Accurate Model for Application Failure due to Transient Faults in Cache

Mehrtash Manoochehri and Michel Dubois

Computer Engineering Technical Report Number CENG-2014-09

Ming Hsieh Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, California 90089-2562

December 2014

Accurate Model for Application Failure due to Transient Faults in Cache

Mehrtash Manoochehri Michel Dubois
Ming Hsieh Department of Electrical Engineering
University of Southern California
Los Angeles, CA
mmanooch@usc.edu dubois@paris.usc.edu

Abstract

To select an appropriate level of error protection in caches, the impact of various protection schemes on the cache FIT (Failure In Time) rate must be evaluated for a target benchmark suite. However, while many simulation tools exist to evaluate area, power and performance for a set of benchmark programs, there is a dearth of such tools for reliability.

This technical report introduces an accurate and versatile cache reliability model called PARMA+ which estimates a cache's FIT rate in the presence of spatial multi-bit transient faults and for a given benchmark program. PARMA+ is an extension of PARMA [21]. PARMA only dealt with single-bit faults. However, PARMA+ is applicable to single-bit and any spatial multi-bit fault pattern distribution and various error protection schemes for caches including early write-back, bit-interleaving and scrubbing.

We first develop the model formally, then we demonstrate its accuracy. We have run reliability simulations with PARMA+ for 13 SPEC2000 benchmark programs and for several distributions of large and small fault patterns and have compared them with accelerated fault injection simulations. PARMA+ has high accuracy and low computational complexity.

1. Introduction

Since cache memories occupy most of a processor's chip area and are highly vulnerable to soft errors, they greatly impact the reliability of a processor. In order to select an appropriate error protection scheme, a designer must estimate and compare the reliability levels of different protection schemes in addition to their energy, performance and area overheads. While many simulators exist to measure the performance of caches, and models such as CACTI [24] can estimate cache area and energy consumption accurately, formal, versatile and accurate models to estimate the FIT rate of caches against transient faults are lacking. It is predicted [1] that, by 2016, 100% of transient faults will be spatial multi-bit faults which cannot be modeled by existing approaches. A spatial multi-bit fault is a single fault event (called an SEU, for "Single Event Upset") that affects more than one bit in the cache array.

Cache protection is enforced at the granularity of protection domains. A protection domain is the set of bits protected by the same error protection code. For example, it can be a word, a set of words or a cache line.

Various schemes to protect caches are implemented in current commercial processors, such as:

1- Parity [10]: Parity detects an odd number of bit faults but an even number of bit faults goes undetected. Parity is good for write-through caches because a back-up copy is always present at a lower level memory and the correct copy can be recovered from there.

2- Single Error Correction Double Error Detection (SECDED) code: This code is common in commercial processors with write-back caches [15]. It can detect two faults and correct one, but a number of faults greater than two is not detected.

3- Cache scrubbing [14]: Cache scrubbing prevents the accumulation of correctable errors and their conversion to unrecoverable ones by reading cache lines periodically and activating their error codes in the process.

4- Early write-back [7]: Clean blocks have back-up copies in the next memory level and have higher reliability when a code's detection capability is more than its correction capability (such as parity and SECDED). Write-backs of dirty blocks before they are due for replacement increase the amount of clean data in the cache and therefore improve reliability.

5- Bit-interleaving [9]: In a bit-interleaved cache, bits of *different* protection domains are placed in adjacent memory cells. Hence, a spatial multi-bit fault may be converted to several single-bit faults in different protection domains, which can be corrected by SECDED.

6- Protection code interleaving: Instead of interleaving bits from different protection domains in the data array, one can also interleave protection codes across the bits of the array, without changing the layout of the array. In this case, adjacent bits are protected by different codes and are part of different protection domains, as in bit-interleaving.

Because these approaches (or a combination thereof) increase the costs of a cache implementation significantly [13, 9, 25], it is very important to select a cost effective combination of error protection schemes in caches. The first and foremost contribution of this technical report is a formal model to benchmark the failure rate of a cache in the presence of any set of spatial multi-bit fault patterns and any protection domain size and topology. The new model is called PARMA+. Fundamentally, it follows the modeling approach in PARMA [21], which is a tool to evaluate the resiliency of caches for single-bit SEUs. The model is applicable to all multi-bit fault

patterns and any major protection scheme used in industry and listed above. No model today can accurately predict the FIT rate of a cache under spatial multi-bit faults and under arbitrary configurations of faults patterns, protection domains, and error correction schemes.

We have compared the predictions of the PARMA+ model with results from accelerated fault-injection simulations for 13 SPEC2000 benchmark programs and found that the FIT rates estimated by PARMA+ are accurate while its computational complexity is low. Thus, PARMA+ can be easily deployed in rapid evaluations of various reliability options in caches.

In the next section, we first review background about transient faults, reliability evaluations, FIT rate computations and spatial multi-bit faults. We then expound the issues in modeling the impact of spatial multi-bit faults on cache reliability.

2. Background

One critical premise of any reliability study is the definition of the failure model. A cache failure happens when the error protection code of a protection domain cannot correct faults in the domain at the time of an access. Error codes fail with different number of bit faults, depending on their strength and on the state of the data, dirty or clean. Table 1 shows the number of faulty bits which fail a protection domain protected by common error codes. Clean data can tolerate more faults in general because a back up copy is available at the next level and can be re-fetched.

PARMA+ computes the FIT rate of an application. During our computations, we assume that hardware is working properly and we just focus on application failure. When we compute the FIT rate of an application, we implicitly assume that the execution of the application is repeated until it is failed. Unlike the case of hardware failure which may be repaired, we do not have any repair for application failure. The equations used later in this section follow this concept.

Transient faults can cause either Detected Unrecoverable Errors (DUEs) or Silent Data Corruption (SDCs) errors [21]. When a DUE is detected, application execution is terminated and thus failed. SDC errors are more subtle because they may or may not cause a failure when they propagate outside of the cache. Whether an SDC error propagated outside the cache will crash the application or not depends on many factors such as the exact location of the fault and the exact time at which the fault happened. Since PARMA+ in its current form only tracks the propagation of faults in the cache and not in other system components, we count an SDC as a terminating application failure like a DUE when it is propagated outside the cache. Therefore, in PARMA+, all errors (DUEs or SDCs) that are propagated outside the target cache are considered terminal failures.

The average discrete failure rate in a time interval $[t_1, t_2]$ is computed by well-known equation (1) [8, 5]. We use the same notations as in these sources. In equation (1), $R(t)$ denotes the probability that the system has survived until time t. t_1 is the cycle when the program starts (cycle 0) and t_2 is the cycle when it ends (t_2 is provided by the simulator).

Protection code	**Number of faulty bits causing failure**	
	Dirty block	Clean block
Parity	Any number of faulty bits	Even number of faulty bits
SECDED	More than 1 faulty bit	More than 2 faulty bits
DECTED	More than 2 faulty bits	More than 3 faulty bits

Table 1: Failure condition for different protection codes.

$$\text{Average failure rate} = \frac{R(t_1) - R(t_2)}{R(t_1) \times (t_2 - t_1)} \quad (1)$$

$R(0)$ equals 1 because cache has no data at t=0 and the application cannot be affected by soft errors because it has not started. Therefore, to compute equation (1), we need to calculate $R(t_2)$. $R(t_2)$ is the probability that the cache does not fail during the execution of the entire program. Since reads and write-backs are the only cache operations that can result in cache failure, $R(t_2)$ is the probability that the cache does not fail at any read or write-back in the cache between cycles 0 and t_2 (end of the program). $R(t_2)$ is computed by equation (2). In this equation, P_j is the probability of failure at access j and max is the total number of reads and write-backs in the program execution.

$$R(t_2) = \prod_{j=1}^{max} (1 - P_j) \quad (2)$$

The average failure rate shown in equation (1) can be converted to equation (3), which computes the average failure rate in one CPU cycle. T_{exe} is the number of cycles of a program execution ($t_2 - t_1$).

$$\text{Average failure rate} = \frac{1 - \prod_{j=1}^{max} (1 - P_j)}{T_{\text{exe}}} \quad (3)$$

To compute the FIT rate, we scale the failure rate of equation (3) to one billion hours as shown in equation (4).

$$\text{FIT rate} = \frac{(1 - \prod_{j=1}^{max} (1 - P_j)) \times 3600 \times 10^9}{T_{\text{exe}} \times Cycle\_Period} \quad (4)$$

Cycle_Period is the cycle time in seconds.

To compute P_j, we need to know the raw FIT rate and the distribution of fault patterns.

