Sampling + DMR: Practical and Low-overhead Permanent Fault Detection

Shuou Nomura   Matthew D. Sinclair   Chen-Han Ho   Venkatraman Govindaraju   Marc de Kruijf
Karthikeyan Sankaralingam

Vertical Research Group
University of Wisconsin – Madison
{nomura,sinclair,chen-han,venkatra,dekruijf,karu}@cs.wisc.edu

ABSTRACT

With technology scaling, manufacture-time and in-field permanent faults are becoming a fundamental problem. Multi-core architectures with spares can tolerate them by detecting and isolating faulty cores, but the required fault detection coverage becomes effectively 100% as the number of permanent faults increases. Dual-modular redundancy(DMR) can provide 100% coverage without assuming device-level fault models, but its overhead is excessive.

In this paper, we explore a simple and low-overhead mechanism we call Sampling-DMR: run in DMR mode for a small percentage (1% of the time for example) of each periodic execution window (5 million cycles for example). Although Sampling-DMR can leave some errors undetected, we argue the permanent fault coverage is 100% because it can detect all faults eventually. Sampling-DMR thus introduces a system paradigm of restricting all permanent faults' effects to small finite windows of error occurrence.

We prove an ultimate upper bound exists on total missed errors and develop a probabilistic model to analyze the distribution of the number of undetected errors and detection latency. The model is validated using full gate-level fault injection experiments for an actual processor running full application software. Sampling-DMR outperforms conventional techniques in terms of fault coverage, sustains similar detection latency guarantees, and limits energy and performance overheads to less than 2%.

Categories and Subject Descriptors: C.4 [Computer Systems Organization] Performance of Systems — Fault Tolerance; C.0 [Computer Systems Organization] General — System architectures

General Terms: Design, Reliability, Performance

Keywords: Fault tolerance, Permanent Fault, Dual-modular redundancy, Sampling, Reliability

1. INTRODUCTION

Device physics, manufacturing, and process scaling engineering are providing significant challenges in producing reliable transistors for future technologies. Many academic experts, industry consortia, and research panels have warned that future generations of silicon technology are likely to be much less reliable with devices likely failing in the field due to permanent faults in silicon [8, 2, 17, 36, 21, 1, 9]. These include manufacturing faults that escape testing [25] and faults that appear during the chip's lifetime [42, 15]. A fault is an anomalous physical condition caused by a manufacturing problem, fatigue, etc.

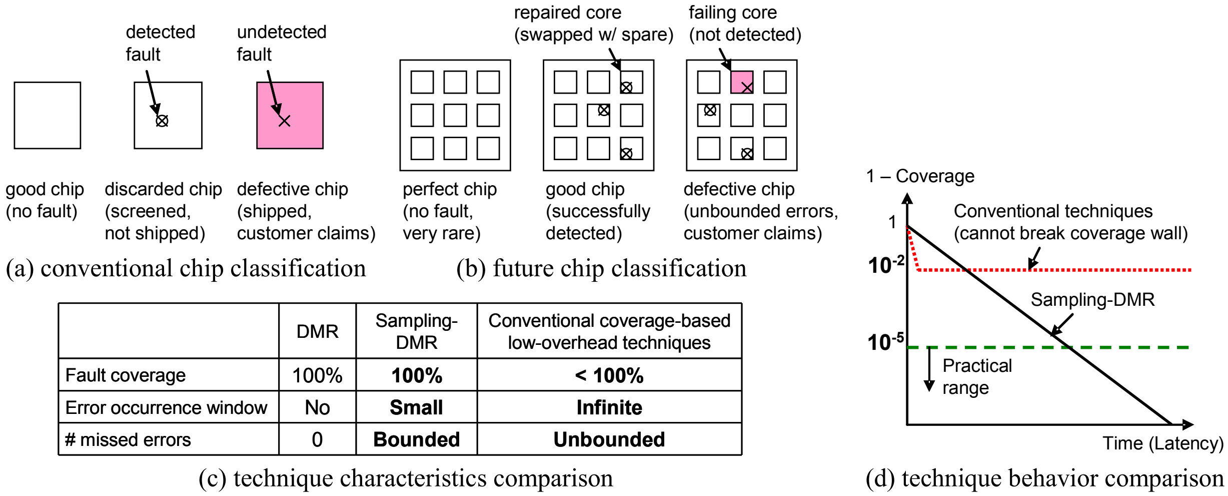
The ITRS Roadmap 2009 edition [2] predicts: *"the ultimate nanoscale device will have high degree of variation and high percentage of non-functional devices right from the start."* and *"Ultimately, circuits that can dynamically reconfigure themselves to avoid failing (or to improve functionality)...will be needed."* To address this problem, a paradigm of logic redundancy with spare cores (or finer-granularity units) on a chip is being embraced. Figures 1(a) and 1(b) contrast this paradigm with the conventional approach where a single fault in a chip renders it defective. In the future, since permanent faults will be numerous, *a defective chip is one with an undetected fault* with detected permanent faults "repaired" by swapping in spare units. Architecture research in this paradigm can be classified along three directions: detection [21, 10, 23, 35, 26, 5], repair [4, 14, 43, 18, 36, 17, 44], and recovery [31, 39].

Efficient and accurate permanent fault detection is required and is the focus of this paper. We address this looming fault detection crisis, identify inefficiencies in the state-of-art detection, and advance the field by significantly improving detection accuracy with little performance overheads ($\leq 5\%$).

What is currently lacking? A fundamental drawback of almost all prior detection approaches is they trade off *fault coverage* (what percentage of faults can be detected) for overhead (area, energy, performance, detection latency etc.). While scan-test or built-in self test (BIST) do provide 100% coverage, they are restricted to stuck-at-faults and thus not a general solution. Using other low-overhead techniques, coverage in the 99% range is common for stuck-at-faults [10, 26, 21] and in the range of 95% for timing faults [30]. Unfortunately, product quality in terms of defective chips increases exponentially with the number of faults. In Section 2, we develop a simple and general model for detemining effective defect rates. As a hypothetical example, consider a 100-core chip, with 10 permanent faults on average budgeted per chip. We show that, we require 99.999% coverage for practical defect rates and 99% coverage results in 10% defective chips. For finer-granularity architectures like GPUs with many spares, defect-rate problems due to low coverage are exacerbated. As shown in Figure 1(d), these techniques cannot break the coverage wall i.e. regardless of how long the detection mechanisms runs some faults remain uncovered and undetectable. *Thus, we argue that* $<100\%$ *fault-coverage is unsuitable for fault-dominated future technology nodes.*

Complete dual-modular redundancy (DMR) detects *architectural*

ISCA'11, June 4–8, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0472-6/11/06 ...$10.00.



(a) conventional chip classification

(b) future chip classification

	DMR	Sampling-DMR	Conventional coverage-based low-overhead techniques
Fault coverage	100%	**100%**	**< 100%**
Error occurrence window	No	**Small**	**Infinite**
# missed errors	0	**Bounded**	**Unbounded**

(c) technique characteristics comparison

(d) technique behavior comparison

Figure 1: Comparison of chip classification and fault detection techniques

errors using a redundant module and provides 100% fault coverage[1]. An error is the effect on the architectural state of a processor due to activation of a fault. However, DMR can sometimes suffer from design complexity [13, 28, 35], and more importantly area and energy overheads that exceed 100%.

Sampling-DMR: In this paper, we propose a simple idea we call Sampling-DMR. Instead of running in DMR mode all the time, with sampling, DMR mode is active only a fraction of the time. Our mechanism is driven by two related observations.

1. *When a permanent fault generates errors frequently, sampling can detect the fault immediately.*
2. *When a fault generates errors infrequently, the number of missed (undetected) errors before Sampling-DMR detects them is low.*

Sampling-DMR provides four key benefits:

1. *100% Fault-detection:* Although the error detection is probabilistic, Sampling-DMR can eventually detect all permanent faults, and hence the coverage is 100%.
2. *Small energy overhead:* By reducing the DMR period to a small fraction, the energy overheads are drastically reduced.
3. *Small area overhead:* Conventional DMR techniques effectively have at least 100% area overhead, because half the resources are for checking. With sampling, these resources are used for checking only a small fraction of the time.
4. *Low design complexity:* Sampling relaxes performance requirements of the implementation. Prior DMR implementations suffer from design complexity resulting from the significant processor pipeline modifications and design optimizations to minimize DMR slowdowns ([13] provides a good overview). With sampling on the other hand, massive slowdowns in DMR mode can be tolerated. For 1%-DMR (sampling 1% of the time), even massive slowdowns of 4X in the DMR period result in only 3% overall slowdown. Hence, even a slow but simple FIFO-based implementation is sufficient as we show in Section 4.

Is this really better than conventional fault detection? Figures 1(c) and 1(d) compare fault detection schemes to Sampling-DMR. Recall that Sampling-DMR works by detecting errors. Fault detection approaches (with typically <100% coverage) allow *undetected and unbounded errors*. When an uncovered fault occurs, the errors it causes will keep occurring in the hardware, thus making them unbounded and the error occurrence window is infinite. Sampling-DMR, on the other hand, provides 100% fault coverage. While it has some probability of missing *errors*, which are thus undetected errors, the number of these undetected errors is bounded, and they are restricted to a finite window of occurrence (which is typically milliseconds). Below, we elaborate on other subtle details on fault detection approaches.

