the set of consumed (I-Fetch or operand Load) bytes in the L1 copy of an access domain (or L2 block as assumed above) as C. Then the complement set $\overline{C}$ contains *unconsumed* bytes. $\overline{C}$ is the set of bytes (with $8\,N_{\overline{C}}$ bits) whose *first_read* bit is not set when the L1 copy of an access domain is evicted. The union of C and $\overline{C}$ is an (L1 copy of an) access domain. The Probability of having k faulty bits *only* in $\overline{C}$, but having no faulty bit in C is obtained as:

$$ {}^{C}Q(0)\cdot{}^{\overline{C}}Q(k) = \prod_{\{i\}\in C} q_{b,i}(0)\cdot \sum_{(\sum l_j = k)} \prod_{\{j\}\in \overline{C}} q_{b,j}(l_j) \qquad (8) $$

$$ {}^{C}Q_m(0)\cdot{}^{\overline{C}_m}Q_m(k) = \prod_{\{i\}\in C} q_{b,i}(0)\cdot \sum_{(\sum l_j = k)} \prod_{\{j\}\in \overline{C}_m} q_{b,j}(l_j) \qquad (9) $$

(9) is used instead of (8) when protection domain is smaller than access domain. C and $\overline{C}$ are divided into subsets of C_m and $\overline{C}_m$ at the boundary of protection domains. As before, m is the optional index indicating the protection domain. Note that $\overline{C}$ is not determined until all the block in an access domain are replaced in L1, or until the end of the execution, whichever comes first due to the accesses to L1. (8) (or (9)) is used to calculate SDC and TRUE DUEs in the following subsections.

In this paper we consider four schemes: no error detection or correction, SED (Single Error Detection, or 1b-parity), SECDED (Single Error Correction Double Error Detection, or 1b-ECC) on each block and SECDED on each word. Table 1 shows SDC, TRUE DUE and/or FALSE DUE that are classified differently in different protection schemes. Composition of domains and the number of faulty bits necessary to diagnose DUE or SDC are shown in Table 1 with different schemes. For example, word-level 1-bit ECC has SDCs when (i) three or more faulty bits are in the protection domains (words) when L2 block is accessed, and (ii) at least one bit among those faulty bits are consumed by committed instructions.

In the following subsections, we show how to calculate SDCs and DUEs using equations (7) and (8).

3.5. SDC Errors

As shown in Table 1, the number of SDCs depends on the protection scheme in L2. For example, L2 block is copied to L1 cache with at least one faulty bit. Then, SDC happens when any of the faulty bit(s) in the copy of the L1 block becomes part of the consumed and committed set of bytes. The Probability calculation of SDC is equivalent to subtracting the Probability of having faulty bit(s) (that are not corrected or detected) in $\overline{C}$, from the Probability of having faulty bit(s) (that are not corrected or detected) in the entire block. Thus, SDCs under no-protection (NP) are calculated as follows:

$$ {}^{B}E^{NP,SDC} = \sum_{k=1}^{8N_B} {}^{B}Q(k) - \sum_{i=1}^{8N_{\overline{C}}} {}^{C}Q(0)\cdot{}^{\overline{C}}Q(i) \qquad (10) $$

Similarly, with single error detection per block (denoted as P1b), SDCs are calculated as:

$$ {}^{B}E^{P1b,SDC} = \sum_{\forall even\, k>0}^{8N_B} {}^{B}Q(k) - \sum_{\forall even\, i>0}^{8N_{\overline{C}}} {}^{C}Q(0)\cdot^{\overline{C}}Q(i) \quad (11) $$

Consider now the case of an L2 cache with 1-bit ECC per block (denoted as ECCb). The expected numbers of SDCs is:

$$ {}^{B}E^{ECCb,SDC} = \sum_{k=3}^{8N_B} {}^{B}Q(k) - \sum_{i=3}^{8N_{\overline{C}}} {}^{C}Q(0)\cdot^{\overline{C}}Q(i) \quad (12) $$

Note that the protection domain in P1b and ECCb is the same as the access domain, i.e. a block.