The raw FIT rate is the expected number of faults in one billion hours and is given in the International Technology Roadmap for Semiconductors (ITRS) [1]. To compute the SEU rate per bit in one cycle, the ITRS raw FIT rate is scaled down as shown in equation (5). In equation (5), $R(SEU)$ is

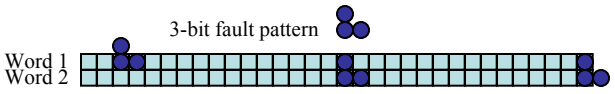

Figure 1: A spatial multi-bit fault can flip different number of bits in a protection domain depending on its location

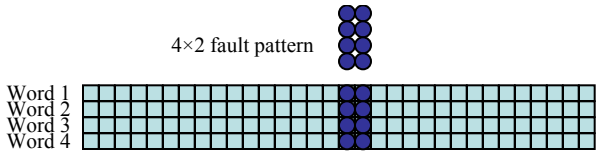

Figure 2: Failure of four words can be dependent on each other

the SEU rate per bit per cycle, FIT_{ITRS} is the ITRS FIT rate for a 1Mbit SRAM array, and f is the processor frequency.

$$R(SEU) = \frac{FIT_{\text{ITRS}}}{10^6 \times 3600 \times f \times 10^9} \quad (5)$$

Because a large number of faults are masked, the raw FIT rate grossly overestimates the cache error rate. For example, if a bit fault happens in a cache block and then the block is overwritten, the fault is masked and cannot cause a failure. Moreover, errors can be erased by error correction. The contribution of PARMA+ is to estimate these masking effects on the FIT rate for a given program and a given protection scheme.

The distribution of fault patterns can be obtained by beam injection experiments [12, 22]. In these experiments, the occurrence rates of various fault patterns are observed on a real chip and the probabilities of occurrence of every pattern in one SEU are estimated. These probabilities depend on the feature size, layout and other characteristics of a chip. They can also be obtained by simulations of physical devices. The probability that pattern i is observed when an SEU occurs is denoted by Q_i. If there are N possible fault patterns, we will have the following equation.

$$\sum_{i=1}^{N} Q_i = 1 \quad (6)$$

Given the raw SEU rate and the distribution of fault patterns, a model must assess the probability that the number of flipped bits since the last access exceeds the correction capability of the error protection code. Two challenges must be met to do this.

First, the number of bit faults due to an SEU in a domain varies with its fault pattern and its location, as illustrated in Figure 1. Thus the model must take into account the shape of the pattern and its location to evaluate the failure rate.

Second, if more than one SEU strike a domain between two consecutive accesses, the effects of the multiple multi-bit faults accumulate and may increase or decrease the probability of an access failure. If multiple multi-bit faults flip bits of different parts of a protection domain, the number of bit faults increases, but, if two SEUs overlap, some bit faults may cancel

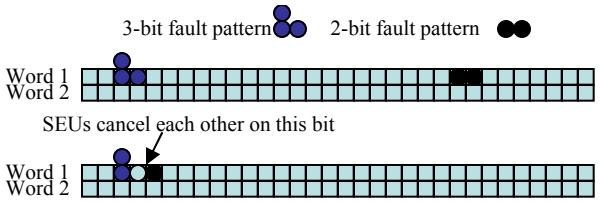

Figure 3: Two multi-bit fault patterns can add or subtract to each other

out. These two cases are illustrated in Figure 3. In the top of Figure 3, two multi-bit faults cause the total number of faults in word 1 to become four while in the bottom, the total number of bit fault in word 1 is two.

Additionally, there are dependencies between failures in adjacent domains. One large fault may cause failures in multiple domains. For example, Figure 2 shows a fault pattern that flips two bits in four vertically adjacent words. If all these words are dirty and protected by SECDED, they will all fail at the next access to them. If words 1, 2, 3 and 4 are accessed in that order, the probability that word 1 failed must be discounted from the probabilities of words 2, 3 and 4 failing, because if word 1 fails, then failing accesses to words 2, 3, and 4 will not happen. This is the most complex part of the PARMA+ model, addressed in Section 6 of the paper.

The rest of the technical report is organized as follows. Section 3 covers the related work. In Section 4, we explain basic assumptions of the model. Section 5 shows how to compute the probability of failure in one domain, independently of the failure of other domains. Section 6 upgrades the model to include failure dependencies with neighboring domains. Section 7 shows how to apply PARMA+ to different techniques such as bit-interleaving, cache scrubbing and DVFS. Section 8 checks the accuracy of PARMA+ by comparing the result of the model with accelerated fault-injection simulations and a previous tool called MACAU [20]. In Section 9, we will explain how we deal with the precision of floating-point numbers in PARMA+. Section 10 describes the PARMA+ tool and Section 11 concludes the technical report.

3. Related work

In order to compute the FIT rate of a cache for a program execution, the raw FIT rate is typically multiplied by the Architectural Vulnerability Factor (AVF). AVF is the probability that a bit fault is converted into a systemic failure. The AVF of a cache for single-bit faults is computed by the following procedure in [4]: Compute the total time between all reads or write-backs of words and divide that time by the simulation time multiplied by the number of cache words. An upper bound for AVF is called Temporal Vulnerability Factor (TVF)[23]. TVF is the fraction of data present and vulnerable in the cache among all data held in the cache. Like AVF, TVF is limited to single-bit faults and is not applicable to multi-bit faults and caches with error correction codes. In [19, 2] the

reliability of a cache is computed in a way similar to [4] by multiplying the raw FIT rate by the fraction of cache data vulnerable during the execution of a program. All these approaches are limited to single-bit faults and caches without error correction codes.

SoftArch [11] computes the cache Mean Time To Failure (MTTF) as if the same program execution is repeated until the first failure occurs. This model only applies to single-bit faults like in [4] and cannot model the effects of error correction codes.

Approximate analytical models [14, 18] have been proposed to estimate the expected time for two-bit faults in a word in order to set the cache scrubbing rate. These approximate models are not specific to programs. Furthermore, they are limited to two temporal single-bit faults and cannot deal with spatial multi-bit faults. In the same vein, a compound Poisson process is proposed in [16] to set the interleaving distance of SECDED codes to correct spatial multi-bit errors. However, this approach does not benchmark the failure rates for given programs.

PARMA [21] is a model that estimates the cache FIT rate in the presence of error codes. During a program execution, PARMA calculates the probability that each cache access fails and thus the application fails. PARMA tracks a block in and out of main memory and tracks accesses into the processor, so that SDCs and true/false DUEs are counted separately. However, PARMA is limited to single-bit SEUs which will not be seen in future technologies as all soft errors will come from spatial multi-bit faults.

In MACAU [20], the interval between two accesses to a word is modeled by a Markov chain. The Markov chain states track the number of bit faults in a word. For example, if a word has 32 bits, there can be 0 to 32 faults in a word and the Markov chain has 33 states. At each access, the probability that the cache has any number of faults can be computed by the Markov chain. MACAU can estimate the cache FIT rate only for a few specific spatial multi-bit fault patterns which were observed in a 65nm technology. MACAU only applies to protection domains of one word. It needs large matrix multiplications, which are very time-consuming. Another shortcoming of MACAU is that the evaluation of the failure rate of a protection domain is done independently of its neighbors. This omission may create large errors as we will show in Section 8.

Fault-injection experiments, either real-life or simulated, are the ultimate approach to estimating the MTTF or FIT rate. However, fault-injection experiments are costly and extremely time-consuming because of the large number of simulation runs needed to obtain a significant estimate, especially when events are rare. Because actual fault rates are extremely low, fault-injection simulations cannot be applied to actual, observed fault rates and can only be run for extremely high fault rates, far from reality.

4. Basic assumptions and equations of PARMA+

PARMA+ computes the FIT rate based on several inputs. One of the inputs is the protection domain. A protection domain can be a block, a word, a byte or any set of bits in a cache. Another input is the protection code in each protection domain. SECDED and parity are typical but stronger codes such as DECTED (Double Error Correction Triple Error Detection) are also possible.

The interval between two consecutive accesses to a protection domain is called a *vulnerability interval* because it is the time during which a domain is exposed to faults before any error correcting mechanism is activated. For example, if the last access to a word was at cycle 1000 and a read happens at cycle 2000, the vulnerability interval is [1000, 2000]. The length of the vulnerability interval in cycles is denoted L. In this example, L is equal to 2000-1000=1000.

We assume that at most one SEU can hit a protection domain in any one cycle (a fraction of a nanosecond). Thus, if the protection domain is one word, there can only be one SEU in a word in one cycle. The model could be refined to accept multiple SEUs in the same cycle, but, given current technology trends in which the FIT rate per megabit of SRAM is expected to stay around 1000 [1] for the foreseeable future, this added complexity would be futile. SEUs that happen in the same protection domain are referred to as "Domain SEUs" (DSEUs). The DSEU rate is larger than the SEU rate because a domain contains multiple bits.

The PARMA+ model can be applied to any memory array such as L1 cache, L2 cache, or even main memory. However, in this section, we focus on the reliability of an L2 cache, and a failure is caused by DUEs or SDC errors that happen when a block is read from L2 or written back by L2. In Sections 5 and 6, it is implicitly assumed that the protection domain is one block and the granularity of cache accesses is equal to the size of a protection domain. In Section 7, we will explain how PARMA+ models cache accesses which read multiple protection domains.

4.1. Illustrative example

To illustrate the equations of PARMA+, we use the following running example in this section. In this example, the cache contains 15 words as shown in Figure 4. Every word has 32 bits and is protected by SECDED. Moreover, all words are dirty. Thus, any word fails with two or more faulty bits.

In this example, an SEU has one of two bit fault patterns. The first fault pattern is a single-bit fault and the second one is a 2×2 fault in which 4 bits are flipped. The probabilities of the fault patterns in an SEU are both 0.5.

