In the Proceedings of the International Conference on Dependable Systems and Networks(DSN'07), June 2007.

Architecture-Level Soft Error Analysis: Examining the Limits of Common Assumptions *

Xiaodong Li†, Sarita V. Adve†, Pradip Bose‡, Jude A. Rivers‡
†Department of Computer Science
University of Illionis at Urbana-Champaign
{xli3,sadve}@uiuc.edu
‡IBM T.J. Watson Research Center
Yorktown Heights, NY
{pbose,jarivers}@us.ibm.com

Abstract

This paper concerns the validity of a widely used method for estimating the architecture-level mean time to failure (MTTF) due to soft errors. The method first calculates the failure rate for an architecture-level component as the product of its raw error rate and an architecture vulnerability factor (AVF). Next, the method calculates the system failure rate as the sum of the failure rates (SOFR) of all components, and the system MTTF as the reciprocal of this failure rate. Both steps make significant assumptions. We investigate the validity of the AVF+SOFR method across a large design space, using both mathematical and experimental techniques with real program traces from SPEC 2000 benchmarks and synthesized traces to simulate longer real-world workloads. We show that AVF+SOFR is valid for most of the realistic cases under current raw error rates. However, for some realistic combinations of large systems, long-running workloads with large phases, and/or large raw error rates, the MTTF calculated using AVF+SOFR shows significant discrepancies from that using first principles. We also show that SoftArch, a previously proposed alternative method that does not make the AVF+SOFR assumptions, does not exhibit the above discrepancies.

1 Introduction

Radiation induced soft errors represent a major challenge to exploiting the benefits from continued CMOS scaling. Soft errors or single event upsets are transient errors caused by high energy particle strikes such as neutrons from cosmic rays and alpha particles from the IC packaging materials. These errors can be catastrophic to program execution by flipping bits stored in storage cells or changing the values being computed by logic elements. While there is still a lack of consensus on the exact soft error rates (SER) of specific circuits, it is clear that the SER per *chip* is growing substantially due to the increasing number of devices on a chip [3, 5, 10, 12].

*This work is supported in part by an IBM faculty partnership award, the MARCO/FCRP Gigascale Systems Research Center, the National Science Foundation under Grant No. CCF-0541383, and an equipment donation from AMD.

If a particle strike causes a bit to flip or a piece of logic to generate a wrong result, we call the bit flip or the wrong result a *raw soft error*. Fortunately, not all raw soft errors cause the program to fail. For example, a soft error in a functional unit that is not currently processing an instruction or in an SRAM cell that is not storing useful data will not harm the execution. Such an error is said to be masked. Research has shown that there is a large masking effect at the architecture (and micro-architecture) levels [2, 4, 6, 9, 14, 15]; e.g., Wang et al. [14] report more than 85% masking.

This paper concerns the impact of architectural masking on the methodology to compute the widely used metric of mean time to failure (MTTF) for soft errors for modern processors.

A widely used methodology to compute MTTF uses two simple steps [9], illustrated in Figure 1: (1) The **AVF step** calculates the failure rate of each individual processor *component* (e.g., ALU, register file, issue queue) as the product of its raw failure rate and a derating factor that accounts for masking. Mukherjee et al. formalized the notion of a derating factor as the architectural vulnerability factor (AVF) [9] and showed how to calculate it for various architectural components [1, 9]. (2) The **SOFR step** calculates the failure rate of the entire processor (or any system) as the Sum Of the Failure Rates (SOFR) of the individual components of the processor or system (as calculated in the AVF step). It calculates the MTTF of the processor (or system) as the reciprocal of its failure rate.

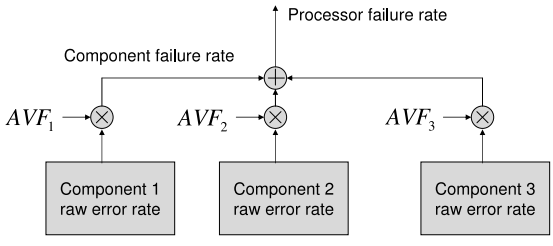


Figure 1. The AVF and SOFR steps for MTTF.

Both the AVF and SOFR steps implicitly make certain assumptions about the statistical properties of the underlying error process. While these assumptions, described below, may

hold for the raw error process, it is unclear whether they hold for the architecturally masked process. The goal of this paper is to examine the validity of these assumptions underlying the mathematical basis of the AVF and SOFR steps, and the implications of these assumptions for evaluating soft error MTTF for real systems.

AVF+SOFR assumptions.

A key assumption behind the AVF step is that the probability of failure due to a soft error in a given component is uniform across a program's execution. This allows a single AVF value to be used to derate the raw error rate of a component. The uniformity assumption is reasonable for raw error events since the probability of a high energy particle strike is no different at different points in the program's execution for most realistic scenarios. However, it is unclear that the assumption holds after incorporating architectural masking. Similarly, a well-documented assumption for the SOFR step is that the time to failure for a given component follows an exponential distribution. Again, the assumption is reasonable and widely accepted for raw error events, but it is unclear that it holds for failures after architectural masking.

Thus, both the AVF and SOFR steps make assumptions about the error process that may be considered questionable, once architectural masking effects are taken into account. The question we address is: Under what conditions (if any) does the violation of the above AVF+SOFR assumptions introduce significant errors in the calculation of the MTTF?

Contributions of this work.

We answer the above question through both mathematical and experimental analysis. Our rigorous mathematical methods identify the assumptions of the AVF+SOFR method, and using some synthesized workloads, analyze the value ranges of various parameters for which the AVF+SOFR assumptions do or do not hold. To validate the conclusions on real world workloads and quantify the relative error of the AVF+SOFR method, we design simulation-based experiments to explore a wide design space.

We find that the impact of the above assumptions on the MTTF calculation depends on three parameters related to the environment, system, and the workload respectively: (1) the raw error rate of the individual components, (2) the number of components in the system on which SOFR is applied, and (3) the length of the full execution or the longest repeated phase of the workload. Specifically, our evaluations show the following.

First, for systems where the individual components have small raw error rates, the total number of components is small, and where the workload consists of repeated executions of a short program, the AVF+SOFR assumptions introduce negligible error. To our knowledge, previously published work using the AVF+SOFR methodology considers systems and workloads that obey the above constraints. This result is by itself significant since it, for the first time, validates the mathematical basis for using the AVF+SOFR methodology.