Finally, consider the case of an L2 cache with 1-bit ECC per word (denoted as ECCw). Let N_W be the number of bytes in a word and M be the number of words in a block. Because one faulty bit in every $8N_W$ bits can be corrected, up to M faulty bits can be corrected in a block. Whenever a block is accessed, we need to include all M words. Let's index each word in the block by m, where m = 1, ..., M. The expected number of SDCs with word-level 1-bit ECC (denoted as ECCw) is:

$$ \begin{aligned} {}^{B}E^{ECCw,SDC} &= \sum_{m=1}^{M} {}^{W}E_m^{ECCw,SDC} \\ &= \sum_{m=1}^{M} \left\{ \sum_{k=3}^{8N_W} {}^{W}Q_m(k) - \sum_{i=3}^{8N_{\overline{C_m}}} {}^{C}Q_m(0)\cdot^{\overline{C_m}}Q_m(i) \right\} \end{aligned} \quad (13) $$

Whenever an L2 cache block is filled in, is accessed due to L1 miss or an L1 block is evicted (discarded or written back to L2), (10)-(13) are invoked to yield the expected number of faults since the last access to the block. These probabilities are accumulated to calculate the expected number of SDC errors until simulation ends. At the end of the simulation the SDC failure rates for each scheme is obtained, and if necessary it is extrapolated to FIT rates under the assumption that the program execution is repeated indefinitely.

3.6. DUE Errors

Unlike SDC errors, DUEs are further classified as TRUE DUEs and FALSE DUEs. As shown in Table 1, the number of DUEs also depends on the protection scheme in L2. DUEs are classified differently in different protection schemes. DUEs are generally raised as soon as they are detected – when L2 block is fetched if L2 block is protected by detection code. Just as in SDCs, PARMA counts it as a TRUE DUE if at least one faulty bit has been consumed and committed by the processor after DUE has been raised. The TRUE DUEs are obtained in the same way as SDCs: By subtracting the Probability of detecting the faulty bits that are only within the $\overline{C}$, from the total Probability of detecting faulty bits in all the protection domains in an access domain. The computation of FALSE DUEs is also automatic because it is just the Probability of DUEs due to faulty bits in $\overline{C}$ only. There is no DUE under NP. With P1b, DUEs are calculated as:

$$ {}^{B}E^{P1b,FALSE/DUE} = \sum_{\forall odd\, i}^{8N_{\overline{C}}} {}^{C}Q(0)\cdot^{\overline{C}}Q(i) \quad (14) $$

$$ {}^{B}E^{P1b,TRUE/DUE} = \sum_{\forall odd\, k}^{8N_B} {}^{B}Q(k) - \sum_{\forall odd\, i}^{8N_{\overline{C}}} {}^{C}Q(0)\cdot^{\overline{C}}Q(i) \quad (15) $$

With ECCb, the expected numbers of DUEs is:

$$ {}^{B}E^{ECCb,FALSE/DUE} = {}^{C}Q(0)\cdot^{\overline{C}}Q(2) \quad (16) $$

$$ {}^{B}E^{ECCb,TRUE/DUE} = {}^{B}Q(2) - {}^{C}Q(0)\cdot^{\overline{C}}Q(2) \quad (17) $$

Finally, with ECCw, the expected number of DUEs is:

$$ \begin{aligned} {}^{B}E^{ECCw,FALSE/DUE} &= \sum_{m=1}^{M} {}^{W}E_m^{ECCw,FALSE/DUE} \\ &= \sum_{m=1}^{M} {}^{C}Q_m(0)\cdot^{\overline{C_m}}Q_m(2) \end{aligned} \quad (18) $$

$$ \begin{aligned} {}^{B}E^{ECCw,TRUE/DUE} &= \sum_{m=1}^{M} {}^{W}E_m^{ECCw,TRUE/DUE} \\ &= \sum_{m=1}^{M} \left\{ {}^{W}Q_m(2) - {}^{C}Q_m(0)\cdot^{\overline{C_m}}Q_m(2) \right\} \end{aligned} \quad (19) $$

4. Simulation

We have integrated PARMA with the SimpleScalar/Alpha simulator [3] to measure FITs for various benchmarks. We first specify the parameters we have used in our simulations.