4.2. SEU rate in one protection domain (SEU rate)

The rate per bit and per cycle of each fault pattern i is equal to $R(SEU) \times Q_i$, i.e., the SEU rate per bit multiplied by the

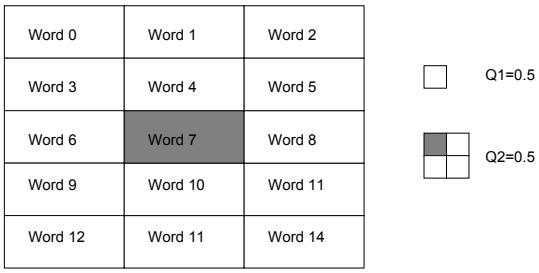

Figure 4: The running example of the technical report

probability that the SEU has fault pattern i. We say that a DSEU *occurs* in a domain when at least *one* bit in the domain is flipped by the fault. To compute the DSEU rate, we first compute the rate at which each fault pattern i occurs in the domain. For the case of a single-bit fault pattern, the rate is simply $R(SEU) \times Q_i \times B$, where B is the number of bits in the protection domain. For multi-bit fault patterns, the calculation is a bit more complex because multi-bit faults may straddle multiple protection domains.

The footprint of a fault pattern is the smallest rectangle that includes the pattern. For example, the footprint of fault pattern 2 of Figure 4 is a 2x2 square. We pick the North-West (N-W) bit of the footprint of the pattern to locate the fault in the cache array, although any other bit could be chosen. In Figure 4, the N-W bit of fault pattern 2 is shaded.

We define N^i_{DSEU} as the number of bits in the cache (inside or outside the domain) such that if the N-W bit of fault pattern i is pinned to one of these bits, at least 1 bit is flipped inside the domain. In order to compute N^i_{DSEU}, the PARMA+ tool pins the N-W bit of pattern i to different bits inside and around the domain and counts the number of cases when at least 1 bit is flipped inside the domain by the fault pattern. For example, if the protection domain is contained between rows G and F and between columns R and S of the cache array, and if all multi-bit fault patterns are confined to an $N \times M$ footprint, the following algorithm computes N^i_{DSEU}.

$$
\begin{aligned}
&\mathrm{N}^{\mathrm{i}}_{\mathrm{DSEU}} = 0;\\
&\mathrm{For}(\mathrm{l} = \mathrm{G} - \mathrm{N}; \mathrm{l} <= \mathrm{F}; \mathrm{l}++)\\
&\quad \mathrm{For}(\mathrm{n} = \mathrm{R} - \mathrm{M}; \mathrm{n} <= \mathrm{S}; \mathrm{n}++)\\
&\quad\quad \mathrm{If}(\text{fault pattern i located at bit position}\\
&\quad\quad\quad (\mathrm{l}, \mathrm{n}) \text{ flips at least one bit in the domain})\\
&\quad\quad\quad\quad \mathrm{N}^{\mathrm{i}}_{\mathrm{DSEU}}++;
\end{aligned}
$$

For the example of Section 4.1, Figure 5 shows in gray the bits of the cache array such that if the N-W bit of any fault pattern is pinned to any of them, the fault occurs in Word 7, i.e., at least one bit is flipped in Word 7. N^1_{DSEU} (for fault pattern 1) is equal to 32 because if fault pattern 1 is pinned to any one of the 32 bits of Word 7, the fault occurs in the domain. Fault pattern 2 occurs in Word 7 if its N-W bit is pinned to any bit of Word 7 (32 bits total) or any bit of Word 4 (32 bits total) or bit 31 of Word 3 or bit 31 of Word 6 (a total of 66 bits),

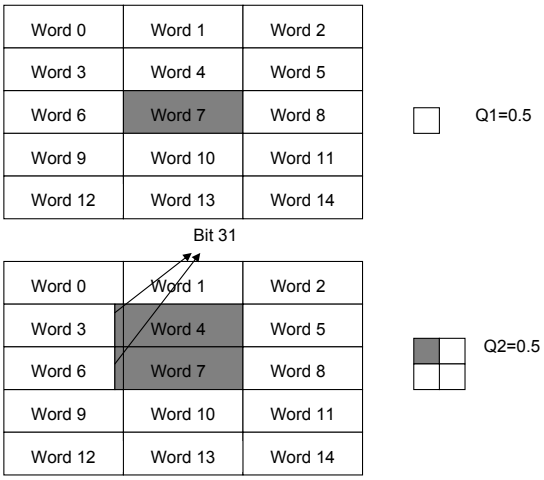

Figure 5: Bits contributing to N^i_{DSEU} for each of the two fault patterns

because at least one bit inside of Word 7 is flipped in all these cases. Consequently, N^2_{DSEU} is equal to 66.

The mean of N^i_{DSEU} over all possible fault patterns is called N_{DSEU} and is computed by equation (7) in which N is the number of fault patterns.

$$N_{\text{DSEU}} = \sum_{i=1}^{N} N^i_{\text{DSEU}} \times Q_i \tag{7}$$

For our example, N_{DSEU} is equal to $0.5 \times 66 + 0.5 \times 32 = 49$.

The DSEU rate in a protection domain is obtained by multiplying the SEU rate per bit by the average number of bit locations in the array such that, if an energetic particle hits that bit, a DSEU occurs.

$$R(DSEU) = R(SEU) \times N_{\text{DSEU}} \tag{8}$$

5. Failure of a domain independently of other domains

In this section, we ignore the failure dependencies that may exist between neighboring domains in the failure rate computation. Section 6 will take into account these dependencies.

Every access to a protection domain may result in domain failure. Accesses are reads and write-backs. The probability of failure due to an access to a domain is given by equation (9). The probability of access failure is the sum of probabilities of having c DSEUs in the domain so that the c DSEUs cause a failure in the accessed domain. In equation (9), L is the length of the vulnerability interval between two accesses in cycles and j is the access number. Since the model assumes at most 1 DSEU per cycle, the maximum number of DSEUs between two accesses to the domain is L.

$$P_j = \sum_{c=1}^{L} P(c \text{ DSEUs}) \times P(\text{access } j \text{ fails} \mid c \text{ DSEUs}) \tag{9}$$

The computation of P(c DSEUs) is done in Section 5.1. P(access j fails | 1 DSEUs) and P(access j fails | 2 DSEUs) independently of other domains are computed in Sections 5.2 and 5.3. Extensions to several DSEUs in a vulnerability interval are straightforward.

5.1. Probability of c DSEUs

The probability that one DSEU occurs in one cycle is denoted P_{DSEU} and is modeled by a Poisson process:

$$P_{\text{DSEU}} = R(DSEU) \times e^{-R(DSEU)} \tag{10}$$

The probability of c DSEUs occurring during L cycles is modeled by a Binomial distribution [21]:

$$P(c \text{ DSEUs}) = \binom{L}{c} \times (P_{\text{DSEU}})^c \times (1 - P_{\text{DSEU}})^{L-c} \tag{11}$$

5.2. Probability of access failure given a single DSEU in the domain

A failure happens in a domain when a certain number of bits are faulty. Table 1 shows the failure condition for several well-known error protection codes. We define N^i_{Fail} as the number of bits in the cache array such that if the N-W bit of fault pattern i is pinned to any one of them, a failure will happen when the domain is accessed next. N^i_{Fail} is always less or equal than N^i_{DSEU} and is different in dirty and clean blocks.

We consider a protection domain contained between rows G and F and between columns R and S of the cache array and multi-bit fault patterns included in an $N \times M$ footprint. N^i_{Fail} is computed as follows.

```
N^i_Fail = 0;
For(l = G − N; l <= F; l++)
  For(n = R − M; n <= S; n++)
    If(Pinning N − W of fault pattern i at bit location
          (l, n) causes domain failure)
            N^i_Fail++;
```

The mean number of fault locations that cause a failure across all patterns is denoted N_{Fail}:

$$N_{\text{Fail}} = \sum_{i=1}^{N} Q_i \times N^i_{\text{Fail}} \tag{12}$$

P(access j fails | 1 DSEU) is the fraction of bits counted in N_{DSEU} which cause a failure.

$$P(\text{access } j \text{ fails} \mid 1 \text{ DSEU}) = \frac{N_{\text{Fail}}}{N_{\text{DSEU}}} \tag{13}$$

In equation (13), N_{Fail} is given by equation (12) and N_{DSEU} is given by equation (7).