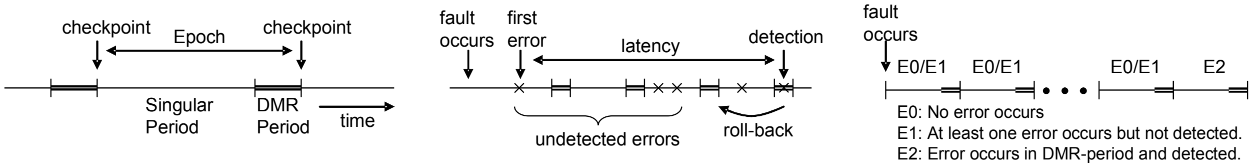
For systems using periodic scan-test with test vectors [10], 100% coverage is feasible, but only for single stuck-at-faults. Even so, undetected errors occur between a fault's excitation to detection and the occurrence window can be tens of seconds, as described in detail in Section 5.4. In addition, there are many types of permanent faults that are not covered by this model (such as bridging, transition, path-delay, and cross-talk faults). The occurrence of such faults is increasing as technology scales, and achieving 100% fault coverage is much harder for these models. For example, in the path-delay fault model, each gate can have additional delay independently. The problem is the number of paths is exponential in the size of circuit (in our experiment, a 16-bit multiplier has 160 billion paths). Achieving high coverage is difficult because one test vector can test only few paths at once. Hence, we can test only some critical paths, but the increase of process variations and small delay defects due to technology scaling significantly increases the number of critical path candidates.

Furthermore, the fixed-test-vector approach based on a specific fault model suffers from unmodeled faults. In addition to 100% fault coverage, it must be proven that the model covers 100% of the actual possible faults. While historically challenging and difficult, the use of non-matured new materials and device structures exacerbates the problem because collecting sufficient data on manufacture defects and wear-out faults takes a long time.

Sampling-DMR on the other hand makes no assumptions about a device-level fault model. Its only assumption is faults are permanent. It operates by detecting architectural errors thus achieving 100% fault coverage, and restricts any errors to a finite window.

Regardless of the detection mechanism, undetected errors and a non-zero error occurrence window is a reality all future systems

[1] DMR's fault coverage is 100%, under the assumption that the probability of the same fault occurring on the two modules at the same time is negligible.



(a) basic concept (b) probabilistic detection (c) mathematical model

Figure 2: Sampling-DMR

must address. Sampling-DMR improves upon the state-of-art by providing 100% coverage and provides formal analysis for the number of undetected errors and detection latency window.

Contributions: This paper makes the following four contributions.

1. *Motivation and Concept:* First, we show 100% coverage for permanent fault detection will become necessary in future technologies. We then introduce the idea of sampling-based error-detection and show it can provide 100% coverage and is a practical low-overhead solution. (Section 2)

2. *Mathematical model of Sampling-DMR:* We develop mathematical models, which are validated, to understand the probabilistic behavior of Sampling-DMR. For a given product quality requirement, the models provide estimations on the number of missed errors and detection latency. (Section 3)

3. *Implementation:* We show an example FIFO-based DMR implementation which leaves the processor's pipeline practically unmodified. It highlights sampling's benefits: low design complexity and tolerance to DMR slowdowns. (Section 4)

4. *Evaluation and effectiveness:* We performed full gate-level fault injection experiments for the OpenRISC processor and the OpenSPARC FPU. Our results show that 1%-DMR and 5%-DMR with simple checkpointing result in correct execution for 96% and 99% of faults. Based on the empirical error sequence data applied to our model, we show 1%-DMR has comparable performance to conventional periodic scan-test in terms of detection latency while providing 100% fault coverage. (Section 5)

2. OVERVIEW OF SAMPLING-DMR

In this section, we first present a simple model to motivate 100% permanent fault coverage. We then present an overview of the Sampling-DMR concept and discuss its system-wide implications.

2.1 Why is 100% fault coverage necessary?

The paradigm of logic-redundancy and repair removes the need for increasing margins in processes, devices, or circuits to accommodate scaling challenges. It allows some number of permanent faults but requires architectural support for reliability and accurate fault detection. As shown in Figure 1(b), in future multicore processors, a defective chip is one with an undetected permanent fault. Intuitively, since permanent faults will produce errors through a chip's lifetime, detecting all of them (i.e. 100% fault coverage) is desirable. We now formulate the fault coverage requirements using a simple mathematical analysis. We assume random fault distribution. Consider a multi-core chip, designed to allow certain number of permanent faults (num_of_faults), for which the designer provisions that many spares. If we use a conventional fault detection technique with per-core fault-coverage of $coverage$, the number of defective chips in terms of defective rate (or probability) is given by:

$$chip\_defect\_rate = (1 - coverage^{num\_of\_faults})$$

As an example, consider a chip designed with 10 spares to allow 10 in-field failures. Even 99% fault coverage results in a defect-rate of 0.0956. This means, out of a million chips, 95,618 will have faults that cannot be detected and although the spares are available, the detection mechanism fails. To achieve 100 defective parts-per-million (10^{-4} defect-rate), which is a typical industry goal, the necessary fault coverage is 99.999%. As the number of faults increases, required coverage increases further. *Thus, practically 100% fault coverage is required for future technologies.*

2.2 Sampling-DMR

Redundant execution in different components with result comparison is one approach for error detection, referred to as dual-modular redundancy (DMR). It can provide 100% fault coverage, but the overhead is excessive. Our approach stems from the following question:

"*Since, permanent fault occurrence is a rare event, do we need to run the redundant execution all the time?*"

Sampled redundant execution also can provide 100% fault coverage because it can eventually detect all faults, since permanent faults continue generating errors. This idea can be implemented in several ways. For example, MapReduce and GPU/CUDA-like programming models allow a software-only implementation by only replicating sampled threads and comparing results. In this paper, we provide a general approach – hardware-based redundant execution for multicore processors.

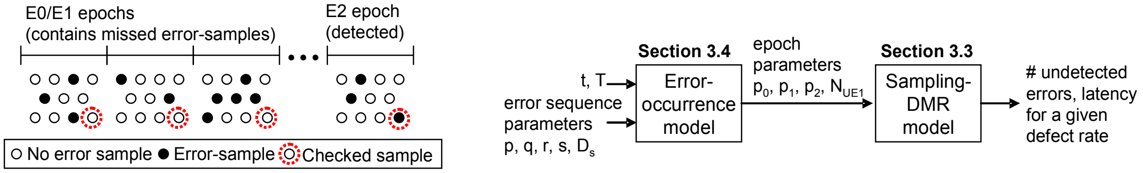
Figure 2(a) shows how Sampling-DMR works. Within an epoch of a group of instructions, a small fraction (N%) at the end executes in DMR mode - we refer to this as N%-DMR (N = 100 is equivalent to conventional DMR). The architectural state of the checked core is first transferred to the checker core at the beginning of every DMR period, and then DMR execution starts. We take checkpoints at the beginning of every non-DMR period. When a fault occurs, it will start producing errors. When an error occurs during a DMR period, it is detected and the system rolls back to the previous checkpoint. We assume a previously proposed efficient checkpointing mechanism like Revive [31] or SafetyNet [39] is used. Since the checkpointing requires buffering of I/O transactions, the length of checkpoint is practically limited to around 5 million cycles.

To determine which core is faulty, detailed diagnosis such as full-DMR execution with another core from the last checkpoint is initiated, and the process is migrated elsewhere.

Sampling-DMR's effectiveness must be considered for two scenarios.

1. If the *first* error occurrence epoch has an error during DMR-period, the checkpoint recovers the system to a clean state.

2. If errors occur during a non-DMR period of an epoch but do not occur during the DMR-period of the epoch, we have "un-


E0/E1 epochs
(contains missed error-samples)
E2 epoch
(detected)
○ No error sample ● Error-sample Checked sample
t, T
error sequence parameters
p, q, r, s, D_s
Section 3.4
Error-occurrence model
epoch parameters
p_0, p_1, p_2, N_{UE1}
Section 3.3
Sampling-DMR model
# undetected errors, latency for a given defect rate

(a) upperbound analysis

(b) model overview (input and output)

Figure 3: Theoretical analysis overview

detected errors” as shown in Figure 2(b). Since the checkpoint is corrupted, we cannot perfectly recover.

At first glance, it seems this probability of case (1) is small and the approach sounds impractical for real systems. An investigation of the mathematical probabilities, coupled with actual error sequences of real processors and applications, show that even 1%-DMR is practically sufficient. When error-rates are high, faults are detected immediately. When error-rates are low, the number of undetected errors is also very small and we observe a bound on this number and the latency to eventually detect it. *Our key contributions are the Sampling-DMR principle, its theoretical model, and experimental results showing its practical effectiveness.*

And what about...