Second, our results show that the AVF+SOFR method can result in large discrepancies in MTTF (up to 100%) for large raw error rates of individual components (e.g., as would be the case in space or in accelerated tests or with components consisting of many millions of bits) and/or systems that have many components (e.g., large clusters of thousands of processors) and/or long-running workloads with different utilization characteristics over large time windows (e.g., server workloads that run at high utilization in the day but low utilization in the night). This problematic part of the design space is certainly much smaller and less common than the space over which AVF+SOFR is valid; however, it is not negligible and represents several realistic systems. Our results give a note of caution against blind use of the AVF+SOFR method for such systems.

Finally, given the limitations of AVF+SOFR identified here, we briefly explore alternative methods for soft error analysis with architectural masking. Traditionally, fault injection in low-level (RTL) simulators has been used (e.g., [2, 4, 14]). This technique does not make the AVF+SOFR assumptions, but requires running numerous experiments that make it impractically slow for architecture-level work (which usually requires simulating long workloads). A more recent methodology called SoftArch [6] uses a probabilistic model based on first principles coupled with architecture-level simulation. SoftArch does not make the AVF+SOFR assumptions. We show here that SoftArch does not exhibit the MTTF discrepancies shown by AVF+SOFR. These experiments are not meant as a complete validation of SoftArch or a full comparison between SoftArch and AVF+SOFR. A fair comparison would require applying both methods to all processor components and using all the advanced optimizations of AVF+SOFR which is beyond the scope of this work. Rather, these experiments point to future work to determine the best combination of methodologies that will provide the best MTTF estimates across all relevant scenarios.

2 Background

2.1 Failure Rate, MTTF, and FIT Rate

Three common terms are often used to discuss system reliability: failure rate, mean time to failure (MTTF), and failures in time (FIT) [13].

The failure rate at time t is the conditional probability that the component will fail in the time interval $[t, t+dt]$, given that it has not failed until time t. When the distribution of the time to failure is exponential, the failure rate is a constant and does not vary with time. We use λ to denote this constant failure rate.

The mean time to failure (MTTF) of a component is simply the expected (average) time to failure. For components with exponentially distributed time to failure, $MTTF$ is simply the reciprocal of the constant failure rate $(1/\lambda)$.

The metric of failures in time or FITs is defined as the number of failures per one billion hours of operation. Often, FITs are referred to as the failure rate and the equation $FIT = \frac{10^9}{MTTF}$ is used. However, this assumes that the failure rate is constant in time, and equivalently, the time to failure follows an exponential distribution.

2.2 The AVF Step and its Assumptions

In a given cycle, only a fraction of the bits in a processor storage component and only some of the logic components will affect the final program output. A raw error event that does not affect these critical bits or logic components has no adverse effect on the program outcome. Mukherjee et al. used the term architecture vulnerability factor (AVF) to express the probability that a visible error (failure) will occur, given a raw error event in a component [9]. The AVF for a hardware component can be calculated as the percentage of time the component contains *Architecturally Correct Execution* (ACE) bits (i.e., the bits that affect the final program output). Thus, for a storage cell, the AVF is the percentage of cycles that this cell contains ACE bits. For a logic structure, the AVF is the percentage of cycles that it processes ACE bits or instructions.

Mukherjee et al. calculate the FIT rate of a processor component as the product of the component's AVF and its raw FIT rate (i.e., the FIT rate of the component if every bit were ACE). Denoting the raw FIT rate of the component as λ_c (also called the raw soft error rate or raw SER) and its AVF as AVF_c, they derive the MTTF of the component as:

$$MTTF_c = \frac{1}{\lambda_c \cdot AVF_c} \quad (1)$$

We show in Section 3.1 that an assumption underlying the above equation is that the time to failure for a program is uniformly distributed over the program. We explore the cases where this assumption is and is not true to assess the validity of the AVF step.

2.3 The SOFR Step and its Assumptions

Sum of failure rates (SOFR) is an industry standard model for combining failure rates of individual processor (or system) components to give the failure rate and MTTF of the entire processor (or system). Let the system contain k components with failure rate of component i as $FailureRate_i$ (which is assumed to be the reciprocal of the MTTF of component i or $1/MTTF_i$). The SOFR model calculates the failure rate ($FailureRate_{sys}$) and the MTTF ($MTTF_{sys}$) of the system as:

$$FailureRate_{sys} = \sum_{i=1}^{k} FailureRate_i = \sum_{i=1}^{k} \frac{1}{MTTF_i} \quad (2)$$

$$MTTF_{sys} = \frac{1}{FailureRate_{sys}} \quad (3)$$

The SOFR model makes two major assumptions [13]. First, it assumes that each component has a constant failure rate (i.e., exponentially distributed time to failure) and the failures for different components are independent of each other. Section 3.2 shows that architectural masking may violate this assumption in some cases. Second, the SOFR model assumes a series failure system; i.e., the first instance of a component failure causes the entire processor to fail. This assumption holds if there is no redundancy in the system. Since our focus is on the impact of program-dependent architectural masking on the statistical properties of the failure process, we continue to make this assumption as well and focus only on the first assumption.

3 Examining the Limits: An Analytical View

This section uses mathematical analysis to understand the limits of the basic assumptions underlying the AVF+SOFR methodology for estimating MTTF for soft errors. Later sections back these results with detailed Monte-Carlo simulations for actual workloads.

Our analysis makes two assumptions that are also made by the AVF+SOFR methodology.

(1) Inter-arrival times for raw errors in a component are independent and exponentially distributed with density function $\lambda e^{-\lambda t}$. It is reasonable to assume that the time to the next high energy particle strike is independent of the previous strike and is exponentially distributed (the process is memoryless). In practice, there is some device- and circuit-level masking, which could possibly render the raw error process that is subject to architectural masking as non-exponential. In our experiments, however, we do not have this low-level masking information available; we therefore assume the best case for the AVF+SOFR methodology – that the inter-arrival time for raw errors before any architectural masking is an exponential process with density function $\lambda e^{-\lambda t}$. We refer to λ as the *raw error rate.*

(2) The workload runs in an infinite loop with similar iterations of length L. This work considers the effect of real application workloads. For a workload that runs for a finite time, there is a possibility that no failure occurs during its execution. For a meaningful interpretation of MTTF for a system running such a workload, we assume that the workload runs repeatedly in a loop until the first failure. All iterations of this loop are identical and each represents a single invocation of the original workload. We refer to the size of this loop iteration as L. Workloads that are naturally infinite also run in a loop. We assume that such a workload also consists of identical iterations, each of size L. This assumption is trivially satisfied since L can potentially be infinite. (All the prior work on AVF+SOFR has been in the context of finite workloads.)