In the case of the example of Section 4.1, it takes more than one faulty bit to cause a failure, and therefore N^1_{Fail} is equal to 0 since pattern 1 is a single-bit fault and cannot cause more than one bit fault in a word. If the N-W bit of pattern 2 is pinned to any bit 0-30 of Words 4 or 7, there will be more than one bit fault in the domain. Hence, N^2_{Fail} is equal to 62. Figure 6 shows (in gray) the bits that contribute to N^2_{DSEU} but do not contribute to N^2_{Fail}. These bits are bit 31 of Words 3, 4, 6 and 7. Therefore we have:

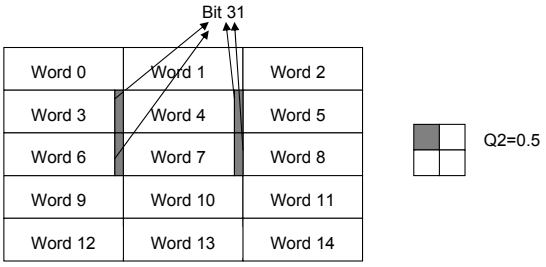

Figure 6: Bits in N^2_{DSEU} that cause no failure in Word 7

$$\frac{N_{\text{Fail}}}{N_{\text{DSEU}}} = \frac{0.5 \times 0 + 0.5 \times 62}{0.5 \times 32 + 0.5 \times 66} = 0.63$$

5.3. Probability of access failure given two or more DSEUs in the domain

Computing the probability of domain failure given two DSEUs in a vulnerability interval is similar to the case of one DSEU. Let $N^{i,m}_{\text{Fail}}$ be the number of cases in which patterns i and m occur in the domain and their superimposition causes domain failure. $N^{i,m}_{\text{Fail}}$ is computed as follows:

```
N^{i,m}_Fail = 0;
For(j = 0; j < N^i_DSEU; j++)
  For(l = 0; l < N^m_DSEU; l++)
    If(superimposition of patterns i and m causes failure)
            N^{i,m}_Fail++;
```

Note that $N^{i,m}_{\text{Fail}}$ is always less than $N^i_{\text{DSEU}} \times N^m_{\text{DSEU}}$ as multi-bit faults can cancel each other as shown in Figure 3. N_{Fail} for two DSEUs is now given by equation (14).

$$N_{\text{Fail}} = \sum_{i=1}^{N} \sum_{m=1}^{N} Q_i \times Q_m \times N^{i,m}_{\text{Fail}} \tag{14}$$

P(access j fails | 2 DSEU) is the fraction of combinations of two DSEUs occurring in the domain whose superimposition causes a domain failure:

$$P(\text{access } j \text{ fails} \mid 2 \text{ DSEUs}) = \frac{N_{\text{Fail}}}{(N_{\text{DSEU}})^2} \tag{15}$$

In equation (15), N_{Fail} is given by equation (14) and N_{DSEU} is given by equation (7).

For the cases of several DSEUs (three or more) in a vulnerability interval of L cycles, the procedure is conceptually

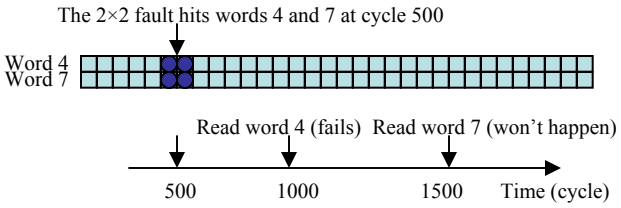

Figure 7: Failures of domains can be dependent on each other

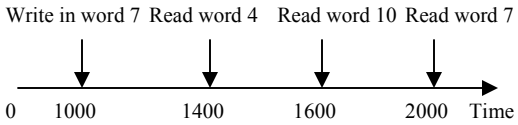

Figure 8: Accesses to word 7 and its neighbors

similar. For example, for three DSEUs, equation (14) would have three summation signs over patterns i, m and k. However, given the state of technology now and for the foreseeable future (until 2024) [1], it is futile to consider more than two DSEUs during a vulnerability interval because the probability of a single DSEU during a vulnerability interval is already extremely small (less than 10^{-15}).

6. Failure dependencies with neighboring domains

In Section 5, the probability of failure (equation (9)) was computed for one protection domain, independently of failures in other domains. However, because spatial multi-bit faults can fail more than one protection domain, the probability that an access in a domain fails is dependent on prior failures in other domains. To illustrate this further, Figure 7 shows an example in which a 2×2 fault occurs in Words 4 and 7 of Figure 4. In this example, when Word 4 is accessed at cycle 1000, the cache fails, raising a DUE and terminating execution. Hence, the access to Word 7 which was supposed to happen at cycle 1500 will not happen.

For the accesses of Figure 7, PARMA+ first computes P_j for the access to Word 4 using equations from Section 5. This probability takes into account that the N-W bit of the 2×2 fault pattern can hit any bit 0-30 of Word 4 or Word 1 (Word 1 is not part of Figure 7) before 1000. When the probability of access failure to Word 7 is calculated at cycle 1500, the failures caused by pinning the N-W bit of the 2×2 fault pattern to bits 0-30 of Word 4 during [0,1000] should be ignored. This is because if the 2×2 fault hits bits 0-30 of Word 4 during [0,1000], it would cause the failure of Word 4 but not Word 7 and the execution would terminate at the access to Word 4.

To compute, the probability of failure in Word 7 at cycle 1500, the vulnerability interval [0,1500] is divided into two subintervals: first, subinterval [0,1000] in which overlapping fault patterns causing failures in both Words 4 and 7 are not counted and, second, subinterval [1001,1500] in which all fault patterns causing failures in Word 7 are counted. Therefore N^i_{Fail} is computed differently in each of these subintervals.

For a spatial multi-bit fault to cause a failure dependency between two accesses in different domains, all three following conditions must be met:

1. The accesses should be either read, write-back or read-modify-writes (or read-before-writes) triggered by write operations. Writes that do not trigger a read operation cannot cause failures.

2. The fault pattern must cause a failure in both domains.

3. One of the accesses must be made during the vulnerability interval of the other access.

Before quantifying failure dependencies, we first define the concept of *neighboring* domains. For the case of one DSEU in a vulnerability interval, a protection domain is a *neighbor* of another domain if at least one of the fault patterns can cause a failure in both domains. In the example of Figure 4, Words 4 and 10 are neighbors of Word 7 because fault pattern 2 can cause the failure of Word 7 and of Words 4 or 10. For the case of two DSEUs in a vulnerability interval, a protection domain is a neighbor of another domain if the superimposition of any pair of fault patterns which occur in one domain can cause the failure of both domains. For three and more DSEUs, neighboring domains can be defined similarly. However, a model for more than two DSEUs in a vulnerability interval is not currently needed due to the extremely low probability of such occurrence.

During a PARMA+ simulation, every protection domain keeps track of accesses to its neighboring domains. In order for a failure dependency to affect the computation of P_j at access j, there should be a prior access to one of the neighbors of the domain satisfying the three failure dependency conditions listed above.

PARMA+ divides a vulnerability interval into subintervals delimited by accesses to neighboring domains. For example, the vulnerability interval [1000, 2000] of the last access in Figure 8 is divided into subinterval 1 [1000, 1400], subinterval 2 [1400, 1600] and subinterval 3 [1600, 2000]. Faults that cause a failure of Word 7 and of Words 4 or 10 during subinterval 1 must be ignored at cycle 2000 because if Word 4 fails at cycle 1400 or Word 10 fails at cycle 1600, then the access to Word 7 at 2000 will not happen, as the application would have been crashed before cycle 2000. For the same reason, all faults that cause a failure in both Words 7 and 10 must be ignored in subinterval 2. Finally, all faults must be counted in subinterval 3 because there is no dependency with neighbors.

DSEUs can be distributed among subintervals in different ways. In the example, a single DSEU can occur in one of the three subintervals, and two DSEUs can be distributed in six ways among the three subintervals: 2-0-0, 0-2-0, 0-0-2, 1-1-0, 1-0-1, and 0-1-1. The number of ways that c DSEUs can be distributed among U subintervals is denoted S. S is equal to the number of ways that c balls can be distributed into U bins

and is computed by equation (16).

$$S = \binom{U + c - 1}{c} \tag{16}$$

The probability of distribution k among S possible DSEU distributions is computed by equation (17). In this equation, L_t is the length of subinterval t in cycles and c_t is the number of DSEUs in subinterval t in distribution k. The denominator of equation (17) is the number of ways that c DSEUs can be distributed in L cycles and the numerator is the number of ways that c DSEUs can have in distribution k, given the length of each subinterval.

$$P(\text{Distribution } k) = \frac{\binom{L_1}{c_1} \times \binom{L_2}{c_2} \times ..... \times \binom{L_U}{c_U}}{\binom{L}{c}} \tag{17}$$

$c = c_1 + c_2 + c_3 + ... + c_U$ and $L = L_1 + L_2 + ... + L_U$.

In order to account for failures in neighboring domains, P(access j fails | c DSEUs) in equation (9) must be computed by equation (18).

$$P(\text{access } j \text{ fails} \mid c \text{ DSEUs}) = \sum_{k=1}^{S} P(\text{Distribution } k \mid c \text{ DSEUs}) \times P(\text{Domain failure with no failure in neighbors} \mid c \text{ DSEUs and Distribution } k) \tag{18}$$

P(Distribution k | c DSEUs) is computed by equation (17). Thus, we need to compute the second term on the right side of equation (18). We first explain how to compute P(Domain failure with no failure in neighbors | 1 DSEU and Distribution k).