1. *Seg-faults and timeouts?* Certain faults can cause the application to cause a segmentation fault or result in endless loops. For early detection, extensions can be done to the operating system.
2. *But errors occur in phases, how can this work?* Intuitively, we expect errors due to a fault to occur in clusters and not regularly spaced in time. Any such burst behavior and burst length has to be *exactly correlated to the sampling frequency and epoch length* to hinder detection. This is unlikely and our detailed empirical results validate this.
3. *What about transient faults like soft-errors?* Soft errors are an orthogonal problem to manufacturing defects or wear-out faults. Circuit-level techniques like the BISER latch [27] provide a good low-overhead solution. So, like other related work [10, 23], we restrict our scope to permanent faults.
4. *How can sampling with application code match test vector coverage?* Periodic testing with test vectors may detect faults more efficiently than Sampling-DMR, since vectors are designed to consume minimum testing time. However, 100% coverage for all fault-models is unlikely and hard as described earlier. Intuitively, the Sampling principle works because of the following. If faults are not excited, they don’t produce errors, and hence do not matter. When faults get excited, they will eventually be detected. Testing, on other hand, must provide 100% coverage at test-generation time.

2.3 Can applications accept undetected errors?

While Sampling-DMR provides 100% fault coverage which is good from a defect-rate perspective, it has some probability of missing errors. An additional guarantee is that these missed or undetected errors are restricted to occur in a finite window. We discuss the system-wide implications of this phenomena.

First, undetected error phases are rare events. The number of times such undetected error phases occur is small. For a system with 10 spares for example, these undetected error windows occur 10 times in the chip’s lifetime, since the manufacturer will likely budget number of spares accounting for expected failures.

Second, eventual detection latency is typically short. Furthermore, our quantitative results show that, with 1%-DMR, these undetected errors will appear only for a short time before being detected: <1 second for 97% of faults and <78 seconds for 99.999% of faults as shown in Figure 10(b).

Third, the zero undetected errors requirement may be unnecessary. Quantitative evidence exists for undetected errors in several systems. Schroeder have shown that uncorrectable errors occur in DRAMs in deployed systems [33]. Bairavasundaram et al. have shown 2.45% of disks have latent errors [6]. Google has demonstrated that 1.3% of memories have unrecoverable bit errors [34]. Reacting to this fact, higher-level lightweight application-level assertions are common. In environments like game consoles and desktops, emerging RMS applications can mask errors using redundancy in algorithms [22]. In mobile devices, users are likely to accept few seconds of downtime (due to errors) in exchange for more battery life. Some have claimed the scenarios where errors are acceptable are increasing and proposed systems with explicitly relaxed reliability [45, 11] or budgeting for a controlled amount of errors [12]. Sridharan et al. present a comprehensive taxonomy and a framework for software recovery that allows hardware errors [40].

So overall, we believe undetected errors can be and must be handled with a full-system view in a paradigm of fault-dominated technologies. *Sampling-DMR can achieve zero defective chips and provide bounded detection latency guarantees while state-of-art detection techniques hit a coverage wall and allow unbounded errors.*

3. THEORETICAL ANALYSIS

In this Section, we present a mathematical analysis on the behavior and effectiveness of Sampling-DMR. After defining terms in Section 3.1, in Section 3.2 we analyze the upper bound on undetected errors as depicted in Figure 3(a). In Sections 3.3 and 3.4, we develop a model, whose output is a distribution of *how many chips* are perfectly detected i.e. for all faults on these chip, there are no undetected errors. And in the remaining *how many undetected errors exist* and *the latency to detect*. Figure 3(b) shows an overview. The inputs to the model are error sequence parameters which consider empirical error occurrence patterns to account for user and application behavior.

3.1 Definition of terms

The model uses four inputs: i) required product quality defined as a defect-rate (D_R), ii) non-DMR period length (T), iii) DMR period length (t), and iv) error sequence parameters.

Since Sampling-DMR’s behavior is probabilistic, we define a defect rate D_R as the probability that a metric (e.g. latency, the number of undetected errors) is worse than the required value. We explain this for undetected errors (UEs). From the stand point of worst-case estimation, for a given D_R, we obtain n_e such that the probability that the number of UEs exceeds n_e is D_R. Here, n_e is the maximum value (i.e. worst-case) for the defect rate of D_R. For chip-level projection, the number of chips that will face a *worse situation than this “worst-case” (i.e. the number of UEs >n_e)* is less than $D_R \times$ total number of chips. Hence, the defect rate D_R corresponds to the conventional product quality in terms of specification guarantee.

We classify epochs under the three categories described below and shown in Figure 2(c):

E_0: No error occurred,
E_1: At least one error occurred but not detected,
E_2: Error occurred and detected.

3.2 Upper bound on undetected errors

First, we present an upper bound analysis on the number of undetected errors, regardless of error occurrence patterns. Figure 3(a) shows the model for analysis. Each epoch has a fixed number of samples (L) and some samples are error-samples. For each epoch, the sampling scheme picks up one sample and checks if it is an error-sample or not. If it is not an error-sample, sampling continues and does this for the next epoch. This scheme detects an error-sample at some point eventually. Here, the question is how many error-samples can be missed when considering a given defect rate D_R. The upper bound is: $-L\log_e D_R$. The proof is as follows.

For Sampling-DMR, one sample is the continuous execution period t cycles in duration, and the number of samples L is $\frac{T+t}{t}$. The assumption here is error occurrence is independent of when DMR-period starts.

Statement: For any distribution of error-samples across any number of epochs, if the total number of error-samples (U) is $-L\log_e D_R$, the probability that sampling cannot detect error-samples (i.e. mis-detection) across the epochs(S) is always less than D_R.

PROOF. Let n be the number of epochs, and let e_k be the number of error-samples for epoch k ($k = 1..n$), and let s_k be the probability of mis-detection for epoch k. This can we written as ($1 -$ detection probability). Therefore, $s_k = 1 - e_k/L$, and $e_k = L(1 - s_k)$. The mis-detection probability across epochs is then, $S = s_1 s_2 s_3 ... s_n = \prod_{k=1}^{n} s_k$.

$$U = \sum_{k=1}^{n} e_k = \sum_{k=1}^{n} L(1 - s_k) = L(n - \sum_{k=1}^{n}(s_k))$$

Since arithmetic mean $\geq$ geometrical mean,

$$U \leq L(n - n(\prod_{k=1}^{n} s_k)^{\frac{1}{n}}) \leq Ln(1 - S^{\frac{1}{n}}).$$

By applying the lemma described below to the right term of above equation, $U \leq Ln(1-S^{\frac{1}{n}}) < -L\log_e S$. Since $U = -L\log_e D_R$ as assumed in the statement, $-L\log_e D_R < -L\log_e S$. Therefore, $S < D_R$. □

Why does the statement provide an upper bound? The probability that sampling cannot detect $-L\log_e D_R$ error-samples is equivalent to the probability that the number of undetected error-samples exceeds $-L\log_e D_R$, and this corresponds to the definition of defect rate. The statement claims such probability is always less than D_R. Since the worst-case number of undetected error-samples for D_R increases as D_R decreases, the worst-case number of undetected error-samples for the defect rate D_R is less than $-L\log_e D_R$. It is the upper bound and is one of key findings: *there is a mathematical upper bound on the missed errors which is only dependent on the DMR ratio and the required defect rate.*

Lemma: $n(1 - p^{\frac{1}{n}}) < -\log_e p$ **(if** $p < 1$**)**

PROOF. We consider the sequence $a_n = n(1 - p^{\frac{1}{n}}) \quad (n = 1, 2, ...)$, and this sequence is strictly increasing. We claim $\lim_{n\to\infty} a_n = -\log_e p$. Let x be $\frac{1}{n}$. To show this claim, it suffices to show that $\lim_{x\to 0} \frac{1-p^x}{x} = -\log_e p$. We use l'Hopital's rule: $\lim_{x\to 0} \frac{g(x)}{h(x)} = \frac{\lim_{x\to 0} g'(x)}{\lim_{x\to 0} h'(x)}$. The derivative of the numerator is $-p^x \log_e p$ and the derivative of the denominator is 1. Therefore, $\lim_{x\to 0} \frac{1-p^x}{x} = \lim_{x\to 0} -p^x \log_e p = -\log_e p$. □

3.3 Sampling-DMR model

The previous section provides only an upper bound. In this section, we develop the probability distribution function for the number of undetected errors and latency. Let p_0, p_1, p_2 be the respective occurrence probability of E_0, E_1, E_2 epochs when any fault is excited. Recall definitions of epochs from Section 3.1 and Figure 2(c). Let N_{UE_1} be the expected number of undetected errors in E_1 epoch. In the next section, we model error occurrence behavior to determine p_0, p_1, p_2, and N_{UE_1} .