We additionally assume that program failure occurs if a raw error is not masked. Although the time to failure and the time to the next raw error event are continuous random variables, for convenience, we often consider time in units of processor cycles below (for architectural masking, for a given cycle, all raw error events during . any part of the cycle are either masked or not masked).

3.1 The AVF Step: MTTF for an Isolated Functional or Storage Unit

The AVF step computes the MTTF of a single component of the processor using equation 1. We examine the validity of this step by deriving the MTTF of a given component from first principles.

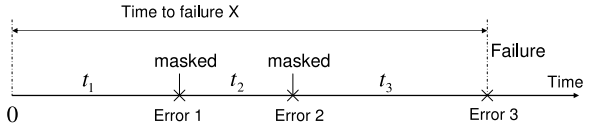


Figure 2. Sequence of raw error events. t_i is the time between two raw error events and is exponentially distributed. X is a random variable representing the time to the first raw error event that is not masked and leads to program failure. The figure shows a case where $X = t_1 + t_2 + t_3$.

Figure 2 illustrates a sequence of raw error events with inter-arrival times of $t_1, t_2, .., t_n, ...$ Each of these times is an instance of a random variable, say T, with exponential density function $\lambda e^{-\lambda t}$. Each raw error has some probability of being masked. Failure occurs at the first raw error that is not masked.

Let X be the random variable that denotes the time to failure. Then $X = t_1 + t_2 + .. + t_k$ if the first $k-1$ raw errors are masked and the kth raw error is not masked. Thus, $X = \sum_{i=1}^{K} t_i$, where K is a random variable such that $K = k$ denotes the event that the first $k-1$ raw errors are masked and the kth raw error is not masked.

Now the MTTF of the component is simply the expected value of X, $E(X)$. Using a standard result for the expectation of a sum of random variables [11], it follows that: $MTTF = E(X) = E(K)E(T)$. We know that $E(T) = \frac{1}{\lambda}$ (this would be the MTTF if there were no architectural masking and every raw error resulted in failure). Thus,

$$MTTF = E(K)\frac{1}{\lambda} \tag{4}$$

Comparing with equation 1, to validate the AVF step, we would need to show that $E(K) = \frac{1}{AVF}$ for all cases. However, $E(K)$ depends on the workload characteristics and the raw error rate λ, and, in general, cannot be analytically derived. Nevertheless, with certain assumptions, we show that we can derive $E(K)$ to be 1/AVF, validating the AVF step for cases where the assumptions hold. We then show counter-examples where these assumptions do not hold, and the MTTF derived from first principles is significantly different from the MTTF derived from the AVF equation 1.

3.1.1 AVF is valid when $L \cdot \lambda \rightarrow 0$

We first show that if the product of the raw error rate and the program loop size is very small, then $E(K) = \frac{1}{AVF}$ (and so the AVF equation holds). Below we show that in this case, any of the L cycles in the program loop are equally vulnerable to a raw error event occurrence. From this, it will follow that the expected value of K (i.e., the count of the first raw error event that is not masked) is the same as 1/AVF.

Let T be the cycle count at which the next raw error event occurs. Then, without loss of generality, $T \ mod \ L$ is the cycle count for this event relative to the start of the loop iteration. Appendix A of an extended version of this paper [7] shows that if $L \cdot \lambda \rightarrow 0$, the random variable $T \ mod \ L$ follows a uniform distribution over $[0, L]$. In other words, for very small $L \cdot \lambda$, any of the L cycles of program execution are equally vulnerable to a raw error event occurrence.

Thus, the probability that the next raw error event occurs at cycle i (relative to the start of the loop iteration) is $1/L$. Let p_i be the probability that a raw error event that occurs at cycle i (relative to the start of the loop iteration) is masked (p_i is 0 or 1 for a given program execution). Therefore, the probability that the next raw error event is masked is $\sum_{i=1}^{L} \frac{1}{L} \cdot p_i$. This value is a constant that we denote by M.

Now to calculate $E(K)$, we first calculate P{K=k}. This is the probability that the first $k-1$ raw error events are masked and the kth raw error event is not masked. Since raw error events are independent, it follows that P{K=k} = $M^{k-1}(1-M)$. That is, K is a geometrically distributed random variable and so $E(K) = 1/(1-M)$. Thus, we just need to show that $1 - M$ is the same as the AVF.

$(1-M)$ can be expressed as $\sum_{i=1}^{L} \frac{1-p_i}{L}$. $1 - p_i$ is the probability that a raw error event at cycle i will not be masked and will cause failure. $1 - M$ is therefore the average of this probability over the entire program length. This is exactly the definition of AVF. Thus, we have shown that the AVF equation 1 is valid when $L \cdot \lambda \rightarrow 0$.

3.1.2 AVF is not valid for some values of λ and L

In this section, we construct a simple (synthetic) program that serves as a counter-example to show that the assumptions behind the AVF step do not always hold.

Consider a program with an infinite loop with iteration size L, such that the considered system component is active for the first A cycles and is idle for the remaining $A+1$ to L cycles of the iteration. As before, let X be the random variable denoting the time to failure for the component running the above program. Let T be the random variable denoting the time to the first raw error event. If T is in cycles $[0, A], [L, L+A], ...$, then the component is active and the time to failure is simply the value of T. Otherwise, the raw error occurs in an idle period, say, of iteration k, and it is masked. Further, any raw errors until the next active period (i.e., until cycle kL) will also be masked.

As seen at cycle kL, the distribution for the time to the next raw error event (starting from kL) is the same as that starting from time 0. This is due to the memoryless property of the exponential distribution.[1] Further, as seen from kL, the masking process is also the same as at time 0, since all iterations are identical. Thus, given that there is no failure until cycle kL, the expected time to failure from cycle kL is again E(X).