(c=1) We define $N^i_{\text{SubFail } k}$ in a way similar to N^i_{Fail}. $N^i_{\text{SubFail } k}$ is the number of bits in the cache array such that if the N-W bit of fault pattern i is pinned to any of them, the domain fails but **none of its neighbors fail** given distribution k. The mean of $N^i_{\text{SubFail } k}$ over all patterns is:

$$N_{\text{SubFail } k} = \sum_{i=1}^{N} N^i_{\text{SubFail } k} \times Q_i \tag{19}$$

Then:

$$P(\text{Domain failure with no failure in neighbors} \mid 1 \text{ DSEU and Distribution } k) = \frac{N_{\text{SubFail } k}}{N_{\text{DSEU}}} \tag{20}$$

In the example of Figure 8 (the access at cycle 2000) with patterns of Figure 4, fault pattern 1 (single-bit fault) cannot fail any domain. Thus, N^1_{Fail} and $N^1_{\text{SubFail } k}$ are both equal to 0. N^2_{Fail} is 62. Among the 62 locations counted in N^2_{Fail}, 31 of them cause a failure in Word 4 and 31 of them cause a failure in Word 10. In subinterval 1, which ends with the access to Word 4, faults in Word 7 that cause a failure in Word 4 or Word 10 must be ignored; thus all faults must be ignored in subinterval 1 and $N^2_{\text{SubFail 1}}$ is equal to zero. In subinterval 2, which ends with the access to Word 10, any fault in Word 7 that causes a failure in Word 10 must be ignored because the application must have survived the access to Word 10 at cycle 1600. Hence, $N^2_{\text{SubFail 2}}$ is 31. Finally, in subinterval 3, after cycle 1600, there is no failure dependency with neighbors and no failure of Word 7 can be ignored. $N^2_{\text{SubFail 3}}$ is equal to 62.

Thus, we have:

$$\frac{N_{\text{SubFail 1}}}{N_{\text{DSEU}}} = \frac{0 \times 0.5 + 0 \times 0.5}{49} = 0$$
$$\frac{N_{\text{SubFail 2}}}{N_{\text{DSEU}}} = \frac{0 \times 0.5 + 31 \times 0.5}{49} = 0.31$$
$$\frac{N_{\text{SubFail 3}}}{N_{\text{DSEU}}} = \frac{0 \times 0.5 + 62 \times 0.5}{49} = 0.63$$

The DSEU can happen in any one of the three subintervals. Using equation (17), the probabilities of distributions 1, 2, and 3 are 0.4, 0.2 and 0.4, respectively. Hence, the probability of access failure computed by equation (18) is $0.4 \times 0 + 0.2 \times 0.31 + 0.4 \times 0.63 = 0.31$.

We now consider the case of two DSEUs in a vulnerability interval.

(c=2) For two DSEUs in a vulnerability interval, the process is similar to the case of one DSEU. We define $N^{i,m}_{SubFail\ k}$ as the number of bits in the cache array such that if fault patterns i and m occur in the domain and their superimposition causes domain failure, but none of its neighboring domains fail given distribution k. The following algorithm computes $N^{i,m}_{SubFail\ k}$.

$\text{N}^{\text{i,m}}_{\text{SubFail k}} = 0;$
For$(\text{l} = 0; \text{l} < \text{N}^{\text{i}}_{\text{DSEU}}; \text{l}++)$
  For$(\text{n} = 0; \text{n} < \text{N}^{\text{m}}_{\text{DSEU}}; \text{n}++)$
    If(superimposition of patterns i and m causes failure of domain but no neighboring domain fails given distribution k)
      $\text{N}^{\text{i,m}}_{\text{SubFail k}}++;$

$N_{\text{SubFail } k}$ is the mean of $N^{i,m}_{\text{SubFail } k}$ over all couples of patterns i and m.

Then:

$$P(\text{Domain failure with no failures in neighbors} \mid 2 \text{ DSEUs and Distribution } k) = \frac{N_{\text{SubFail } k}}{(N_{\text{DSEU}})^2} \tag{21}$$

This procedure can be generalized to compute equation (18) for more than two DSEUs.

7. Model extensions

So far, the model has been developed in the context of a standard cache array under nominal voltage and an error code protecting a contiguous domain. However, the model is applicable to other environments such as:

Bit-interleaving: PARMA+ is applicable to bit-interleaved cache arrays or arrays with interleaved protection codes. In both cases the only difference is that the protection domain is not made of contiguous bits in the array, and N^i_{DSEU}, N^i_{Fail} and other such variables are computed in an interleaved array. No equation of PARMA+ is changed. We will evaluate the accuracy of PARMA+ in bit-interleaved caches in Section 8.

Early write-back and cache scrubbing: PARMA+ is also applicable to caches with early write-back or/and scrubbing. In these cases, the model simulates the additional accesses. In the case of early write-backs this means extra write-backs. In the case of scrubbing this means extra reads at periodic times.

Different sizes of protection domain and access granularity: If the protection domain is smaller than the cache access granularity such as word-level protection in L2, each L2 write-back or read accesses several protection domains. To model this, N^i_{Fail} is counted as the number of bits in the cache array such that if the N-W bit of fault pattern i is pinned to any one of them, *at least one of the protection domains fails*. $N^{i,m}_{\text{Fail}}$ and other such variables (like $N^i_{\text{SubFail}\,k}$) are also computed similarly while other equations do not change. If the protection domain is larger than the granularity of a cache access, the entire domain will be accessed and the protection code will be checked. Hence, if the access granularity is smaller than the protection domain size, the tool always accesses the entire domain like the case in which the access granularity is equal to the protection domain size.

Dynamic Voltage and Frequency Scaling (DVFS): To reduce energy consumption, modern processors change their voltage and frequency dynamically. At lower voltages, the soft error rate increases because a smaller charge is stored in each SRAM cell [6]. This application of the model is more challenging. For each voltage level of the DVFS scheme, different inputs must be provided to the tool. First, the ITRS FIT rate in equation (5) must be replaced by the raw FIT rate for the various non-nominal voltages employed by the DVFS scheme. Second, the fault patterns and their probabilities (Q_i of equation (6)) must be determined for the non-nominal voltages as well.

8. Simulations

In this section, we compare the PARMA+ model with fault-injection simulations. Since the actual SEU rate is extremely small, fault-injection rates must be raised drastically so that fault-injection simulations can complete in a reasonable amount of time, especially given that a large number of them must be done for each design point. This is one of the reasons why formal models such as PARMA+ are useful in order to feasibly estimate FIT rates at actual SEU rates.

The system configuration in all the simulations reported here is shown in Table 2. We simulate 13 SPEC 2000 benchmarks with SimpleScalar [3]. Each benchmark is fast-forwarded for 100 million instructions and is run in detail for an additional 100 million instructions. We could use Simpoint

Parameter	Value
Functional Units	4 integer ALUs, 1 integer multiplier/divider, 4 FP ALUs, 1 FP multiplier/divider
LSQ Size / RUU Size	16 Instructions / 64 Instructions
Issue Width	4 instructions / cycle
Frequency	3 GHZ
L1 data cache	64KB, 4-way, 32 byte lines, 2 cycles latency
L2 cache	1MB unified, 8-way, 32 byte lines, 8 cycles latency
L1 instruction cache	16KB, 1-way, 32 byte lines, 1 cycle latency
Feature Size	32nm

Table 2: Evaluation parameters.

and run 100 million instructions after fast forwarding a certain number of instructions but we did not do that for two reasons. First, we would need to fast forward tens of billions of instructions which increases the simulation time significantly and we would not be able to run many fault injection experiments. Second, Simpoint is important in performance evaluations as it tries to reproduce the same CPI and cache miss rate as the entire benchmark. However, for our reliability evaluation, the CPI or cache miss rate do not matter.

We perform reliability simulation experiments on an L2 cache protected by SECDED and we call an error a failure when the error is propagated outside the L2 cache by either a read or write-back operation. PARMA+'s results depend on the layout of the cache because it considers the failure dependencies between neighboring domains and the dependencies vary in different layouts. We assume a typical layout for the cache and implement all dependency equations accordingly. In our simulations, the size of a cache row is equal to the size of cache line.

In our simulations, the probability of failure at access j is computed as

$$P_j = P(1\text{ DSEU}) \times P(\text{access } j \text{ fails} \mid 1\text{ DSEU}) + P(2\text{ DSEUs}) \times P(\text{access } j \text{ fails} \mid 2\text{ DSEUs})$$

This is equation (9) where we neglect the probability of having more than 2 DSEUs because the probability of having several DSEUs in a vulnerability interval is extremely small.

In PARMA+, the failure rate is computed in a single simulation run for each benchmark by equation (3). Since fault patterns used in this section causes failure of the cache with 1 DSEU, the probability of failure due to 2 DSEUs will not impact the FIT rate. Thus, in order to make the simulation faster, we do not consider failure dependencies for the case of two DSEUs. This is because the number of combinations which is computed by equation (16) is very large and this would

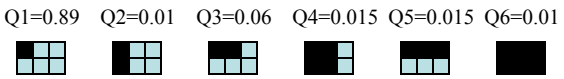

Figure 9: Small fault patterns (black squares show faulty bits)

greatly increase the simulation time. In PARMA+ simulations, P (access j fails | 1 DSEU) is computed by equations (18), (19) and (20) and P (access j fails | 2 DSEUs) is computed by equations (14) and (15).

Fault-injection simulations are run at least 10,000 times for each benchmark and the failure rate is estimated as the number of simulation runs in which the application fails because of transient faults in the L2 cache, divided by the total number of simulations. For some benchmarks with very low rate of significant faults events we had to increase the number of fault-injection simulations up to 30,000.