4.1. Parameters in PARMA Simulations

The PARMA model starts with a probability p representing the probability that any one bit is flipped within one clock period. In this paper, we use the ITRS 2007 [21] predicted value of Poisson rate λ, which is 1,150 SEUs in 10^9 hours for one megabits of SRAM built in the High Performance CMOS technology. We use it directly to calculate p. The target processor frequency is 3GHz, and the value of p is 1.0155×10^{-25} from the Poisson probability mass function.

Table 2. Cache Hierarchy Specification

Cache	Size	Associativity	Latency [cyc]
IL1	16KB	1-way	2
DL1	16KB	4-way	3
UL2	256KB	8-way	NP/P1b: 7 ECCw (4B):10 ECCb (64B): 11

In our simulations we took into account the increased latency caused by decoding the check bits. This adds extra latency to a block access, when the

decoding is in the critical path. 1-bit parity causes little performance, power and area penalty, but ECC does. Both space (number of check bits) and time (decoding delays) overheads are incurred for SECDED (Single Error Correction Double Error Detection) ECC. Naseer et al [15] reported the decoder delays for various protection codes in 90nm technology. We extrapolated the reported 1-bit ECC decoder delay from [15] using Matlab and scaled it to 65nm technology. In short, we assume seven check bits with 0.827ns decoder delay for 4-byte word-level SECDED code and 11 check bits with 1.376ns decoder delay for 64-byte block-level SECDED, resulting in different L2 cache latencies for different protection schemes in Table 2.

We chose nine benchmarks from the SPEC2K suite with different memory footprints as measured by [19]. The list of benchmarks is shown in Table 3. All benchmarks were compiled for the alpha ISA. We ran 1-billion committed instructions after selecting one SimPoint [16] for each benchmark.

The target processor built on 65nm technology is a 4-wide out-of-order processor with 64-entry ROB, 32-entry LSQ, and McFarling's hybrid branch predictor. The processor is running at 3GHz with 150 cycles latency to off-chip main memory. The L1 caches (I and D) have 32-byte block and the unified L2 cache has 64-byte block. All the caches are non-blocking, write-back caches. All the cache parameters shown in Table 2 are obtained from Cacti 5 [22]. The delay of fetching and decoding ECC check bits is included in the L2 cache access latencies.

4.2. Integration of PARMA in SimpleScalar

In order to measure the program-specific vulnerability of the L2 cache, we ran each benchmark sample for one billion instructions while collecting the expected number of failures. Whenever a memory block resides in the L2 cache, the vulnerability clocks associated with all the bytes in the block tick up. SimpleScalar/Alpha [3] is augmented with a binary search table to keep track of the entire memory footprint of a program. The simulator keeps track of all memory accesses at the byte level to update the vulnerability clocks precisely.

Because we concentrate on the vulnerability of the last level (L2) cache, major events that contribute to the overall vulnerability happen rarely and the simulation must execute very long traces in general. In the following section, we show the simulation results and assess the usefulness of the protection schemes based on the observed vulnerabilities.

PARMA needs to track all the blocks in the memory footprint to correctly maintain the vulnerability clock cycles. Each byte in any L2 block requires holding time-stamps (unsigned long integer) to count vulnerability clock cycles when the block is accessed. Also, each byte requires holding several bits of metadata such as, whether the byte has been updated in L1 or read by the processor. Therefore, the memory overhead of PARMA is about 35 times larger than the total simulated memory footprint, after including the space to hold the memory addresses used to index in a large Binary Search Tree (BST) holding all the memory data.

The computation overhead for calculating the Probability of having k faulty bits in the access domain from (7) is $O(n^2)$ when n is the total bits in the access domain. Therefore, calculating the entire Probability distribution of having $k = 0, \ldots, 8N_S$ faulty bits (that is, n different probabilities) gives computation overhead of $O(n^3)$. As a result, SimpleScalar with PARMA runs 25 times slower than the basic SimpleScalar.

5. Results

In this section we first show the failure rate measurement results obtained for the execution samples of one billion instructions each. In order to provide more intuitive numbers, we extrapolate measured failure rates into FITs. The main purpose for these experiments is to demonstrate the PARMA framework, especially its precise calculation of the FIT rates and its applicability to any protection scheme.