It follows that given that the first raw error event occurs in the idle period of the kth iteration, the expected time to failure

[1] Recall that for an exponential distribution, $P(T < t + \triangle t | T > t) = \frac{(e^{-\lambda t} - e^{-\lambda(t+\triangle t)})}{e^{-\lambda t}} = 1 - e^{-\lambda \triangle t}$. That is, given that a raw error has not occurred at time t, the probability that the error will occur within some time $\triangle t$ after t is the same as that of it occurring within $\triangle t$ after time 0.

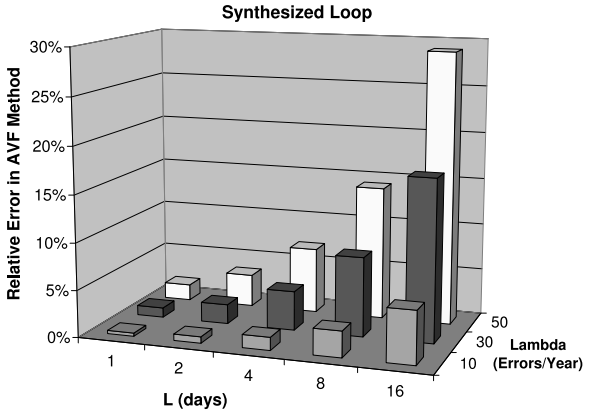


Figure 3. The relative error in the AVF step applied to a large 100MB cache running a loop with iteration size of L days with each iteration busy for $L/2$ days and idle for the rest. Lambda is the raw error rate of the entire cache (the smallest value represents 0.001 FIT per bit).

is $kL + E(X)$. Now using a standard result for conditional expectation [11], we get the following:

$$E(X) = E(E(X|T)) = \int_0^\infty E_{X|T}(t) \cdot f_T(t)dt$$
$$= \int_0^A \lambda e^{-\lambda t} t dt + \int_A^L \lambda e^{-\lambda t}(L + E(X))dt +$$
$$\int_L^{L+A} \lambda e^{-\lambda t} t dt + \int_{L+A}^{2L} \lambda e^{-\lambda t}(2L + E(X))dt ...$$

The above equation has the following closed form solution (Appendix A of [7]), giving the MTTF of the component from first principles:

$$E(X) = \frac{1-e^{-\lambda L}}{1-e^{-\lambda A}} \cdot \left(\frac{Le^{-\lambda L}}{(1-e^{-\lambda L})^2} - \frac{Le^{-\lambda A}e^{-\lambda L}}{(1-e^{-\lambda L})^2} - \frac{Ae^{-\lambda A}}{(1-e^{-\lambda L})} + \right.$$
$$\left. \frac{1}{\lambda}\frac{(1-e^{-\lambda A})}{(1-e^{-\lambda L})} + L\frac{e^{-\lambda A}-e^{-\lambda L}}{(1-e^{-\lambda L})^2}\right)$$

The AVF for our program is $\frac{A}{L}$; therefore, the MTTF according to the AVF method is:

$$E_{AVF}(X) = \frac{L}{A} \cdot \frac{1}{\lambda}$$

Now we can calculate the relative difference between the MTTF from first principles and from the AVF method as:

$$\frac{|E_{AVF}(X) - E(X)|}{E(X)}$$

When λL is very small, we can show that the two MTTFs converge to the same value. For other cases, there can be a significant difference. Figure 3 shows the difference between the two MTTF values for a 100MB cache for different values of L and λ. We vary L from 1 to 16 days, setting A as $L/2$ in each case. We start with λ at 10^{-8} errors/year per bit (0.001 FIT/bit) [6] which translates to 10 errors/year for the full cache. We additionally show results for λ of 3 and 5 times this value to represent changes in technology and altitude. Although the errors are small for the baseline (smallest) value of λ, they can be significant for higher values. Later sections perform a more systematic experimental exploration of the full parameter space.

3.2 The SOFR Step: MTTF for Multiple Functional and/or Storage Units

The SOFR step derives the MTTF of a system using the MTTFs of its individual components, as shown in equations 2 and 3. As discussed in Section 2.3, it assumes that for each component, the time to failure follows an exponential distribution with a constant failure rate (in conjunction with the AVF step, this rate is the product of the component's raw error rate and AVF). We next explore the validity of this assumption, given that each component sees significant architectural masking.

Again, the validity of the assumption depends on the values of the component's raw error rate λ and the program loop size L. Sections 3.2.1 and 3.2.2 respectively discuss cases for which the assumption is and is not valid.

3.2.1 SOFR is valid when $L \cdot \lambda \to 0$

We show that if $L \cdot \lambda \to 0$ for a component, then the time to failure, X, for that component is exponentially distributed with rate parameter $\lambda \cdot AVF$.

Section 3.1.1 showed that in this case, $X = \sum_{i=1}^{K} t_i$, where K follows a geometric distribution with mean $1/AVF$ and the t_i's are exponentially distributed with rate λ. We can calculate the density function of X as follows:

$$f_X(x) = \lim_{\triangle x \to 0} \frac{P(x < X < x + \triangle x)}{\triangle x}$$
$$= \lim_{\triangle x \to 0} \sum_{i=1}^{\infty} \frac{P(x < X < x + \triangle x | K = i) P(K = i)}{\triangle x}$$

where $P(x < X < x + \triangle x | K = k) = P(x < \sum_{j=1}^{k} t_j < x + \triangle x)$.

$\sum_{j=1}^{k} t_j$ is the sum of several independent exponentially distributed random variables with rate λ. Such a sum follows the Erlang-n distribution which has the probability density function of $\frac{\lambda(\lambda x)^{n-1}}{(n-1)!}e^{-\lambda x}$ [13]. Thus,

$$f_X(x) = \sum_{i=1}^{\infty} ((1 - AVF)^{i-1}(AVF)\frac{\lambda(\lambda x)^{i-1}}{(i-1)!}e^{-\lambda x}$$
$$= (AVF)\lambda e^{-\lambda x} \sum_{i=1}^{\infty} \frac{((1-AVF)\lambda x)^{i-1}}{(i-1)!}$$
$$= \lambda(AVF)e^{-\lambda(AVF)x}$$

This is an exponential distribution with rate $\lambda \cdot AVF$. This validates the assumption for the SOFR step for the case when $\lambda \cdot L$ is small.

3.2.2 The general case for λ and L values

In general, it is difficult to analytically characterize the time to failure distribution function for real (or even synthetic) programs after architectural masking. In this section, to demonstrate a mathematical basis, we choose a distribution that is "close" to exponential (and mathematically tractable) and determine the validity of using SOFR on that distribution.