In order to understand the importance of cross-domain failure dependencies, we also compare the results of fault injections with a version of PARMA+ that does not take into account failure dependencies across domains. Another reason to have this comparison is because the running times of the simulations with or without dependencies are different. The PARMA+ model without failure dependencies of Section 5 runs at about the same speed as performance simulations on SimpleScalar. By contrast, the simulations including the failure dependencies of Section 6 for 1 DSEU run 10 times slower. We refer to the model of Section 5 (PARMA+ without cross-domain failure dependencies) as PARMA+$_{\text{light}}$. For PARMA+$_{\text{light}}$, P (access j fails | 1 DSEU) is computed by equations (12) and (13) and P (access j fails | 2 DSEUs) is computed by equations (14) and (15). Note that in simulations of this section, PARMA+ and PARMA+$_{\text{light}}$ have the same equations for two faults but the contribution of two DSEUs on the FIT rates on the results shown in this section is negligible.

We also compare PARMA+ with MACAU [20]. To the best of our knowledge, MACAU is the only existing model which can compute failure rates in presence of spatial multi-bit faults. However, it is only applicable to faults shown in Figure 9, it is limited to protection domain sizes of one word, and it cannot be applied to bit-interleaved caches or caches with interleaved error code. Moreover the computational complexity of MACAU is very high.

The deviation between estimates given by formal models such as PARMA+, PARMA+$_{\text{light}}$ and MACAU, and fault injection simulations is computed as:

$$\text{Deviation} = |1 - \frac{\text{Model failure rate}}{\text{Fault injection failure rate}}| \times 100$$

We report on five sets of simulations. First we compare PARMA+ and MACAU with fault-injection simulations with fault patterns shown in Figure 9 observed in previous physical beam injection experiments in a 65nm technology [22]. In Figure 9, faulty bits are shown in a 2×3 fault footprint (black bits are faulty). In this set of experiments, the raw SEU rate is 8.04×10^{14} FIT/Mbit and SECDED is applied at the word level since MACAU is only applicable to word-level protection domains.

Benchmark	PARMA+	MACAU
gcc	0.6	3.9
mesa	6.6	89.1
gzip	4.3	10.5
applu	0.4	2.2
twolf	6.9	23.2
mcf	0.7	9.9
crafty	1.1	15.3
equake	2.6	11.2
lucas	0.1	9.1
galgel	0.9	6.4
ammp	0.2	5.5
mgrid	1.3	8.5
swim	0.6	3.2
Average	2.0	15.1

Table 3: PARMA+ and MACAU deviations under fault patterns of Figure 9 (accelerated SEU rate).

Benchmark	Fault injection results	95% confidence interval
gcc	0.874	[0.871, 0.876]
mesa	0.002	[0.0011, 0.0028]
gzip	0.557	[0.550, 0.563]
applu	0.9892	[0.9889, 0.9894]
twolf	0.021	[0.0193, 0.0226]
mcf	0.8125	[0.809, 0.815]
crafty	0.449	[0.441, 0.456]
equake	0.0034	[0.0027, 0.0040]
lucas	0.781	[0.777, 0.784]
galgel	0.076	[0.071, 0.080]
ammp	0.9932	[0.9930, 0.9933]
mgrid	0.93	[0.928, 0.931]
swim	0.882	[0.879, 0.884]

Table 4: Fault injection results of Table 3 and their 95% confidence interval

Table 3 shows that the deviation of PARMA+ is on average 2.0% while the deviation of MACAU is 15.1%. Hence, the accuracy of PARMA+ is better than MACAU for the fault patterns that are applicable to MACAU. Table 4 shows the fault injection results (their absolute values which is teh fraction of experiments in which the L2 cache fails) used in Table 3 and their 95% confidence interval.

Second, we compare PARMA+ and PARMA+$_{\text{light}}$ to fault-injection simulations with fault patterns of Figure 9. In this set of experiments, the raw SEU error rate is 8.04×10^{14} FIT/Mbit and each cache block is protected by SECDED. We

Bench.	Figure 9 Patterns		Figure 10 Patterns	
	PARMA+	PARMA+$_{light}$	PARMA+	PARMA+$_{light}$
gcc	1.8	5.5	2.9	268.1
mesa	5.0	19.2	9.3	69 .0
gzip	2.5	15.8	4.9	325.7
applu	0.03	0.5	3.8	166.4
twolf	1.7	14.8	2.8	82.2
mcf	0.7	7.3	0.9	235.9
crafty	4.0	12.7	4.2	103.5
equake	3.0	9.8	8.7	65.0
lucas	2.3	10.7	4.5	364.4
galgel	1.6	4.9	4.0	22.7
ammp	0.08	0.3	2.4	50.1
mgrid	1.4	3.9	3.9	133.9
swim	1.1	5.5	5.1	178.2
Average	1.9	8.4	4.3	164.2

Table 5: PARMA+ and PARMA+$_{light}$ deviations under fault patterns of Figures 9 and 10 (accelerated SEU rate).

Benchmark	Fault injection results	95% confidence interval
gcc	0.882111	[0.879, 0.884]
mesa	0.0021	[00158, 00262]
gzip	0.5727	[0.566, 0.579]
applu	0.9922	[0.9920, 0.9923]
twolf	0.0242	[0.021, 0.027]
mcf	0.8165	[0.813, 0.819]
crafty	0.4583	[0.451, 0.465]
equake	0.00356	[0.0028, 0.0042]
lucas	0.7846	[0.780, 0.788]
galgel	0.0792	[0.074, 0.084]
ammp	0.9942	[0.9940, 0.9943]
mgrid	0.927	[0.926, 0.928]
swim	0.897	[0.895, 0.898]

Table 6: Fault injcetion results and their 95% confidence intervals on fault patterns of Figure 9 used in Table 5

ran 30,000 fault-injection simulations for mesa and equake to obtain statistically acceptable results. For this set of patterns, the PARMA+ model is highly accurate and its deviation from fault-injection simulations is on average 1.9%. By comparison, PARMA+$_{light}$ is less accurate with an average deviation of 8.4%.

In this case, since a fault pattern can flip bits in up to two domains, PARMA+$_{light}$ may count a fault twice in two neighboring domains and overestimate the failure rate.

In the third set of simulations, we compare PARMA+ and PARMA+$_{light}$ with fault-injection simulations for a set of large fault patterns in order to stress the models. Note that MACAU is not applicable to those large fault patterns. Figure 10 shows the fault patterns with their probability of occurrence. In Figure 10, black squares show faulty bits in fault patterns with up to an 8×8 fault footprint. Bit faults are in adjacent or non-adjacent bits and fault patterns are much more complex than the faults in Figure 9. These fault patterns were generated randomly with the goal to increase the complexity of the faults as much as possible. In this set of experiments, the raw error rate is 8.04×10^{12} FIT/Mb and each cache block is protected by SECDED. Table 5 shows that the deviations of PARMA+ and PARMA+$_{light}$ with respect to fault-injection simulations are 4.3% and 164.2%, respectively. While the accuracy of PARMA+ is very good in both sets of experiment. PARMA+$_{light}$ grossly overestimates the failure rate with the fault patterns of Figure 10 because fault patterns flip bits of up to eight consecutive rows and the model may count the same fault eight times. The 95% confidence intervals for fault injection experiments of Table 5 are shown in Tables 6 and 7.

Benchmark	Fault injection results	95% confidence interval
gcc	0.253	[0.245, 0.262]
mesa	0.0013	[0.00089, 0.00171]
gzip	0.083	[0.077, 0.088]
applu	0.34	[0.33, 0.35]
twolf	0.0104	[0.008, 0.012]
mcf	0.167	[0.160, 0.174]
crafty	0.165	[0.157, 0.172]
equake	0.0023	[0.00176, 0.00274]
lucas	0.1186	[0.112, 0.124]
galgel	0.0187	[0.0160, 0.0213]
ammp	0.611	[0.605, 0.616]
mgrid	0.332	[0.324, 0.339]
swim	0.1816	[0.174, 0.188]

Table 7: Fault injection results and their 95% confidence intervals on fault patterns of Figure 10 used in Table 5

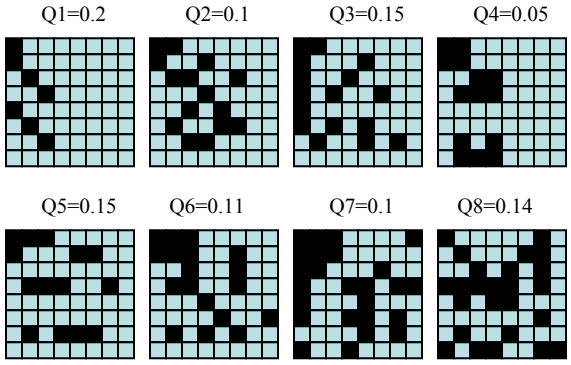

Figure 10: Large fault patterns (black squares show faulty bits)

In our fourth set of simulations, the bits in the L2 cache array are 2-way or 4-way interleaved. 64-bit words are protected by SECDED in blocks of 256 bits. In the case of the 2-way interleaved array, bits of the first 64-bit word are placed in bits 0, 2, 4,....126 and the third word is stored in bit locations

Bench.	2-way Interleaved		4-way Interleaved	
	PARMA+	PARMA+$_{\text{light}}$	PARMA+	PARMA+$_{\text{light}}$
gcc	2.6	126.8	7.3	88.9
mesa	3.4	23.3	0.001	0.002
gzip	5.2	272.5	10.3	280.2
applu	4.3	147.9	0.8	106.3
twolf	8.7	44.0	1.5	25.0
mcf	6.5	205.1	1.9	125.3
crafty	1.6	49.3	0.7	19.9
equake	6.4	14.5	0.00005	0.00007
lucas	7.3	289.2	7.8	185.6
galgel	3.3	37.7	4.0	22.4
ammp	2.8	46.4	4.1	39.8
mgrid	7.2	116.2	5.2	90.0
swim	3.7	217.1	1.0	137.5
Average	4.8	121.4	3.4	85.6

Table 8: PARMA+ and PARMA+$_{\text{light}}$ deviations in bit-interleaved caches (accelerated SEU rate).