5.1. Simulation Results

Table 3 shows the FIT rates measured in our simulations. Average FITs are derived from (3). During the execution of each sample of one billion instructions, more than 90% of blocks are touched more than four times in most of the benchmarks except in parser (65.9%) and vpr (81.9%). All the cache lines are filled in a 256KB L2 cache after the one billion committed instructions in all the benchmarks.

Without any error protection the SDC FIT (first column in Table 3) is still very high, varying from 72 to 910, which is not acceptable to most manufacturers today. With 1-bit parity, SDC FITs are effectively turned into TRUE DUE FITs. The SDC FIT with 1-bit parity is 16 orders of magnitude smaller than the SDC FIT with no protection. Hence, 1-bit parity is more than sufficient to satisfy the stringent SDC FIT requirements of most manufacturers. However the DUE FIT rate may not be within the reliability budget for an L2 cache. Upgrading to block-level 1-bit ECC

Table 3. FIT in Various Protection Schemes

(a) NP_ SDC: no-protection/SDC
P1b_TRUE_DUE:1bit parity/TRUE DUE
(b) P1b_FALSE_DUE:1bit parity/FALSE DUE
(c) P1b_ SDC: 1bit parity/SDC
(d) ECCb_TRUE_DUE: block-level 1bit ECC/TRUE DUE
(e) ECCb_FALSE_DUE: block-level 1bit ECC/FALSE DUE
(f) ECCb_SDC: block-level 1bit ECC/SDC
(g) ECCw_TRUE_DUE: word-level 1bit ECC /TRUE DUE
(h) ECCw_FALSE_DUE: word-level 1bit ECC /TRUE DUE
(i) ECCw_SDC: word-level 1bit ECC/SDC

Bench	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)
ammp	443.95	152.74	7.23E-14	7.47E-14	6.80E-15	9.64E-29	3.41E-15	1.50E-15	2.43E-31
art	72.68	0.31	5.26E-17	5.27E-17	2.27E-21	3.95E-35	3.19E-18	1.00E-20	1.41E-37
crafty	909.63	360.9	5.99E-14	6.93E-14	1.23E-14	1.29E-29	3.00E-15	1.78E-15	2.84E-32
galgel	158.93	37.75	2.81E-16	2.81E-16	3.50E-17	1.74E-33	1.31E-17	6.10E-18	4.26E-36
gcc	102.86	6.44	1.86E-16	2.09E-16	1.10E-17	1.08E-33	1.12E-17	1.80E-18	3.02E-36
mesa	589.81	47.72	1.29E-14	1.20E-14	4.00E-16	2.64E-31	6.38E-16	9.50E-17	7.17E-34
parser	659.58	254.66	1.38E-13	1.02E-15	6.60E-16	4.72E-33	6.63E-15	3.03E-15	4.72E-31
twolf	377.85	182.27	3.82E-15	3.92E-15	8.80E-16	3.93E-32	1.86E-16	1.03E-16	9.72E-35
vpr	440.90	167.05	5.41E-15	5.45E-15	8.40E-16	7.24E-32	2.48E-16	1.33E-16	1.77E-34
Avg	338.57	116.41	2.25E-14	1.46E-14	1.80E-15	1.08E-29	1.10E-15	5.20E-16	5.88E-32

or word-level 1-bit ECC, the average DUEs drops by a factor of 2.77×10^{16} and 2.81×10^{17} respectively.

Due to different memory footprints, and different memory behaviors in each benchmark program, the FIT rate varies widely among programs, as shown in Table 3. We summarize the various average FIT rates calculated from (3), for various protection schemes and types of errors in Table 3.

A high percentage of DUEs are TRUE, which means that a small number of DUEs detected at the cache level are false-positive. Therefore appropriate error recovery mechanisms from DUEs are worth investing in. In the presence of ECC, the fraction of DUEs that are TRUE is determined by the number of bytes read by the processor, as well as by the number of faulty bits successfully corrected by the ECC code within these bytes.