The number of undetected errors is controlled by "How many E_1 epochs before E_2 epoch?", which we establish first. "The probability that the number of E_1 epochs is exactly n" is the same as "the probability that an E_2 epoch occurs after n E_1 epochs; with any number of interspersed E_0 epochs". By using conditional probability, the probability that a non-E_0 epoch is E_1 epoch is $\frac{p_1}{1-p_0}$. The probability that a non-E_0 epoch is E_2 epoch is $\frac{p_2}{1-p_0}$. Hence,

$$\text{Probability of exactly } n \text{ } E_1 \text{ epochs} = (\frac{p_1}{1-p_0})^n \times (\frac{p_2}{1-p_0})$$

We want to determine the probability that the number of E_1 epochs (errors) exceeds n; we define a function $\text{cp}(n)$ for this. It gives the probability that E_1 occurs $n+1$ times continuously in non-E_0 epochs. Hence,

$$\text{Probability of greater than } n \text{ } E_1 \text{ epochs: } \text{cp}(n) = (\frac{p_1}{1-p_0})^{n+1}$$

The probability of greater than n_e undetected errors is approximated as:

$$\text{Probability of greater than } n_e \text{ errors: } \text{cp}(\frac{n_e}{N_{UE_1}}) \qquad (1)$$

Latency is represented by the number of non-E_2 epochs from the first E_1 epoch. Similarly to above, for $n > 0$,

$$\text{Probability of exactly } n \text{ latency: } \frac{p_1}{1-p_0}(1-p_2)^{n-1} p_2$$

$$\text{Probability of greater than } n \text{ latency: } \frac{p_1}{1-p_0}(1-p_2)^{n} \qquad (2)$$

3.4 Error occurrence model

To put the model to use for real systems, we must answer the question: "what is p_0, p_1, p_2, and N_{UE_1} ?" This strongly depends on the epoch length, the DMR period length, and the error sequence. The purpose of the error-occurrence model is to simplify the error sequence for analysis by representing it with few parameters. We developed four models with increasing sophistication.

Constant-rate model: This is the simplest model with one parameter. We assume that faults generate errors at a constant per-cycle probability of p. The corresponding probabilities are as follows:

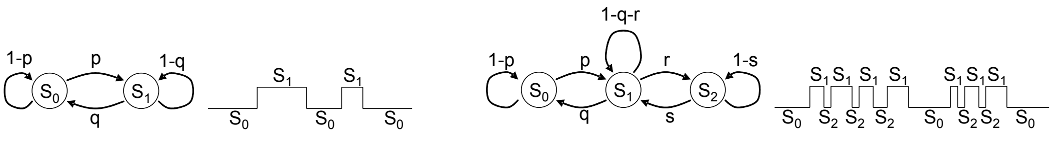
$$p_0 = (1-p)^{T+t}, \quad p_2 = 1 - (1-p)^t, \quad p_1 = 1 - p_0 - p_2$$

The expected number of undetected errors in E_1 epochs is:

$$N_{UE_1} = \frac{\sum_{j=1}^{T} j \cdot \binom{T}{j} \cdot p^j (1-p)^{T-j}}{1-(1-p)^T} = \frac{pT}{1-(1-p)^T}$$

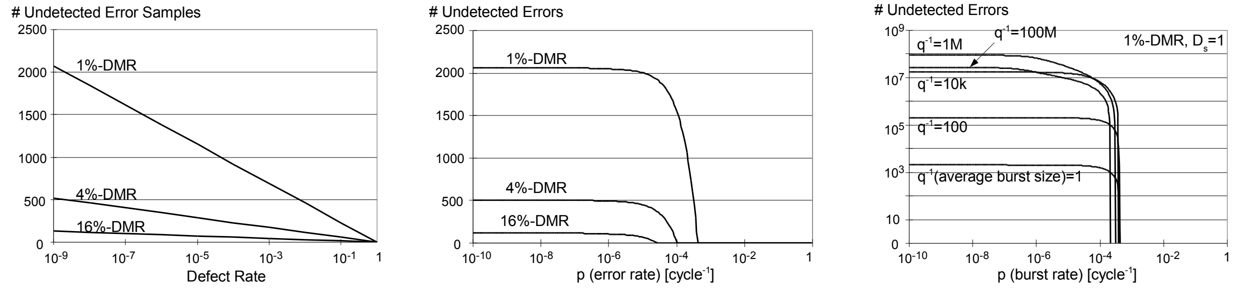
In this equation, the numerator of the fraction represents the average number of errors in a non-DMR period. The denominator represents the probability that a non-DMR period contains at least one error.

Discretization: The limitation of constant-rate model is that it cannot represent bursts. Using coarse time-unit allows to mask the effect of short burst. For example, if an error occurs at a constant rate but continues during 10 cycles once it occurs, considering only the average per-cycle probability makes the detection probability optimistic. In this case, a time-unit of 1000 cycles masks this short-burst effect. The model has the per time-unit error probability p and


1-p
p
1-q
S_0
S_1
q
S_1
S_1
S_0
S_0
S_0
1-q-r
1-p
p
r
1-s
S_0
S_1
S_2
q
s
S_1 S_1 S_1 S_1 S_1 S_1 S_1
S_0 S_2 S_2 S_2 S_0 S_2 S_2 S_0

(a) 2-state HMM (b) 3-state HMM

Figure 4: Burst error-occurrence model using HMM



(a) upperbound analysis (b) constant-rate model (c) 2-state HMM (worst-case 3-state HMM)

Figure 5: Analysis results on worst-case undetected errors (epoch size: 5 million cycles)

the average number of errors in a time-unit D_s. The parameters are

$$p_0 = (1-p)^{T+t}, \qquad p_2 = 1-(1-p)^t,$$

$$p_1 = 1 - p_0 - p_2, \qquad N_{UE_1} = D_s \frac{pT}{1-(1-p)^T}$$

Burst model using 2-state HMM: The limitation of discretization is that it cannot represent bursts longer than the time unit. We introduce a 2-state Hidden-Markov-Model (HMM) to represent the hysteresis of error occurrences. Figure 4(a) shows the model. It has two states, namely S_0 and S_1. Errors occur when the state is S_1, and no error occurs when the state is S_0. The transition probability of S_0 to S_1 is *p*, and that of S_1 to S_0 is *q*.

Here, let s_0 and s_1, respectively, be the limit probability that a state is S_0 and S_1 after infinite time passed. The state transition probability equation is

$$\begin{pmatrix} s_0 \\ s_1 \end{pmatrix} = \begin{pmatrix} 1-p & q \\ p & 1-q \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \end{pmatrix}$$

Since $s_0 + s_1 = 1$,

$$s_0 = \frac{q}{p+q}, \; s_1 = \frac{p}{p+q}$$

The probability that an epoch is E_0 is that the state is S_0 at the first cycle and does not change *during the epoch*. The probability that a *DMR-period* does not contain any errors is the probability that a state is S_0 at the fist cycle and does not change *during the DMR-period*. The probability that an epoch is E_2 is 1 − this previous probability, and N_{UE_1} is calculated by using s_0 and s_1 as before. Hence,

$$p_0 = s_0(1-p)^{(T+t-1)}, \qquad p_2 = 1 - s_0(1-p)^{(t-1)},$$

$$p_1 = 1 - p_0 - p_2, \qquad N_{UE_1} = D_s \frac{s_1 T}{1 - s_0(1-p)^{(T-1)}}$$

Burst model using 3-state HMM: The limitation of the 2-state HMM is that it cannot represent short glitches in burst period. Figure 4(b) shows the 3-state HMM model, in which we introduce the third state S_2 as the short-glitch state. Similar to the two-state model, we define the state transition probability *p*, *q*, *r*, *s* as shown in the figure. Here, let s_0, s_1, and s_2, respectively, be the limit probability that a state is S_0, S_1, and S_2 after infinite time. The state transition probability equation is

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 1-p & q & 0 \\ p & 1-q-r & s \\ 0 & r & 1-s \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix}$$

Since $s_0 + s_1 + s_2 = 1$,

$$s_0 = \frac{qs}{ps+pr+qs}, \; s_1 = \frac{ps}{ps+pr+qs}, \; s_2 = \frac{pr}{ps+pr+qs}$$

Similar to the 2-state model, but considering S_2 state as the same as S_0 state, the parameters are

$$p_0 = s_0(1-p)^{(T+t-1)} + s_2(1-s)^{(T+t-1)},$$

$$p_2 = 1 - s_0(1-p)^{(t-1)} - s_2(1-s)^{(t-1)},$$

$$p_1 = 1 - p_0 - p_2,$$

$$N_{UE_1} = D_s \frac{s_1 T}{1 - s_0(1-p)^{(T-1)} - s_2(1-s)^{(T-1)}}$$

3.5 Model results

Based on these models, we show the upper bound on undetected errors (UE). We consider the defect-rate of $D_R = 10^{-9}$ (1 defective chips in a billion), which is practically a zero defects guarantee.

Regardless of error occurrence patterns, there is an upper bound on the number of undetected error *samples*, which is determined only by the DMR ratio. It is 2072 for 1%-DMR as shown in Figure 5(a).

When errors occur at a constant rate, the maximum number of UEs is 2062 for 1%-DMR as shown in Figure 5(b). If the error-rate *p* is high, the number of UEs is zero because the first DMR period always detects them. For 1%-DMR and 5 million epoch, the threshold error-rate for zero UEs is 0.0004 as the figure shows.