Benchmark	Fault injection results	95% confidence interval
gcc	0.238	[0.230, 0.245]
mesa	0.0009	[0.00056, 0.00124]
gzip	0.0728	[0.067, 0.077]
applu	0.3348	[0.327, 0.342]
twolf	0.0056	[0.0041, 0.0070]
mcf	0.149	[0.142, 0.155]
crafty	0.1267	[0.120, 0.132]
equake	0.0013	[0.00089, 0.00171]
lucas	0.1145	[0.108, 0.120]
galgel	0.0181	[0.015, 0.020]
ammp	0.5727	[0.566, 0.579]
mgrid	0.2862	[0.278, 0.293]
swim	0.1809	[0.174, 0.187]

Table 9: Fault injection experiments and their 95% confidence intervals for fault injection experiments of Table 8 on the 2-way bit-interleaved cache

Benchmark	Fault injection results	95% confidence interval
gcc	0.1895	[0.182, 0.196]
mesa	0	[0, 0]
gzip	0.0619	[0.055, 0.066]
applu	0.3193	[0.311, 0.326]
twolf	0.0018	[0.0009, 0.0026]
mcf	0.1368	[0.130, 0.143]
crafty	0.0666	[0.061, 0.071]
equake	0	[0, 0]
lucas	0.1057	[0.100, 0.111]
galgel	0.0123	[0.010, 0.014]
ammp	0.4755	[0.468, 0.482]
mgrid	0.2123	[0.205, 0.219]
swim	0.1709	[0.164, 0.177]

Table 10: Fault injection results and their 95% confidence intervals for fault injection experiments of Table 8 on the 4-way bit-interleaved cache

128, 130, 132,...254. In the case of 4-way interleaving, bits of the first 64-bit word are placed in bits 0, 4, 8, ...,252 and the third 64-bit word is placed in bits 2, 4, 8, ..., 254. In this set of simulations, the raw error rate is 8.04×10^{12} FIT/Mb. We use the patterns of Figure 10. Table 8 shows that the deviations of PARMA+ and PARMA+$_{\text{light}}$ compared to fault-injection simulations in the 2-way interleaved cache array are on average 4.8% and 121.4%, respectively. In the 4-way interleaved cache array the deviations are 3.4% and 85.6% on average. Tables 9 and 10 show the 95% confidence intervals for the fault injection experiments used in Table 8. Hence, PARMA+ is also very accurate in the case of bit-interleaving. These results demonstrate once more that cross-domain dependencies must be accounted for in general.

In our last set of simulations, we compare the FIT rate predictions of PARMA+ and PARMA+$_{\text{light}}$ at the fault rate of 1150 FIT/Mb, which is the raw soft error rate according to ITRS. At this error rate, fault injection simulations are not feasible. So we cannot compare to them. Table 11 compares the FIT rate of PARMA+ and PARMA+$_{\text{light}}$ under fault patterns of Figures 9 and 10. Each cache block is protected by SECDED. The differences between PARMA+ and PARMA+$_{\text{light}}$ are much larger with the large fault patterns of Figure 10 because a fault pattern affects more domains.

The simulation time of PARMA+ is reasonable. Each benchmark simulation finished in up to three hours on our server. The most time-consuming part of PARMA+ simulations is the equations of Section 6. The simulation time of PARMA+ increases when fault patterns affect more rows as in Figure 10. In our simulations, PARMA+ increases the simulation time on average by around one order of magnitude (10 times) as compared to native Simplescalar.

The simulation time of PARMA+$_{\text{light}}$ is almost the same as performance simulations without PARMA+$_{\text{light}}$ because all equations except equations (9), (11) and (4) are independent from the benchmark program and can be computed at the beginning of the simulation. Equations (9), (11) and (4) are also very simple to compute.

9. Floating point precision of PARMA+

The mathematical model of PARMA+ is fairly simple. However, we may need to modify equation (2) in order to implement it on a practical machine.

In C and C++, the longest floating-point type is "long double" which has 80 bits in some implementations and 128 bits in others [17]. The number of decimal digits in the mantissa of

Bench.	Figure 9 Patterns		Figure 10 Patterns	
	PARMA+	PARMA+$_{\text{light}}$	PARMA+	PARMA+$_{\text{light}}$
gcc	497.9	582.8	6606.6	24902.5
mesa	0.6	0.8	31.7	70.6
gzip	24.82	305.5	2450.0	12250.1
applu	822.5	1018.1	7320.2	40556.3
twolf	8.4	9.8	376.6	691.1
mcf	561.3	679.5	5900.1	26492.5
crafty	131.8	147.9	3856.2	8321.6
equake	1.0	1.2	60.9	108.8
lucas	680.1	849.9	5558.8	33601.6
galgel	34.9	37.0	991.0	1468.8
ammp	165.1	188.1	2958.6	7562.1
mgrid	244.9	288.6	7239.4	13031.1
swim	790.9	977.1	7061.0	38641.9
Average	299.15	363.3	3600.8	14867.8

Table 11: PARMA+ and PARMA+$_{\text{light}}$ FIT rates under fault patterns of Figure 9 and 10 (actual SEU rate).

these two representations is 19 and 31, respectively. Because of rounding of the mantissa, $1-P_j$ in equation (2) will be equal to 1 if P_j is less than 10^{-19} (80-bit representation) or 10^{-31} (128-bit representation). If all P_js of a program execution cause this rounding error, R(t2) in equation (2) will become equal to 1 and the FIT rate estimate in equation (4) will become zero.

The criticality of the rounding problem depends on two factors.

1- Value of P_j.

2- Precision of the floating-point library.

One can determine at the beginning of the simulation whether there will be a rounding problem given a particular C compiler. P_j depends on several parameters: the protection domain size, the length of the vulnerability interval, the protection code, the fault patterns and the raw error rate per bit per cycle. For a specific configuration, all of these parameters are constant except for the length of the vulnerability interval, which is variable during a benchmark execution. For simple approximate computations of this section we do not consider domain failure dependences between domains. Let P be an approximation for the average of P_js. P can be approximated for example by considering the average vulnerability interval lengths in several benchmarks. If P is much greater than 10^{-19} or 10^{-31} (depending on the floating-point library), the FIT rate obtained by PARMA+ simulations is reliable. Otherwise, it may not be reliable because the probability of failure in many accesses may be ignored due to rounding errors.

If the floating-point library does not provide enough precision for a given configuration, we change $1-\prod_{j=1}^{max}(1-P_j)$ in equation (4) as follows to get rid of $1-P_j$.

$$1-\prod_{j=1}^{max}(1-P_j)=\sum_{j=1}^{max}P_j-\sum_{j=1}^{max}\sum_{i=j+1}^{max}P_jP_i + \sum_{j=1}^{max}\sum_{i=j+1}^{max}\sum_{k=j+1}^{max}P_jP_iP_k-.... \quad (22)$$

Equation (22) is also difficult to implement as we need to save all P_js from the beginning of the program. If a program is large, saving all failure probabilities can fill the entire memory space. One simple alternative is to approximate the right side of equation (22) by its first term, $\sum_{j=1}^{max}P_j$. If the value of $\sum_{j=1}^{max}P_j$ is much larger than other terms on the right side of equation (22), this approximation is valid. If P is 10^{-19} (for 80-bit long double implementation) and the program executes 1 trillion reads and write-backs in the simulated cache (arguably a number way beyond the reach of simulators in the foreseeable future), the value of $\sum_{j=1}^{max}P_j$ is around 10^{-10}, while the value of $\sum_{j=1}^{max}\sum_{i=j+1}^{max}P_jP_i$ is around 10^{-20}. Therefore, $\sum_{j=1}^{max}P_j$ is much larger than all other terms and is a reliable approximation to equation (22). Note that if P is less than 10^{-19}, the approximation would be more accurate.

10. PARMA+ tool

This section explains the structure of the PARMA+ tool. The tool is implemented on top of SimpleScalar and it measures the FIT rate of the L2 cache. In the PARMA+ tool, it is assumed that the size of a cache row is equal to the size of a cache line.

The tool has an interface which receives several inputs as follows.

1- Raw FIT rate (FIT/Mbit): This is the rate at which cache bits are flipped by energetic particles.