We choose the following probability density function for the time to failure (after architectural masking) for a component.

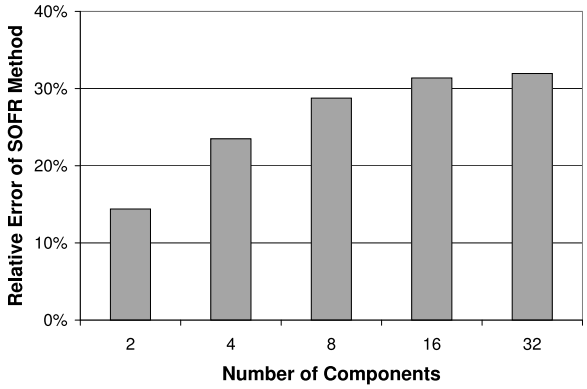


Figure 4. The relative error introduced by the SOFR step for a synthesized example.

$$f_X(x) = \begin{cases} \frac{2}{\sqrt{\pi}} e^{-x^2} & x \in [0, \infty] \\ 0 & \text{elsewhere} \end{cases}$$

The cumulative distribution function (CDF) of X is $F_X(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$, $x \in [0, \infty]$.

It follows that the MTTF of the component is $E(X) = \frac{2}{\sqrt{\pi}} \int_0^\infty x e^{-x^2} dx = \frac{1}{\sqrt{\pi}}$.

Assume a system with N such identical components where X_i denotes the time to failure for component i. Since we assume series failure, it follows that the time to failure of the system, Y, is $\min(X_1, X_2, ..., X_N)$.

The CDF of Y is $F_Y(y) = 1 - (1 - F_X(y))^N$.

The PDF is $f_Y(y) = \frac{dF_Y(y)}{dy} = N*(1-F_X(y))^{N-1} * f_X(y)$

The MTTF of the system is $E(Y) = \int_0^\infty f_Y(y) y dy$

The above integration cannot be calculated analytically. We solve it numerically using a software package to derive the real MTTF for N from 2 to 32.

The SOFR step calculates the MTTF of the system using Equations 2 and 3. For the component MTTFs used in the equations, we use the real MTTF derived above ($\frac{1}{\sqrt{\pi}}$):

$$MTTF_{sofr} = \frac{1}{\sum_{i=1}^{N} \sqrt{\pi}} = \frac{1}{N\sqrt{\pi}}$$

Figure 4 shows the error in $MTTF_{sofr}$ relative to the MTTF derived from first principles. We see that the error grows from 15% for a system with two components to about 32% for a system with 32 components.

3.3 Summary of implications

Our mathematical analysis so far provides intuition for when the AVF+SOFR method works. The AVF step averages the "utilization" of a component over the whole program. It therefore makes the implicit assumption that every point of the program will have uniform probability of being hit by a soft error. The SOFR step assumes that the time to failure for each individual component follows the exponential distribution. Our analysis shows that the above assumptions are valid when $\lambda \cdot L \rightarrow 0$. However, in the general case, these assumptions may not hold. We show mathematically tractable synthetic examples to illustrate a few such cases. The next sections provide a more systematic experimental exploration of the parameter space to assess the extent of the errors due to these assumptions.

Base Processor Parameters	
Processor frequency	2.0 GHz
Fetch/finish rate	8 per cycle
Retirement rate	1 dispatch-group (=5, max) per cycle
Functional units	2 integer, 2 FP, 2 load-store, 1 branch
Integer FU latencies	1/4/35 add/multiply/divide
FP FU latencies	5 default, 28 divide (pipelined)
Reorder buffer size	150 entries
Register file size	256 entries (80 integer, 72 FP, and various control)
Memory queue size	32 entries
iTLB	128 entries
dTLB	128 entries
Base Memory Hierarchy Parameters	
L1 Dcache	32KB, 2-way, 128-byte line
L1 Icache	64KB, 1-way, 128-byte line
L2 (Unified)	1MB, 4-way, 128-byte line
Base Contentionless Memory Latencies	
L1 Latency	1 cycles
L2 Latency	10 cycles
Main memory Latency	77 cycles

Table 1. Base POWER4-like processor configuration.

4 Experimental Methodology

This section describes the methodology for our experimental analysis of the assumptions of the AVF and SOFR steps. For each step, we first evaluate the assumptions for single processor systems common today running SPEC CPU2000 applications, and using detailed simulation to determine architectural masking. We then take a broader view, and evaluate the assumptions for a large design space, including large clusters of processors and a broader range of (synthesized) workloads, but with less detailed simulation of architectural masking.

For both cases, we first generate a *masking trace* that indicates, for each system component, whether a raw error in a given cycle would be masked for the evaluated system and workload. To calculate the real MTTF of the system (without the AVF+SOFR assumptions), we use the Monte Carlo technique to model the raw error process, apply the masking trace to the process, and determine the MTTF of the modeled system.

4.1 Today's Uniprocessors Running SPEC

To determine the impact of architectural masking in a modern processor, we study an out-of-order 8-way superscalar processor (Table 1) running programs from the SPEC CPU2000 suite (9 integer and 12 floating point benchmarks). To generate the masking trace, we use Turandot [8], a detailed trace-driven microarchitecture-level timing simulator. We simulate an instruction trace of 100 million instructions for each SPEC benchmark running on the above processor configuration.

We choose four processor components to study the impact of architecture masking: the integer, floating point, and instruction decode units, and the 256 entry register file, with raw error rates of $2.3 * 10^{-6}$, $4.5 * 10^{-6}$, $3.3 * 10^{-6}$, and $1.0 * 10^{-4}$ errors/year respectively (10^{-8} errors/year = 0.001 FIT). Li et al. [6] derived these error rates using published device error rates for current technology [12] and estimates of the number of devices of different types in different components [6].

For the integer, floating point, and instruction decode units, we assume that a raw error is masked in a cycle if the unit is not processing an instruction in that cycle (i.e., the unit is not busy). If the unit is busy processing an instruction, then for simplicity, we conservatively assume that the error is not masked and will lead to failure. For the register file, we assume that the raw error strikes happen on each register with equal probability and error in a given register is masked if the register contains a value that will never be read in the future. If the register's value will be read, we conservatively assume the error is not masked and will lead to failure. Our assumptions of when an error is not masked are conservative since it is possible that an error in an active unit or in a register value that will be read may not affect the eventual result of the program. We did not perform a more sophisticated analysis to more precisely determine when an error is masked because such an analysis is orthogonal to the point of this paper and beyond the scope of this work.