When error occurrence shows burst behavior, the maximum number of UEs increases as the epoch size and the burst size increases. Figure 5(c) shows the result of 2-state HMM. The maximum number of UEs is about 100 million when the burst size is 1 million cycles occurring at a low-rate. This matches the upper bound with worst-case number of errors in samples (i.e. 50K errors/sample × 2072 samples = 104 million errors), and this can be considered as an unrealistic worst-case.

In summary, the burst effect determines the number of UEs. In Section 5, we analyze empirical relationship between latency and undetected errors to confirm the actual impact of this burst effect.

4. IMPLEMENTATION

We now present one example to show a simple Sampling-DMR implementation. Normal DMR techniques have high area over-

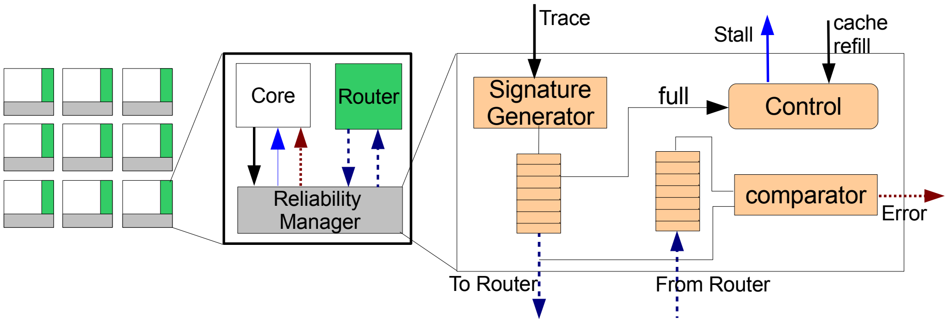


Figure 6: FIFO-based DMR Implementation

heads and sometimes complex core modifications to minimize performance degradation due to synchronization and result comparison of two redundant modules. Complex core modifications are especially problematic because these introduce uncovered faults. For example, CRT [13] utilizes a load-data transfer mechanism from checked core to checker core, rendering the entire data cache controller uncovered. To detect permanent faults, sampling can be applied on top of these conventional DMR implementations like CRT, Fingerprinting [38], DCC [19], thus reducing their effective energy and overheads. A second benefit is that sampling effectively provides more cores for useful computation, because with conventional DMR, implicitly half the cores simply act as checkers and do not perform useful computation. While these techniques with full DMR provide permanent *and* transient fault coverage, with sampling, a technique like the BISER latch [27] is necessary to provide good transient fault coverage.

Sampling-DMR's basic property that performance degradation in DMR mode (even 2X) is not a concern, because DMR execution period is limited to a small fraction, provides an opportunity to investigate simple, yet efficient designs. We can focus on addressing design complexity, area overhead, and keeping core modifications to a minimum.

4.1 FIFO-based implementation

Figure 6 shows our FIFO-based implementation. We assume a many-core processor with shared L2 caches interconnected with a mesh network. For each core, we add a DMR control module named reliability manager (RM), which consists of small control circuits and two shallow FIFO buffers, and we use the existing on-chip networks for trace data transfer. The area of RMs is negligibly small compared to cores/networks, and the core modifications are limited to the following mechanisms: i) generating per-instruction output trace, ii) cache refill trigger signal, iii) stall inputs signal, and iv) transferring of the architectural state between cores.

In the checked core, the RM receives architectural state updates (at commit-time) from the core and writes it into the sender FIFO. Typically the state update information is register name/value pairs and store address/value pairs. The RM interfaces with the L2-cache network and encodes the ID of the checker core (Section 4.3) with the FIFO data to send messages to the checker core. To reduce this inter-core communication we use fingerprinting [38].

In the checker core, RM receives messages from the coupled checked core into a receiver FIFO. The checker core's RM writes its architectural state updates (redundant execution) into its sender FIFO. A comparator compares elements in the two FIFOs. Any time they differ, the RM raises a DMR error exception. To determine which core is faulty, detailed diagnosis such as full-DMR execution with another core from the last checkpoint is initiated, and the application process is migrated elsewhere. The checker core should monitor dirty eviction signals in the cache and synchronize the checker core with the checked core to avoid memory incoherence between cores as described in Section 4.2.

We assume the network enforces in-order delivery of messages and we use the interconnection network's flow control mechanism to automatically throttle the checker's FIFO if it is executing at a faster rate than the checked core. The control-path in the RM includes a simple state-machine that must stall the processor when the FIFOs become full and interface the flow-control signals from the router with the FIFOs.

4.2 Common challenges for DMR implementations

In contrast to full-DMR and redundant multi-threading [28] solutions, Sampling-DMR tolerates radical performance degradations because DMR is active only for a small fraction of time. We exploit this tolerance to slowdowns to provide simple solutions for the following common problems.

Memory incoherence: Stores to memory and load/store ordering have been the main challenges for previous redundancy-based techniques. Like Reunion [37], we allow either core to get ahead and allow both to read/write memory. Stores pose a small problem. If both cores are allowed to write back dirty cache-lines asynchronously, there is potential for an earlier load-miss from the trailing core getting this "new" data. To avoid this problem, the cores are synchronized on any dirty-evictions, which is simple to implement: the checker core is stalled until the store is received from the checked core. This occurs only infrequently and even if done frequently, Sampling-DMR can tolerate large slowdowns.

For shared-memory programs, there are load-store ordering issues between different real threads. A store to an address can execute in a different thread in the time gap between when loads to that address execute on the checker and checked core. This can lead to input incoherence. It can be solved by executing Reunion's re-execution protocol which introduces complications in single-stepping the processor. LaFrieda et al. suggest this overhead can become untenable for some types of core coupling considering the fingerprint comparison overheads [19]. Their age table can be incorporated into the RM. *Because we can tolerate large slowdowns, a simpler implementation is sufficient: synchronize both cores on all cache refills.*

Microarchitectural state difference: In this design the checker core is not a microarchitectural mirror because its branch-prediction tables, dependence predictor tables and other speculative states are not copied over. Thus faults in these structures are not covered. However, by design these structures cannot affect architectural state and hence do not affect correctness, only performance: the branch predictor stuck-at taken for example. Such microarchitecture state can be included in the initialization of the checker and these signals can be included in the trace sent to the RM. It may introduce fur-

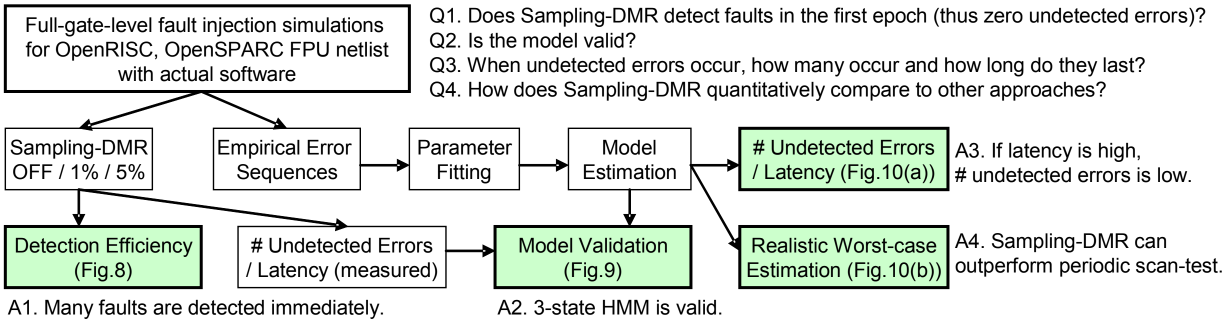

Full-gate-level fault injection simulations for OpenRISC, OpenSPARC FPU netlist with actual software
Q1. Does Sampling-DMR detect faults in the first epoch (thus zero undetected errors)?
Q2. Is the model valid?
Q3. When undetected errors occur, how many occur and how long do they last?
Q4. How does Sampling-DMR quantitatively compare to other approaches?
Sampling-DMR OFF / 1% / 5%
Empirical Error Sequences
Parameter Fitting
Model Estimation
# Undetected Errors / Latency (Fig.10(a))
A3. If latency is high, # undetected errors is low.
Detection Efficiency (Fig.8)
# Undetected Errors / Latency (measured)
Model Validation (Fig.9)
Realistic Worst-case Estimation (Fig.10(b))
A4. Sampling-DMR can outperform periodic scan-test.
A1. Many faults are detected immediately.
A2. 3-state HMM is valid.

Figure 7: Evaluation overview (questions and answers)

ther slowdowns or increase energy, *but sampling is more forgiving of any slowdowns which is a key difference from prior proposals.*

4.3 Mode transition management

The specific challenge for Sampling-DMR is how to manage the difference of required cores between DMR period (requires 2 cores) and non-DMR period (requires 1 core). While, this can be handled by adding extensions to the OS and its scheduler, we describe one design below which avoids system software modifications. We assume the chip exposes a fixed number of virtual CPUs (VCPUs) to the OS as proposed in Mixed-Mode Multicore (MMM) system [45].