One distinct advantage of word-level ECC over block-level ECC is that word-level ECC can correct more words and thus yields less TRUEs than block-level ECC does. The numbers show that word-level ECC capable of correcting one faulty bit in every four-byte word is on average about 13 times stronger than block-level ECC capable of correcting one faulty bit in every 64-byte block. However, word-level ECC requires 10% more silicon area in terms of total cache size. Because the DUEs of block-level ECC is already tiny (of the order of 10^{-15}~10^{-17}), reducing the failure rate again by an order of magnitude may not be a critical reliability gain, although using 10% more silicon might be critical. If the reduction of failure rate is the sole purpose of a (stronger) word-level ECC, investing more silicon may not be justified.

The fact that ECC is extremely reliable as compared to any non-correcting scheme implies that partial ECC schemes to reduce power and area constraints while maintaining dependability should be closely scrutinized. For example, take a crude scheme in which half of all cache lines is protected by 1-bit parity and the other half is protected by block-level 1-bit ECC. From Table 3, let's assume that we run gcc and accesses to L2 are uniformly distributed across cache lines, then the SDC would roughly be $0.5\times102.86+0.5\times2.2\times10^{-16}\approx51.43$, which shows that vulnerability is heavily dominated by the part not protected by ECC. Thus, adopting partial error correcting schemes may not enhance dependability much. PARMA is flexible enough to be applicable to any protection scheme so that dependability verification becomes feasible in an accurate way.

6. Conclusion

Reliability has become a great concern, given the uncertainty on how dependable future systems will be as technology evolves and feature size keeps shrinking.

SRAM cells are vulnerable to SEUs but a large amount of SEUs can be corrected or masked. Thus their actual vulnerability is often much less than their intrinsic vulnerability would suggest. Intrinsic FIT rates are thus derated by vulnerability factors. Several approaches to measure the vulnerability derating factor have been used such as injecting faults in circuits, computing an AVF, or developing mathematical expressions under heavy approximations. However, to the best of our

knowledge, no general model was ever proposed before, which is flexible enough to apply to many proposed complex protection schemes while maintaining accuracy.

We have introduced a modelling framework called PARMA relying on benchmark simulations and stochastic fault modelling to evaluate the true impact of soft errors in SRAM structures on the reliability of processor execution and we have applied the framework to a last level cache. With vulnerability clocks associated with each byte (the smallest addressable unit of memory), we show that the model can calculate very fine grained vulnerability in a last level cache. Because the model enumerates all possible faulty states and derives expected failure rates based on that state space, the modelling framework can intuitively and easily be applied to any protection scheme for any SRAM based storage structure. More importantly, the model can provide an accurate quantitative basis to trade-off various design targets such as reliability, power, area and performance.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. CSR-0615428 and CCF-0834798.

References

[1] Asadi, H., Sridharan, V., Tahoori, M. B., and Kaeli, D. 2006. Vulnerability analysis of L2 cache elements to single event upsets. In Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, 3001 Leuven, Belgium, 1276-1281.

[2] Bajura, M.A.; Boulghassoul, Y.; Naseer, R.; DasGupta, S.; Witulski, A.F.; Sondeen, J.; Stansberry, S.D.; Draper, J.; Massengill, L.W.; Damoulakis, J.N., "Models and Algorithmic Limits for an ECC-Based Approach to Hardening Sub-100-nm SRAMs," Nuclear Science, IEEE Transactions on , vol.54, no.4, pp.935-945.

[3] Doug Burger and Todd M. Austin. The SimpleScalar Tool Set Version 2.0. Technical Report 1342, Calculater Sciences Department, University of Wisconsin-Madison.

[4] B. Epstein, "Truncated life tests in the exponential case," Annals of Mathematical Statistics, 25.3.

[5] D. Ernst, N. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. Proceedings of the 36th Annual International Symposium on Microarchitecture, pages 7-18, 3-5.

[6] Xin Fu, Tao Li, and José Fortes, Sim-SODA: A Unified Framework for Architectural Level Software Reliability Analysis, Workshop on Modeling, Benchmarking and Simulation (Held in conjunction with ISCA-33).

[7] Hazucha, P.; Svensson, C., "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," IEEE Transactions on Nuclear Science, vol.47, no.6, pp.2586-2594.

[8] S. H. Hwang and G. S. Choi, "On-Chip Cache Memory Resilience," Proc. of the Intl. Symp. on High-Assurance Systems Engineering, pp. 240-247.