Our detailed Turandot simulation produces a masking trace for each simulated SPEC application. The trace contains information on whether a raw error in a given cycle in one of the four considered processor components will or will not be masked.

4.2 Broader Design Space Exploration

We also explore a broad design space for the AVF and SOFR steps. We consider a variety of systems consisting of various numbers of components, operating in various environments, with different raw error rates, and running different workloads. We use the term *system* to include a single processor (either a full processor or only a part of it) or a large cluster of thousands of processors. A *component* of a system is the smallest granularity at which the analysis for architectural masking is applied. Specifically, the AVF is calculated at the granularity of a component; the SOFR step then aggregates the information from the different components to give the MTTF for the entire system. In our SOFR experiments, we use component MTTFs obtained from the Monte Carlo method; therefore, the error reported is only that caused by the SOFR step.

Based on our analysis in Section 3, the key parameters affecting the AVF and SOFR steps are the raw error rate of the different components of the processor (or system), the number of components in the system (only for SOFR), and the program loop size or workload. The following discusses the space we explore for each of these important parameters. Table 2 summarizes this space.

Component raw error rate. The component raw error rate depends on the number of devices or elements (bits of on-chip storage or logic elements such as gates) in the component and the raw error rate per element. We denote the number of elements in a component as N. N can be as large as 10^9 for large cache structures or if we consider the entire processor as one component in a large cluster of multiple processors. To keep the design space exploration tractable, without loss of generality, we assume that all N elements have the same raw error rate.

Dimension	Value				
N	10^5	10^6	10^7	10^8	10^9
S	1	5	100	2000	5000
C	2	8	5000	50000	500000
Workload	SPEC fp	SPEC int	day	week	combined

Table 2. The design space explored. N = number of elements (e.g., bits) in a component; S = scaling factor for the baseline raw error rate of an element (depends on technology and altitude); and C = number of components in the system (e.g., processors in a cluster).

We also explore different values for the raw error rate per element. Under current technology, the terrestrial raw error rate per bit for on-chip storage is about 10^{-8} errors/year (0.001 FIT), which we refer to as the baseline raw error rate. To account for changes in the raw error rate due to technology scaling and at high altitudes, we introduce a parameter S that we use to scale the above baseline rate. We use scaling factors of 1, 5, 100, 2,000, and 5,000 in our analysis. The larger factors correspond to systems running in airplanes flying at a high altitude and for systems in outer space because of strong radiation at those heights [16]. Test systems using accelerated conditions are also subject to high raw error rates.

The raw error rate for a given component is determined as the product of N, S, and the above baseline raw error rate (Table 2).

Number of components: We denote the number of components in the system as C. We study a wide range of values for C, ranging from 2 to $500,000$. The larger numbers represent large cluster systems with C components (each of which may be a full processor or a microarchitectural component within a processor, depending on the granularity at which AVF is collected).

Workload and generation of the masking traces: We evaluate all systems in the broad design space with the SPEC CPU2000 benchmarks mentioned in Section 4.1. However, these are short programs (small loop iteration size L). Many real world workloads show large differences in behavior over long time scales (large L) that are difficult to capture with the SPEC benchmarks. In an attempt to simulate some of the behaviors of real world applications, we construct three synthetic applications. The first (called *day*) is a continuous loop where the loop iteration size is set to 24 hours. The loop is busy during the day (half the time) and idle at night. The second (called *week*) is a loop with iteration size one week. It is busy during the five business days of the week and idle for the weekend. The third (called *combined*) concatenates two SPEC benchmarks in

a loop with iteration size of 24 hours. The first half of the iteration runs one benchmark and the second half runs the other benchmark.

For a system with multiple processors, we assume all processors run the same workload. Additionally, for the synthesized workloads, we assume that a component is a full processor; e.g., C=2 implies a 2 processor system. We assume that each processor masks raw errors only during the idle portion of the workload (e.g., night time for the day workload). For the SPEC workloads, we again assume that each component is a full processor (running the same benchmark). For the masking trace, we use the SPEC masking traces for three units in each processor (integer, floating point, and instruction decode) – we apply these three traces to the corresponding units simultaneously to determine whether there is a processor-level failure.

4.3 Monte Carlo Simulation

To calculate the real MTTF, we perform Monte Carlo simulation where we do the following for each trial. For each component in the modeled system, we generate a value from an exponential distribution with rate specified by the modeled system (Table 2). This value gives the arrival time of the next raw error event for the component. We use the masking trace of the workload to determine whether a raw error at that time would be masked. If it is masked, we generate a new raw error event from an independent exponential distribution for that component and repeat. If it is not masked, we consider the component failed. The component that is earliest to fail gives the time to failure of the system for this trial. We run a total of 1,000,000 trials and report the average of the time to failure as the MTTF of the modeled system/workload configuration.

5 Results

5.1 AVF and SOFR with Today's Uniprocessors Running SPEC

We first evaluated the discrepancy between the Monte Carlo MTTF and the MTTF using the AVF and SOFR steps for today's uniprocessors running SPEC (as described in Section 4.1). We found that the MTTF from the AVF step matched the Monte Carlo MTTF very well for each of the four processor components and each benchmark ($< 0.5\%$ discrepancy for all cases). Similarly, the processor MTTF calculated using the SOFR step also matched the Monte Carlo MTTF very well.

Thus, for single processor systems with a small number of small components running SPEC benchmarks, the AVF+SOFR method works very well. We note that in prior work, the method has been applied primarily in this context. These results are consistent with our mathematical analysis. The loop size L for the SPEC benchmarks and the component raw error rates used here are small; therefore, from Sections 3.1.1 and 3.2.1, we expect that the AVF and SOFR assumptions would be valid.

5.2 AVF: A Broad Design Space View

For the design space described in Table 2, we computed the component MTTF using the Monte Carlo and AVF methods as described in Section 4. Note that since the AVF step is applicable to only a single component, C = 1 for all experiments in this section. Further, for a given workload, since only the product of N and S matters, we report relative error in the AVF step as a function of different values of $N \times S$.