A thin firmware VM layer manages Sampling-DMR operation as follows. Every VCPU is mapped to two physical cores (VCPU pair) by the firmware. Each physical core has four modes, namely, checker mode, checked mode, non-DMR mode, and free mode. The firmware starts a process by marking one core to be in non-DMR mode and the other is marked available (free mode). When entering DMR mode, the firmware activates the checker core (by finding a core in free mode) and changes its mode to the checker mode. It then copies the entire architecture state (registers and cache-lines) from the checked core to checker core (using some microcode). It executes in DMR mode for the DMR period length. If an error occurs in DMR mode, the RM triggers a DMR error exception. If no error occurs, the checked core's mode is changed to non-DMR, and the checker core is marked free. The key benefit of the firmware VM layer is that it allows the VCPUs to be arbitrarily paused and allows quick transition in and out of DMR mode.

5. EVALUATION

While our formal model provides analysis based on some error sequence parameters, we examine empirical behavior in this Section. Figure 7 shows the overview of our evaluation framework and the key questions we investigate. We use detailed gate-level simulation and full-system performance simulators to provide a comprehensive evaluation. Readers may skip ahead to the final key result in Section 5.4 which compares Sampling-DMR to conventional techniques as shown in Figure 10(b).

5.1 Q1: Does Sampling-DMR work in real settings?

Method: The epoch size is set to 5 million cycles, and we examine DMR-ratio of 1% and 5%. Programs start with DMR period. We consider the following two simulations.
1) OpenRISC: We synthesize the OpenRISC [29] RTL code and inject faults on every logic gate output. This framework provides the most detailed results, but can execute only simple programs because of library and operating system limitations. The applications we evaluate are: H.264 decoder (33 Mcycles), G.721 decoder (6 Mcycles), and JPEG decoder (6 Mcycles). We use the Synopsys 90nm library for synthesis and use emulation with the Virtex-5 FPGA for acceleration. The fault model is stuck-at 0/1 and slow-to-rise/fall faults. The number of experiments is 21,372 (fault sites) * 4 (fault types) * 3 (applications) = 256,564.
2) OpenSPARC FPU: We also consider gate-level fault injection for large complex applications. We consider the PARSEC [7] and SPECCPU [41] suites. We built a hybrid framework which simulates and does fault injection for the functional units alone at the gate-level and the rest of program executes a native x86 instruction stream. We use a synthesized gate-net of the OpenSPARC FPU (which consists of three pipelines and supports 21 instructions) and use binary instrumentation to invoke gate-level simulation for FP instructions. Since this co-simulation is 100,000-times slower, we must limit the duration of fault injections. This duration was determined manually such that, fault injection's result did not change when the fault injection duration was increased further. The fault model is stuck-at 0/1 faults. The number of experiments is 23,128(fault sites) * 2(fault types) * 23(applications) = 1,063,888. We assume 1GHz operation and instruction per cycle of 1. *This experiment is chosen as a stress-test of Sampling-DMR because FPU errors provide low error rates and various types of burst and phase behavior.*
Q1: Does it work? *Yes, Sampling-DMR detects all faults eventually. In addition, many faults are immediately detected. With 5%-DMR, 99% of faults result in error-free execution when considering the full processor. Stress test of FPU shows 95% faults result in correct execution (no error in application results).*
Details: Figure 8 shows the effectiveness of Sampling-DMR, which we study using established fault classification terminology. We compare every fault's effect with and without Sampling-DMR detection. The first row shows results with No-DMR, and the second and third rows show 1%-DMR and 5%-DMR respectively. The result of faults on applications is classified into the following five categories: i) Architecture-masked: faults that do not cause any architectural errors, ii) Application-masked: architectural errors occur, but are masked because of natural error-correction or redundancy in application and its output is unmodified, iii) Timeout: faults causing endless loops, iv) Segmentation fault: faults causing segmentation faults, and v) SDC: all other faults result in silent data corruption.

Our results show that, for the OpenRISC processor, 96% and 99% of faults result in error-free (Architecture-masked) execution for 1%-DMR and 5%-DMR, respectively. Note this is not to be interpreted as a fault-coverage number. They are the percentage of faults resulting in error-free execution when considering recovery with 5ms checkpoint interval. This shows detection latency is typically low. *All permanent faults are eventually detected, thus delivering 100% coverage.*

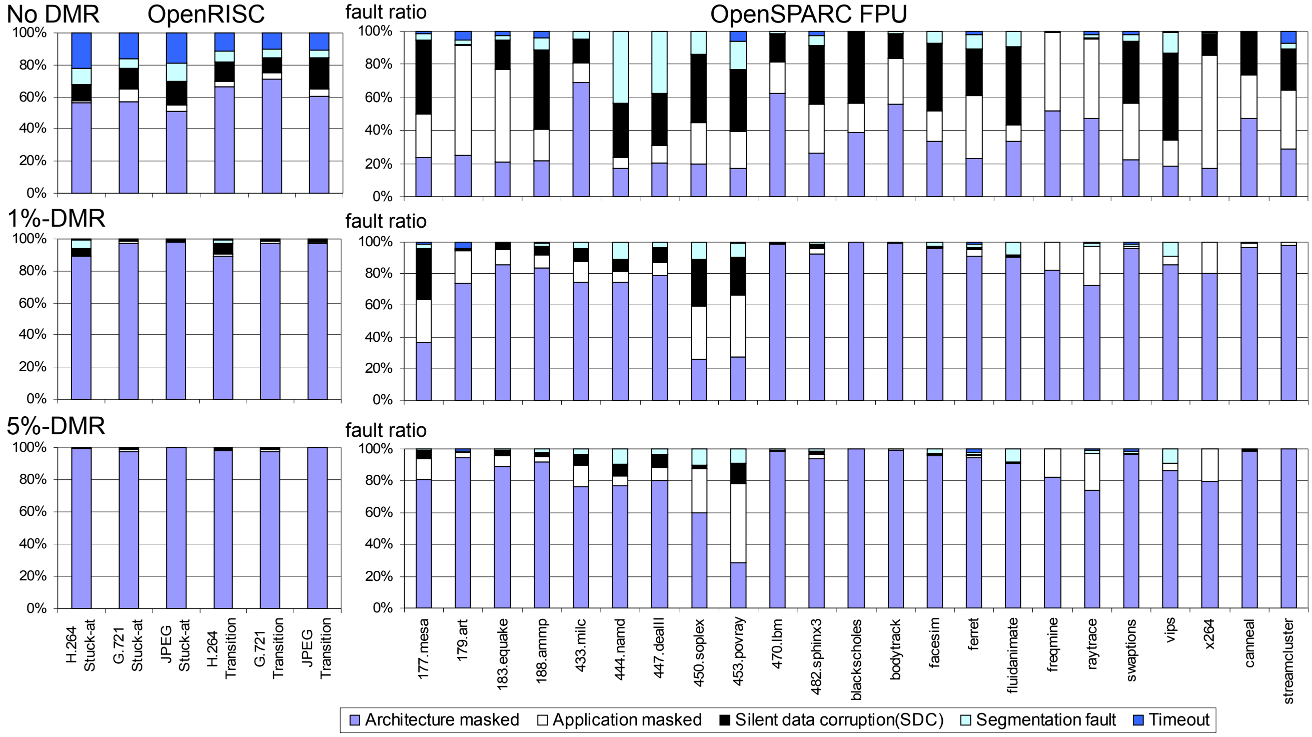


Figure 8: Application behavior when considering recovery with 5ms checkpoint interval (not final coverage)

For the OpenSPARC FPU also Sampling-DMR sustains a high percentage of architecture-masked faults i.e. no undetected errors. Overall, the OpenSPARC FPU results are worse than the OpenRISC results. This is because faults in OpenRISC are more likely excited than that in the FPU. *Hence, for the rest of the evaluation, we use these FPU results to stress Sampling-DMR, thus making our chip-level defect-rate projections conservative.*

Examining the cases of faults which result in some undetected errors, we can see that Sampling-DMR's restricting of error occurrence to a small window, is something applications seem to be able to tolerate. This is evidenced by the fact that, the ratio of application-masked faults also increases for some applications (450.soplex and 453.povray), when comparing No-DMR to 1%-DMR and 5%-DMR.

5.2 Q2: Is the model valid?

Method: The model parameters are derived from the error sequences obtained in the OpenSPARC FPU experiment in Section 5.1. We consider error sequences from all experiments in which Sampling-DMR detects fault but misses some errors. The obtained error sequences are discretized into 1K cycles time-unit and discretization is applied to every model.

The parameter fitting is as follows.

- For the *discretized constant-error-rate model*, the error-rate parameter p is obtained by simply dividing the number of error-time-units (time-units with errors) by the number of total time-units. Error density D_s is the average number of errors in error-time-units.

- For the *2-state HMM*, p is obtained by counting the number of transitions from no-error-time-unit to error-time-unit and dividing by the total number of no-error-time-units. Similarly, q is number of transitions from error-time-unit to no-error-time-unit, divided by the total number of error-time-units.

- For the *3-state HMM*, it is necessary to distinguish if a non-error time-unit is for state S_0 or state S_2 to obtain the parameters of p, q, r and s. Here, we use a heuristic to determine this. The length of continuous non-error time-units determines it. If it is less than geometrical mean of the maximum and the minimum length, it is for S_2, else it is for S_0. The probability parameters are obtained similarly to the 2-state HMM.