[9] Jeffrey W. Kellington, Ryan McBeth, Pia Sanda, and Ronald N. Kalla. IBM POWER6 processor soft error tolerance analysis using proton irradiation. In Proceedings of the 3rd IEEE Workshop on Silicon Errors in Logic. System Effects (SELSE-3).

[10] Man-Lap Li, Pradeep Ramachandran, Rahmet Ulya Karpuzcu, Siva Kumar Sastry Hari, Sarita Adve. "Accurate Microarchitecture-Level Fault Modeling for Studying Hardware Faults". International Conference on High Performance Calculater Architecture.

[11] Li, X., Adve, S. V., Bose, P., and Rivers, J. A. 2007. Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions. In Proceedings of the 37th Annual IEEE/IFIP international Conference on Dependable Systems and Networks (June 25 - 28, 2007). DSN. IEEE Calculater Society, Washington, DC, 266-275.

[12] Li, X. Adve, S.V. Pradip Bose Rivers, J.A. SoftArch: An Architecture Level Tool for Modeling and Analyzing Soft Errors. In Proceedings of the 2005 international Conference on Dependable Systems and Networks (June 28 - July 01, 2005). DSN. IEEE Calculater Society, Washington, DC, 496-505.

[13] S. S Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, and T.Austin. A systematic methodology to calculate the architectural vulnerability factors for a high-performance microprocessor. Proceedings of the 36th Annual International Symposium on Microarchitecture, pages 29-40.

[14] Mukherjee, S. S., Emer, J., Fossum, T., and Reinhardt, S. K. 2004. Cache Scrubbing in Microprocessors: Myth or Necessity?. In Proceedings of the 10th IEEE Pacific Rim international Symposium on Dependable Computing (Prdc'04) (March 03 - 05, 2004). PRDC. IEEE Calculater Society, Washington, DC, 37-42.

[15] Naseer, R.; Boulghassoul, Y.; Draper, J.; DasGupta, S.; Witulski, A., "Critical Charge Characterization for Soft Error Rate Modeling in 90nm SRAM," IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007, pp.1879-1882, 27-30.

[16] Perelman, E., Hamerly, G., Van Biesbrouck, M., Sherwood, T., and Calder, B. 2003. Using SimPoint for accurate and efficient simulation. In Proceedings of the 2003 ACM SIGMETRICS international Conference on Measurement and Modeling of Calculater Systems (San Diego, CA, USA, June 11 - 14, 2003). SIGMETRICS '03. ACM, New York, NY, 318-319.

[17] M. Rebaudengo, M. S. Reorda, and M. Violante, "An Accurate Analysis of the Effects of Soft Errors in the Instruction and Date Caches of a Pipelined Microprocessor," Proc. of the ACM/IEEE Design, Automation and Test in Europe Conf. and Exhibition (DATE'03), pp. 602-607, Munich, Germany.

[18] Roche, P.; Palau, J.M.; Bruguier, G.; Tavernier, C.; Ecoffet,R.; Gasiot, J., "Determination of key parameters for SEU occurrence using 3-D full cell SRAM simulations," IEEE Transactions on Nuclear Science, vol.46, no.6, pp.1354-1362.

[19] S. Sair and M. Charney, "Memory Behavior of the SPEC2000 Benchmark Suite," IBM Research Report, RC 21852 (98345), 2000.

[20] A.M.Saleh, J.J.Serrano, and J.H.Patel, "Reliability of Scrubbing Recovery Techniques for Memory Systems," IEEE Transactions on Reliability, Vol.39, NO.1.

[21] Semiconductor Industries Association, "International Technology Roadmap for Semiconductors."

[22] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. Cacti 5.1. Technical Report HPL-2008-20, Hewlett-Packard Development Company.

[23] Vilas Sridharan, Hossein Asadi, Mehdi B. Tahoori, David Kaeli, "Reducing Data Cache Susceptibility to Soft Errors," IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 4, pp. 353-364.

[24] Weaver, C., Emer, J., Mukherjee, S. S., and Reinhardt, S. K. 2004. Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor. SIGARCH Comput. Archit. News 32.

[25] J. F. Ziegler and W. A. Lanford, "The Effect of Cosmic Rays on Calculater Memories," Science, Vol. 206, No. 776.