We found that for each SPEC benchmark, the AVF step works well for all N and S values studied (relative error $<$ 0.5%). However, for the longer running synthesized workloads, we observe significant discrepancy when $N \times S$ is large (i.e., component raw error rate is large). Figure 5 shows the error in the AVF MTTF relative to the Monte Carlo MTTF for representative values of $N \times S$ for the three synthesized workloads. In all three cases, for $N \times S \geq 10^9$, the AVF step sees significant errors (up to 90%). This high value of $N \times S$ may occur when the AVF step is applied to either large components (e.g., a 125MB cache with $N = 10^9$ bits), or when the component size is moderate but the raw error rate per element (bit) is high (e.g., $S = 1000$ because of high radiation at high altitudes).

Our experiments show both positive and negative errors, depending on the workload. Thus, AVF may either over- or under-estimate MTTF in practice.

Again, the above observations match well with our theoretical analysis in Section 3.1. Thus, for SPEC like benchmarks that run for a short time, it is safe to use the AVF step to calculate the MTTF of a component. However, the AVF step must be applied carefully when using a workload with large variations over large time scales coupled with either a large component or a large per-element raw error rate for the component.

5.3 SOFR: A Broad Design Space View

Figures 6(a) and (b) report the error in the SOFR step relative to the Monte Carlo method for three representative SPEC benchmarks and the three synthesized benchmarks respectively. For each case, the error is reported for representative values of C and $N \times S$ covered by the design space in Table 2.

Focusing on the SPEC workloads (Figure 6(a)), we see that the SOFR step is accurate for systems with a small number of components (C = 2 or 8) for all studied values of $N \times S$. When system size grows to 5,000 components or larger, we see significant errors, but only with very large values of $N \times S$. For example, for a cluster of 5,000 processors with each processor containing $N = 10^9$ bits of on-chip storage, the baseline raw error rate would need to scale 2,000 times or more to see a significant error. In practice, terrestrial systems will likely fall into the part of the design space where the SOFR step does not introduce any significant error for SPEC applications.

Focusing on the synthesized workloads (Figure 6(b)), for the day workload, we see a significant error using the SOFR step when $N \times S \geq 10^8$ and $C \geq$ 5,000. The error increases as these parameters increase. For example, with 12.5MB of storage for each processor ($N = 10^8$) and baseline raw error rate ($S =$

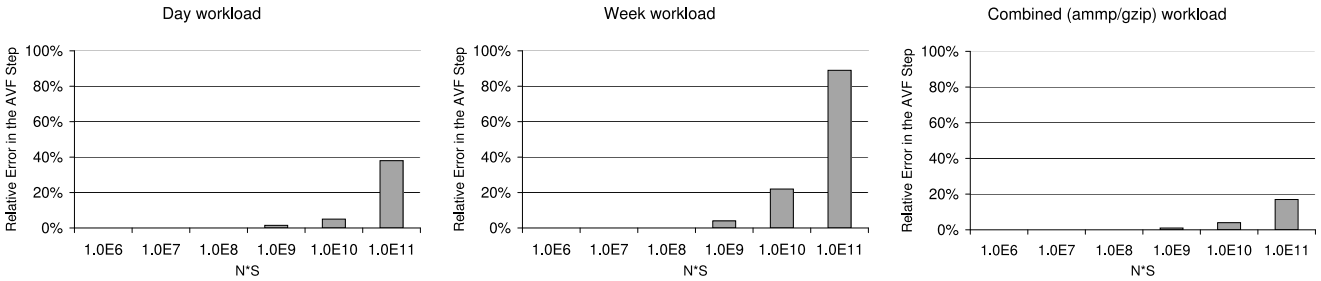
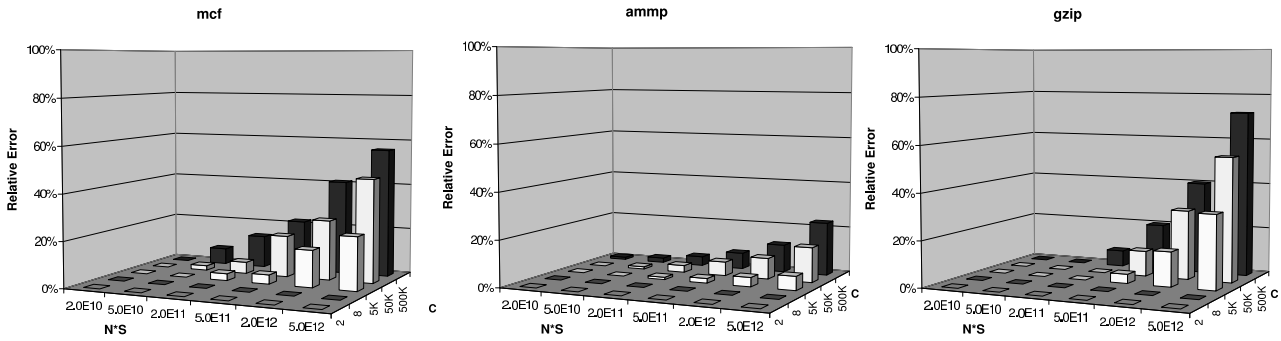
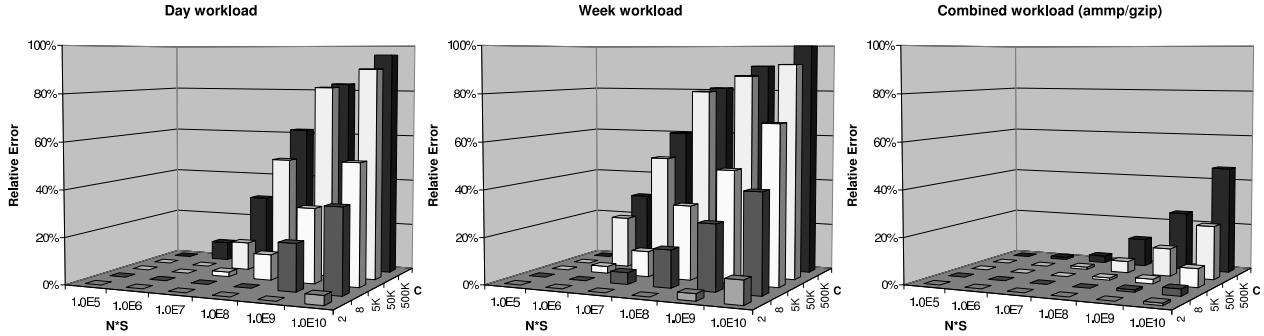


Figure 5. Error in MTTF from the AVF step relative to the Monte Carlo method for the synthesized workloads for representative values of $N \times S$ (# bits in the component $\times$ scaling factor for baseline raw error rate).