2- Processor frequency (GHZ): The frequency of the processor is fed into the tool.

3- Access granularity (bit): The number of bits read in each cache access.

4- Domain size (bit): Number of bits protected by an error protection code.

5- Protection code: There are three options in the tool. 2 refers to SECDED, 1 is parity and 0 is no protection.

6- Mode: The PARMA+ tool has 4 operating modes. Mode 1 refers to the case in which up to two DSEUs are considered in each vulnerability interval and the failure dependencies are considered for both 1 DSEU and 2 DSEUs. In mode 1, P_j is

computed as:

$$P_j = P(1\text{ DSEU}) \times \sum_{k=1}^{S} P(\text{Distribution } k \mid 1\text{ DSEU}) \times$$
$$P(\text{Domain failure with no failure in neighbors} \mid 1\text{ DSEU and}$$
$$\text{Distribution } k) + P(2\text{ DSEUs}) \times \sum_{k=1}^{S} P(\text{Distribution } k \mid 2$$
$$\text{DSEUs}) \times P(\text{Domain failure with no failure in neighbors} \mid 2$$
$$\text{DSEUs and Distribution } k)$$

Mode 2 refers to the case in which there is at most one DSEU in a vulnerability interval and the failure dependencies are computed. In mode 2, P_j is computed as:

$$P_j = P(1\text{ DSEU}) \times \sum_{k=1}^{S} P(\text{Distribution } k \mid 1\text{ DSEU}) \times$$
$$P(\text{Domain failure with no failure in neighbors} \mid 1\text{ DSEU and}$$
$$\text{Distribution } k)$$

Mode 3 refers to the case in which one DSEU is considered and failure dependencies are ignored. In mode 3, P_j is equal to:

$$P_j = P(1\text{ DSEU}) \times \frac{N_{\text{Fail}}}{N_{\text{DSEU}}}$$

Mode 4 refers to the case in which up to 2 DSEUs are considered but failure dependencies are ignored for both the case of 1 DSEU and 2 DSEUs. In this mode, P_j is:

$$P_j = P(1\text{ DSEU}) \times \frac{N_{\text{Fail}}}{N_{\text{DSEU}}} + P(2\text{ DSEUs}) \times \frac{N_{\text{Fail}}}{(N_{\text{DSEU}})^2}$$

7- Interleaving degree: This determines the degree of bit-interleaving.

8- Fault patterns and their probability: Currently, the tool receives up to 10 fault patterns which are contained in an 8×8 square.

To compute FIT rates, the tool first runs an initialization function which is run only once at the beginning of the program. The initialization function first reads the input file and saves the inputs in some variables. Then it computes N^i_{DSEU}, N^i_{Fail} and $N^{i,m}_{\text{Fail}}$ (in modes 1 or 4). Furthermore, the initialization function decides how the tool deals with the approximation problem based on the input parameters.

At the time of each read or write-back, the probability that an access fails is computed. This probability is computed based on the operating mode. When failure dependencies are considered, $N^i_{SubFail\,k}$ and/or $N^{i,m}_{SubFail\,k}$ are computed for each access. This step is time-consuming and greatly increases the simulation time. When an access happens to a protection domain, the neighbors of that protection domain save the time and the type of the access as they would need this information later when they consider dependencies.

After computing the probability of an access failure, we use that either in $\sum_{j=1}^{max} P_j$ or in equation (2). At the end of the program, PARMA+ prints the average failure rate and FIT rate.

11. Conclusions

Reliability is a critical concern in today's computer systems. Since caches have a great impact on system reliability, the accurate modeling of cache FIT rates is important. Without an accurate cache FIT rate model, designers cannot configure reliability features in caches in an optimum way. This may cause them to provide higher or lower reliability than is required, with either higher overheads or higher vulnerability.

Since all fault patterns will be multi-bit from the year 2016, there is an urgent need to model cache FIT rates in presence of multi-bit faults. This technical report introduces the PARMA+ model capable of estimating FIT rates under all possible sequences of multi-bit faults with very high accuracy and low simulation times. Our evaluations and comparisons with fault-injection simulations demonstrate that it is critical to include failure dependencies across domains, especially for large spatial faults. PARMA+ can model the FIT rate of a cache equipped with major existing reliability features such as bit-interleaving, early write-back, scrubbing and various common error protection schemes. Furthermore, it is able to model faults with any set of patterns and any cache configurations including low power techniques such as DVFS.

In our future work, we will extend PARMA+ to models which are proposed in academic papers (not implemented in commercial processors) such as [13, 9]. We will also simulate more sets of fault patterns with the tool.

References

[1] "International Technology Roadmap for Semiconductors (ITRS), 2011 Edition."

[2] H. Asadi, V. Sridharan, M. Tahoori, and D. Kaeli, "Vulnerability analysis of L2 cache elements to single event upsets," in *Proc. of Design Automation and Test conference in Europe (DATE)*, 2006, pp. 1276–1281.

[3] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An Infrastructure for Computer System Modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[4] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. Mukherjee, and R. Rangan, "Computing Architectural Vulnerability Factors for Address-based Structures," in *Proc. of International Symposium on Computer Architecture (ISCA)*, 2005, pp. 532–543.

[5] M. Finkelstein, *Failure Rate Modeling for Reliability and Risk*. Springer, 2008.

[6] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: simple techniques for reducing leakage power," in *Proc. of International Symposium on Computer Architecture (ISCA)*, 2002, pp. 148–157.

[7] P. Genua, "Error Correction and Error Handling on PowerQUICC™ III Processors," DOI=http://www.freescale.com/files/32bit/doc/app_note/AN3532.pdf.

[8] P. Hoang, *Handbook of Engineering Statistics*. Springer, 2006.

[9] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe, "Multi-bit Error Tolerant Caches Using Two Dimensional Error Coding," in

Proc. of International Symposium on MicroArchitecture (MICRO), 2007, pp. 197–2009.

[10] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded sparc processor," *IEEE Micro*, vol. 25, pp. 21–29, 2005.

[11] X. D. Li, S. V. Adve, P. Bose, and J. A. Rivers, "SoftArch:An Architecture Level Tool for Modeling and Analyzing SoftErrors," in *Proc. of International Conference on Dependable Systems and Networks (DSN)*, 2005, pp. 496–505.

[12] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of Multi-bit Soft Error Events in Advanced SRAMs," in *Proc. of IEEE International Electron Devices Meeting*, 2003, pp. 21.4.1–21.4.4.

[13] M. Manoochehri, M. Annavaram, and M. Dubois, "CPPC: Correctable Parity Protected Cache," in *Proc. of International Symposium on Computer Architecture (ISCA)*, 2011, pp. 223–234.

[14] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt, "Cache Scrubbing in Microprocessors: Myth or Necessity?" in *Proc. of IEEE Pacific Rim Symposium on Dependable Computing (PRDC)*, 2004, pp. 37–42.

[15] N. Quach, "High availability and reliability in the Itanium processor," *IEEE Micro*, vol. 20, no. 5, pp. 61–69, 2000.

[16] R. W. S. Baeg, S. Wen, "Sram Interleaving Distance Selection with a Soft Error Failure Model," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, pp. 2111–2118, 2009.

[17] G. M. Saeed, *An Introduction to Object-Oriented Programming in C++*. Morgan Kaufmann Publishers Inc., 2001.

[18] A. Saleh, J. Serrano, and J. H. Patel, "Reliability of Scrubbing Recovery Techniques for Memory Systems," *IEEE Transactions on Reliability*, vol. 39, no. 1, pp. 114–122, 1990.

[19] V. Sridharan, H. Asadi, M. B. Tahoori, and D. Kaeli, "Reducing Data Cache Susceptibility to Soft Errors," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 353–364, 2006.

[20] J. Suh, M. Annavaram, and M. Dubois, "MACAU: A Markov model for reliability evaluations of caches under single-bit and multi-bit upsets," in *Proc. of International Symposium on High Performance Computer Architecture (HPCA)*, 2012, pp. 3–14.

[21] J. Suh, M. Manoochehri, M. Annavaram, and M. Dubois, "Soft error benchmarking of L2 caches with PARMA," in *Proc. of International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2011, pp. 85–96.

[22] A. D. Tipton, J. A. Pellish, J. M. Hutson, R. Baumann, X. Deng, A. Marshal, M. A. Xapsos, H. S. Kim, M. R. Friendlich, M. J. Campola, C. M. Seidleck, K. A. LaBel, M. H. Mendenhall, R. Reed, R. Schrimpf, R. Weller, and J. D. Black, "Device-Orientation Effects on Multiple-Bit Upset in 65 nm SRAMs," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, pp. 2880–2885, 2008.

[23] S. Wang, J. Hu, and S. Ziavras, "On the characterization and optimization of on-chip cache reliability against soft errors," *IEEE Transaction on Computers (TC)*, vol. 58, no. 9, pp. 1171–1184, 2009.

[24] S. J. E. Wilton and N. P. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.

[25] D. H. Yoon and M. Erez, "Memory Mapped ECC: Low-Cost Error Protection for Last Level Caches," in *Proc. of International Symposium on Computer Architecture (ISCA)*, 2009, pp. 83–93.