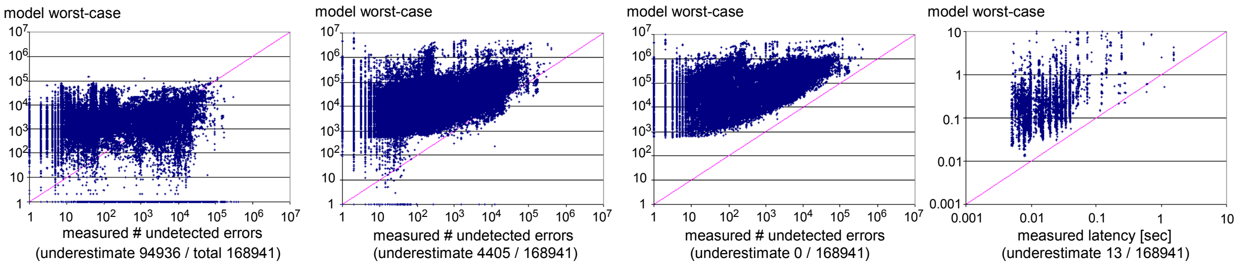
After parameter fitting, we obtain p_0, p_1, p_2 and N_{UE_1}. Then, we derive the number of undetected errors and the latency corresponding to 10^{-5} defect-rate and compare to empirical results. This is the lowest defect rate to meaningfully validate, since we have 168,941 measurements ($\frac{1}{168,941} = 5.9 * 10^{-6}$).

Q2: Is the model valid? *Yes, the Sampling-DMR model using the 3-state HMM never under-predicts the number of undetected errors and rarely under-predicts the latency.* Figure 9 plots the relationship between the actual measurement and the model worst-case estimation for 10^{-5} defect-rate. Points above the $45°$ line means the model over-predicts. We use the 3-state HMM for the rest of evaluations because it never underestimates the number of undetected errors and rarely underestimates the latency and is always within 50%.

5.3 Q3: How many undetected errors and how long do they last?

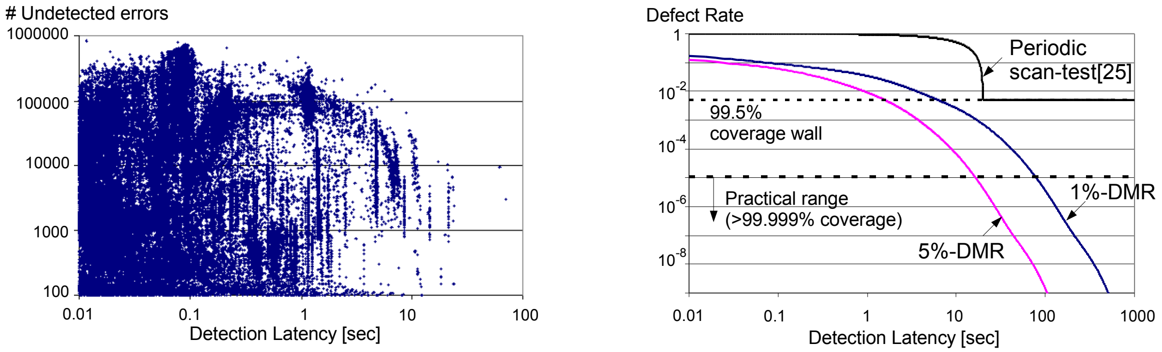
Method: We use the model parameters for 3-state HMM based on error sequences of all experiments (46,256 faults * 23 applications). For each error sequence, we derive the average number of undetected errors and the average latency by the model.

Q3: How many undetected errors and how long do they last?



(a) discretized constant-rate model (b) 2-state HMM (c) 3-state HMM (d) 3-state HMM (latency)

Figure 9: Model validation results (model worst-case estimation compared with measured data)



(a) latency and # undetected errors (b) latency and defect rate

Figure 10: Estimated behavior based on empirical data(FPU stress test) and 3-state HMM.

If latency is high, the number of undetected errors is small. Figure 10(a) plots the relationship between the detection latency and the number of undetected errors. It takes the maximum at the latency of 0.1sec and it decreases as the latency increases. *This implies that the impact of burst effect, which increases the number of undetected errors, is limited when latency is high.* This characteristic is reasonable and can be tolerated by systems. If latency is low, users can re-run the application, and hence the number of undetected errors itself is not a concern. If latency is high, errors occur sparsely in time domain like soft-errors. This is the case of application masking and as discussed by Feng et al. systems and users may naturally tolerate this [12].

5.4 Q4: Comparison to state-of-art

To compare to state-of-art, we determine the distribution of detection latency from the model. Without the model, determining the behavior of very low error-rate faults would be impossible because of simulation time slowdowns. Again, we consider the FPU results which makes our projections conservative. It is a stress test, since we consider low error-rate faults and assume the entire chip will have such faults.

Method: First, we obtain the model parameters for all experiments (46,256 faults * 23 applications). Then, for each experiment, we derive the defect rate as a function of detection latency. Next, we obtain the defect-rate across all experiments, assuming all F experiments have the same occurrence probability: $D_{R_{all}} = \sum_{j=1}^{j=F} D_{R_{experiment_j}}/F$.

Q4: Comparison to conventional techniques: *1%-DMR can outperform periodic scan-test in terms of latency and defect rate.*

Figure 10(b) shows the defect-rate that can be sustained for different ranges of the latency. Recall we are doing distributions across chips (and hence different users) and this is FPU-based stress test projections. It shows the worst-case latency for practically required 10^{-5} defect-rate is about 78 seconds and 16 seconds for 1%-DMR and 5%-DMR, respectively. Over 99% of chips see a latency of less than 3.6 seconds and 0.9 seconds respectively.

Figure 10(b) also plots the latency of periodic scan-test[23], considering their reported coverage of 99.5% and an optimistically low test time of 200ms (ignoring the 34.2 second test data transfer time). We assume 20 second test period to limit test-time overhead to 1%, and assume that the first error occurs immediately when a fault occurs. As the figure shows, 1%-DMR can outperform periodic scan-test. Considering concurrent test like SWAT and Argus, Sampling-DMR outperforms them in terms of defect rate because they cannot break the coverage wall and cannot reach the practical coverage region of >99.999%. To be clear Argus has detection latency of a few cycles on the permanent faults it does cover (and it covers transient fault), but we argue low detection latency without 100% coverage is of limited value when permanent fault rates are high as expected in future technology nodes.

5.5 Implementation/performance overhead

With sampling, slowdown in DMR mode is not a significant problem, as shown with our simple model considering: i) a slowdown factor S in DMR mode because the master copy may slow down due to synchronization overheads with the checker, and ii) T_{trans}: the delay incurred in transitioning in and out of DMR mode. With Sampling-DMR, total slowdown is: $S_{tot} = (t * S + T + T_{trans})/(T + t)$. For 5 million-cycle epochs, 2X slowdowns with 20,000 cycle transition costs, overall performance reduction is only 1.4%.

Hence we only briefly report on our simulation-based performance evaluation using the GEMS Multifacet infrastructure [24]. The primary goal is to quantify transition costs and the critical factor affecting DMR slowdown which is cache-refills. Others have extensively studied and reported on this phenomenon [37, 45] and so our discussion is brief. We consider dual-issue out-of-order cores with 32KB data-caches and a shared 2MB L2-cache. For transitioning to DMR, we simulate transfer delay of 10-cycles per cache-line and a 20-cycle penalty in DMR-mode for all cache refills to synchronize both cores. This models a conservative implementation to avoid incoherence for multi-threaded applications. Cache refills occur at rates ranging from every 30 cycles to every 2000 cycles and our benchmarks showed DMR slowdowns ranging from 1.1X to 2X with overall slowdowns always less than 2%.

6. RELATED WORK

We discuss other low-overhead fault detection approaches. Although some of them have lower overhead than Sampling-DMR, their fault coverage is not always 100%, and they have some other drawbacks as described earlier.

The first approach is on-line test using existing scan-chain circuits [10, 23]. Although they are simple and non-intrusive with respect to the microarchitecture and provide greater than 99% coverage for stuck-at fault model, sometimes even 100%, they have low coverage for timing faults and cause false-positive or false-negative detections because the operational environment and test environment are different. On-line test also embraces a paradigm of allowing undetected errors because faults/errors between test periods cannot be caught.

The second approach is software anomaly detection. SWAT [16, 21, 32] is a primarily software technique with some simple hardware extensions, built on the thesis that software anomalies can detect hardware faults and an absence of anomalies is inferred as error-free execution. Although it has low overhead, the coverage is low and SDCs may occur. For example, SDCs in floating-point units, SIMD datapaths, and other specialized functional units can be quite large because they don't trigger the software anomalies as extensively.

The third approach is asymmetrical hardware redundancy, which uses simpler hardware for error detection than the hardware under test. Examples include DIVA [5] and Argus [26]. Although these cover transient faults also, their overheads become large when the baseline processor itself is simple. Furthermore, re-implementing all of the datapath units such as SIMD datapaths increases their overheads.