(a) SPEC benchmarks



(b) Synthesized benchmarks

Figure 6. Error in MTTF from the SOFR step relative to the Monte Carlo method for representative values of C (# components) and $N \times S$ (bits per component $\times$ scaling factor for baseline raw error rate) for (a) SPEC and (b) synthesized benchmarks.

1), a 5,000 processor cluster sees an error in MTTF of 11%. For a similar cluster of 50,000 processors, the error jumps to 50%. While large, such a cluster is not unrealistic. For the week workload, since the loop size is larger than the day workload, the MTTF errors are correspondingly larger. Thus, the 5,000 and 50,000 processor systems mentioned above respectively see MTTF errors of 32% and 80% for this workload. With larger processors (more storage bits) or larger systems, the error can grow to 90% or more. Thus, for these workloads, the SOFR step incurs significant errors for realistic systems.

Finally, the combined workload (with two SPEC applications) shows a relative error smaller than for the day or week workload, but there is still a significant error for some cases.

In summary, for SPEC benchmarks under current technology and on the ground, the SOFR step gives accurate MTTF estimates. However, in general, for larger scale workloads, care must be taken to examine the workload behavior, number of system components (e.g., processors), and the raw error rate for the components (governed by component size and per-bit or per-element error rate) before applying SOFR.

5.4 The SoftArch Method

SoftArch is an alternate architecture-level model to calculate processor MTTF due to soft errors [6]. In conjunction with a microarchitectural timing simulator, SoftArch keeps track of the probability of error in each instruction or data bit that is generated or communicated by different processor structures. A bit may be erroneous if it is struck by a high energy particle (error generation) or if it is computed from previously erroneous bits (error propagation). SoftArch uses simple probability theory to track probabilities of these error generation and propagation events, thereby keeping track of the probability of error in any bit. As the program executes in the microarchitectural simulator, SoftArch identifies which values could affect program outcome and when (e.g., values propagated to memory on a store). Using the error probability for such values and the time they affect program output (i.e., potentially result in failure), SoftArch is able to determine the mean time to (first) failure.

SoftArch's probabilistic approach does not require the AVF and SOFR assumptions; it is therefore useful to explore whether SoftArch can be applied to the parts of the design space where AVF+SOFR shows significant discrepancies from the Monte Carlo method. We used SoftArch to estimate MTTF for the entire design space studied here. We found that for every point in this space, the error in MTTF computed by SoftArch relative to the Monte Carlo MTTF is less than 1% for a single component and less than 2% for the full system. Thus, SoftArch does not exhibit the discrepancies shown by AVF+SOFR. These results are not meant to provide a complete validation of SoftArch or a complete comparison between SoftArch and AVF+SOFR (such an analysis is outside the scope of this work). Rather, these results suggest alternative methodologies and motivate future work combining the best of existing methodologies for the most accurate MTTF projections across the widest design space.

6 Conclusions

This paper examines key assumptions behind the AVF+SOFR method for estimating the architecture-level processor MTTF due to soft errors. We use rigorous theoretical analysis backed by simulation-based experiments to systematically explore the applicability of the AVF and SOFR steps across a wide design space. Our analysis and experiments show that while both steps are valid under the terrestrial raw soft error rate values of today's technology for standard workloads (e.g., SPEC), there are cases in the design space where the assumptions of the AVF and SOFR steps do not hold. In particular, for long running workloads with large component-level utilization variations over large time scales, the assumptions are violated for systems with a large number of components and/or with high component-level raw error rate (i.e., large component size and/or large per-bit or per-element raw error rate). Under these conditions, the projected MTTF of the modeled system or chip could show large errors. In general, the paper builds better understanding about the conditions under which the standard AVF+SOFR method may be used to project MTTF accurately, and alerts users to the risks of using the model blindly in conditions where the foundational axioms of the model break down. The paper also shows that an alternative architecture-level method (that does not make the AVF+SOFR assumptions) does not show the above discrepancies in MTTF, motivating future work to determine the best combination of methodologies to provide the best estimates for all scenarios.

References

[1] A. Biswas et al. Computing the Architectural Vunerability Factor for Address-Based Structures. In *ISCA*, 2005.

[2] E. W. Czeck and D. Siewiorek. Effects of Transient Gate-level Faults on Program Behavior. In *Proc. of the International Symposium on Fault-Tolerant Computing*, 1990.

[3] T. Karnik et al. Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes. *IEEE Transactions on Dependable and Secure Computing*, 1(2):128–143, June 2004.

[4] S. Kim and A. K. Somani. Soft Error Sensitivity Characterization for Microprocessor Dependability Enhancement Strategy. In *DSN*, 2002.

[5] X. Li et al. SER Scaling Analysis of a Modern Superscalar Processor with SoftArch. In *Proc. of SELSE-1 Workshop*, 2005.

[6] X. Li et al. SoftArch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors. In *DSN*, 2005.

[7] X. Li et al. Architecture-Level Soft Error Analysis: Examining the Limits of Common Assimptions (extended version). Technical Report UIUCDCS-R-2007-2833, UIUC, March 2007. Available at http://rsim.cs.uiuc.edu/Pubs/07dsn-tr.pdf.

[8] M. Moudgill et al. Validation of Turandot, a Fast Processor Model for Microarchitecture Evaluation. In *International Performance, Computing and Communication Conference*, 1999.

[9] S. Mukherjee et al. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor. In *MICRO*, 2003.

[10] H. T. Nguyen and Y. Yagil. A Systematic Approach to SER Estimation and Solutions. In *Proc. of IRPS-41*, 2003.

[11] S. Ross. *A First Course in Probability* (Chapter 7). Prentice Hall, 2001.

[12] P. Shivakumar et al. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In *DSN*, 2002.

[13] K. Trivedi. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Prentice Hall, 1982.

[14] N. Wang et al. Characterizing the Effects of Transient Faults on a Modern High-Performance Processor Pipeline. In *DSN*, 2004.

[15] C. Weaver et al. Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor. In *ISCA*, 2004.

[16] J. F. Ziegler. Terrestrial Cosmic Rays. *IBM Journal of Research and Development*, 40(1):19–39, 1996.