The fourth approach is circuit level wear-out fault prediction based on the insight that all wear-out faults initially cause timing faults [3]. It may be effective for HCI/NBTI faults, since degradation is slow. However, it requires that the path excitation rate is high for signals whose arrival time is in the detection window. However, for TDDB, the transition from soft breakdown (slight delay degradation) to hard breakdown (stuck-at fault) occurs rapidly [20]. Furthermore, the approach is of limited value for gates that are not on critical timing paths.

7. CONCLUSION

As technology scales, energy efficient ways to address hardware lifetime reliability are becoming important. In this paper, we first showed that practically 100% permanent fault detection is required to sustain reasonable defect rates for future multicore chips. We then propose a novel technique for permanent fault detection by applying fundamental sampling theory to dual-modular redundancy. We use DMR for detecting errors, but restrict it to a small sampling window. First, this provides 100% fault coverage with low overhead. Second, simple designs, even if slow, become reasonable to consider, which we demonstrate with our simple FIFO-based design that leaves the processor pipeline effectively unmodified.

We developed a detailed mathematical model and extensive empirical evaluation and show that 1%-DMR and 5%-DMR with simple checkpointing result in error-free execution for 96% and 99% of faults for the OpenRISC processor.

The ideas and evaluation in the paper result in three main implications. First, our results showed that even 1%-DMR compares favorably to conventional techniques in terms of defect rate and detection latency. Second, we showed that conventional techniques and Sampling-DMR introduce the issue of some number of undetected errors in hardware and fault coverage alone as a metric for system designers is of limited value. We contend that system designers must embrace a paradigm of some hardware errors to provide low overhead and practical reliability support for permanent faults. Providing latency bounds, guarantees on number of errors etc., can then become practical aids for systems developers. Finally, the general principle of Sampling-DMR opens up possibility for other implementations and uses.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers and the Vertical group for comments and the Wisconsin Condor project and UW CSL for their assistance. We thank Kazumasa Nomura for help in developing the proof on Sampling-DMR's upper-bound error analysis. We thank José Martínez for detailed comments and feedback that immensely helped improve the presentation of this paper. Many thanks to Guri Sohi, Kewal K. Saluja, and Mark Hill for several discussions that helped refine this work. Support for this research was provided by NSF under the following grants: CCF-0845751, CCF-0917238, and CNS-0917213 and Toshiba corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

9. REFERENCES

[1] Ccc visioning study on cross-layer reliability, http://www.relxlayer.org/.

[2] Semiconductor Industry Association (SIA), Design, International Roadmap for Semiconductors, 2009 edition.

[3] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra. Circuit failure prediction and its application to transistor aging. In *VLSI Test Symposium*, 2007.

[4] A. Ansari, S. Feng, S. Gupta, and S. Mahlke. Necromancer: enhancing system throughput by animating dead cores. pages 473–484, 2010.

[5] T. Austin. DIVA: A Reliable Substrate for Deep Submicron MicroarchitectureDesign. In *MICRO '99*.

[6] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An analysis of latent sector errors in disk drives. In *SIGMETRICS*, pages 289–300, 2007.

[7] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *PACT '08*.

[8] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25:10–16, November 2005.

[9] M. A. Breuer, S. K. Gupta, and T. Mak. Defect and Error Tolerance in the Presence of Massive Numbers of Defects. *IEEE Design and Test*, 21(3):216–227, 2004.

[10] K. Constantinides, O. Mutlu, T. M. Austin, and V. Bertacco. Software-based online detection of hardware defects mechanisms, architectural support, and evaluation. In *MICRO '07*, pages 97–108.

[11] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An

architectural framework for software recovery of hardware faults. In *ISCA*, 2010.

[12] S. Feng, S. Gupta, A. Ansari, and S. Mahlke. Shoestring: Probabilistic soft-error reliability on the cheap. In *ASPLOS-15*, 2010.

[13] M. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz. Transient-fault recovery for chip multiprocessors. In *ISCA '03*.

[14] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The stagenet fabric for constructing resilient multicore systems. In *MICRO 41*, pages 141–151, 2008.

[15] A. Haggag, M. Moosa, N. Liu, D. Burnett, G. Abeln, M. Kuffler, K. Forbes, P. Schani, M. Shroff, M. Hall, C. Paquette, G. Anderson, D. Pan, K. Cox, J. Higman, M. Mendicino, and S. Venkatesan. Realistic Projections of Product Fails from NBTI and TDDB. In *Reliability Physics Symposium Proceedings*, pages 541 –544, 2006.

[16] S. K. S. Hari, M.-L. Li, P. Ramachandran, B. Choi, and S. V. Adve. mSWAT: Low-Cost Hardware Fault Detection and Diagnosis for Multicore Systems. In *MICRO '09*.

[17] L. Huang and Q. Xu. Test economics for homogeneous manycore systems. In *ITC*, 2009.

[18] U. R. Karpuzcu, B. Greskamp, and J. Torrellas. The BubbleWrap many-core: popping cores for sequential acceleration. In *MICRO '09*.

[19] C. LaFrieda, E. Ipek, J. F. Martinez, and R. Manohar. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *DSN '07*, 2007.

[20] Y. Lee, N. Mielke, M. Agostinelli, S. Gupta, R. Lu, and W. McMahon. Prediction of logic product failure due to thin-gate oxide breakdown. In *IRPS*, 2006.

[21] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *ASPLOS XIII*, pages 265–276, 2008.

[22] X. Li and D. Yeung. Application-Level Correctness and its Impact on Fault Tolerance. In *HPCA '07*, 2007.

[23] Y. Li, S. Makar, and S. Mitra. Casp: concurrent autonomous chip self-test using stored test patterns. In *DATE '08*, pages 885–890.

[24] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, , and D. A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News (CAN)*, 2005.

[25] E. J. McCluskey, A. Al-Yamani, J. C.-M. Li, C.-W. Tseng, E. Volkerink, F.-F. Ferhani, E. Li, and S. Mitra. Elf-murphy data on defects and test sets. *VLSI Test Symposium, IEEE*, 2004.

[26] A. Meixner, M. E. Bauer, and D. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. In *MICRO '07*.

[27] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim. Robust system design with built-in soft-error resilience. *Computer*, 38(2):43–52, 2005.

[28] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt. Detailed design and evaluation of redundant multithreading alternatives. In *ISCA '02*, pages 99–110.

[29] Openrisc project, http://opencores.org/project,or1k.

[30] I. Pomeranz and S. M. Reddy. An efficient non-enumerative method to estimate path delay fault coverage. In *ICCAD*, pages 560–567, 1992.

[31] M. Prvulovic, Z. Zhang, and J. Torrellas. Revive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors. In *ISCA '02*.

[32] S. K. Sahoo, M.-L. Li, P. Ramchandran, S. Adve, V. Adve, , and Y. Zhou. Using likely program invariants to detect hardware errors. In *DSN '08*.

[33] B. Schroeder, E. Pinheiro, and W.-D. Weber. Dram errors in the wild: a large-scale field study. In *SIGMETRICS '09*, pages 193–204.

[34] B. Schroeder, E. Pinheiro, and W.-D. Weber. Dram errors in the wild: a large-scale field study. In *SIGMETRICS '09*, pages 193–204, 2009.

[35] E. Schuchman and T. N. Vijaykumar. BlackJack: Hard Error Detection with Redundant Threads on SMT. In *DSN '07*, pages 327–337.

[36] S. Shamshiri, P. Lisherness, S.-J. Pan, and K.-T. Cheng. A cost analysis framework for multi-core systems with spares. In *Proceedings of International Test Conference*, 2008.

[37] J. C. Smolens, B. T. Gold, B. Falsafi, and J. C. Hoe. Reunion: Complexity-effective multicore redundancy. In *MICRO 39*, 2006.

[38] J. C. Smolens, B. T. Gold, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. Fingerprinting: bounding soft-error detection latency and bandwidth. In *ASPLOS-XI*, pages 224–234, 2004.

[39] D. J. Sorin, M. M. K. Martin, M. D. Hill, and D. A. Wood. Safetynet: improving the availability of shared memory multiprocessors with global checkpoint/recovery. In *ISCA '02*.

[40] V. Sridharan, D. A. Liberty, and D. R. Kaeli. A taxonomy to enable error recovery and correction in software. In *Workshop on Quality-Aware Design*, 2008.

[41] Standard Performance Evaluation Corporation. *SPEC CPU2006*, 2006.

[42] A. W. Strong, E. Y. Wu, R.-P. Vollertsen, J. Sune, G. L. Rosa, T. D. Sullivan, S. E. Rauch, and III. *Reliability Wearout Mechanisms in Advanced CMOS Technologies*. Wiley-IEEE Press.

[43] D. Sylvester, D. Blaauw, and E. Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *IEEE Design and Test*, 23(6):484–490, 2006.

[44] X. Tang and S. Wang. A low hardware overhead self-diagnosis technique using reed-solomon codes for self-repairing chips. *Computers, IEEE Transactions on*, 59(10):1309 –1319, oct. 2010.

[45] P. M. Wells, K. Chakraborty, and G. S. Sohi. Mixed-mode multicore reliability. In *ASPLOS -XIV*, 2009.