Modeling the Impact of Permanent Faults in Caches

DANIEL SÁNCHEZ, University of Murcia[1]
YIANNAKIS SAZEIDES, University of Cyprus
JUAN M. CEBRIÁN, JOSÉ M. GARCÍA, and JUAN L. ARAGÓN, University of Murcia

The traditional performance cost benefits we have enjoyed for decades from technology scaling are challenged by several critical constraints including reliability. Increases in static and dynamic variations are leading to higher probability of parametric and wear-out failures and are elevating reliability into a prime design constraint. In particular, SRAM cells used to build caches that dominate the processor area are usually minimum sized and more prone to failure. It is therefore of paramount importance to develop effective methodologies that facilitate the exploration of reliability techniques for caches.

To this end, we present an analytical model that can determine for a given cache configuration, address trace, and random probability of permanent cell failure the exact expected miss rate and its standard deviation when blocks with faulty bits are disabled. What distinguishes our model is that it is fully analytical, it avoids the use of fault maps, and yet, it is both exact and simpler than previous approaches. The analytical model is used to produce the miss-rate trends (*expected miss-rate*) for future technology nodes for both uncorrelated and clustered faults. Some of the key findings based on the proposed model are (i) block disabling has a negligible impact on the *expected miss-rate* unless probability of failure is equal or greater than 2.6e-4, (ii) the fault map methodology can accurately calculate the *expected miss-rate* as long as 1,000 to 10,000 fault maps are used, and (iii) the *expected miss-rate* for execution of parallel applications increases with the number of threads and is more pronounced for a given probability of failure as compared to sequential execution.

Categories and Subject Descriptors: B.3.4 [**Hardware**]: Memory Structures—*Reliability, testing, and fault-tolerance*

General Terms: Processor Caches, Process Variation, Reliability

Additional Key Words and Phrases: Reliability, caches, yield, fault tolerance

ACM Reference Format:
Sánchez, D., Sazeides, Y., Cebrián, J. M., García, J. M., and Aragón, J. L. 2013. Modeling the impact of permanent faults in caches. ACM Trans. Architec. Code Optim. 10, 4, Article 29 (December 2013), 23 pages.
DOI: http://dx.doi.org/10.1145/2541228.2541236

[1] Currently working for Intel Labs Barcelona.

This work has been jointly supported by the Spanish MEC and European Commission FEDER funds under grant “TIN2012-31345” and “TIN2012-38341-C04-03.’ Daniel Sánchez was also supported by a mobility grant by HiPEAC (FP7 Network of Excellence). The research leading to this article is supported by the European Commission FP7 project “Energy-Conscious 3D Server-on-Chip for Green Cloud Services” (Project No: 247779 “EuroCloud”)’.

Author’s addresses: D. Sánchez (corresponding author), J. M. Cebrián, J. M. García, and J. L. Aragón, Dept. of Computer Science, University of Murcia, Murcia 30100, Spain; email: {dsanchez, jmcebrian, jmgarcia, jlaragon}@ditec.um.es; Y. Sazeides, Dept. of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus; email: yanos@cs.ucy.ac.cy.

DOI: http://dx.doi.org/10.1145/2541228.2541236

1. INTRODUCTION

For the past 50 years, technological advances have enabled the continuous miniaturization of circuits and wires. The increasing device density offers designers the opportunity to place more functionality per unit area and, in recent years, has allowed the integration of large caches and many cores into the same chip. Unfortunately, the scaling of the device area has been accompanied by at least two negative consequences: a slowdown of both voltage scaling and frequency increase due to slower scaling of leakage current as compared to area scaling [Borkar 1999; Frank 2002; Taur 2002], and a shift to a probabilistic design and less reliable silicon primitives due to static [Borkar et al. 2003] and dynamic [Bowman et al. 2009] variations.

A previously published resilience road map underlines the magnitude of the problem we are confronting [Nassif et al. 2010]. Table I shows the p_{fail} (probability of failure) predicted in Nassif et al. [2010] for inverters, latches, and SRAM cells due to random dopant fluctuations as a function of technology node.

These trends render essential the development of reliability techniques against permanent faults for future processors that are both scalable and performance effective. This is especially important for caches that take up most of the real estate in processors and contain numerous, vulnerable-to-failure SRAM cells.

In the literature many different solutions can be found to deal with permanently faulty cells such as spares, robust cells, frequency and voltage binning, and error-correcting codes. These techniques maintain the cache capacity but at the cost of either increasing the complexity or decreasing the performance of the cache. Another approach, central to this work, consists of disabling cache blocks [McNairy and Mayfield 2005; Patterson et al. 1983; Sohi 1989] that contain faulty bits upon permanent fault detection (at manufacturing or in the field). These disabled blocks are not replaced with a spare,[2] which results in a reduction of cache capacity. Block disabling is an attractive option because of its low overhead, one bit per cache block,[3] but the reduced cache capacity can degrade performance.

Previous block-disabling-based studies [Ishihara and Fallah 2005; Lee et al. 2007a, 2007b, 2011; Pour and Hill 1993; Roberts et al. 2007; Shirvani and McCluskey 1999; Sohi 1989] rely on the use of an arbitrary number, small or large, of random fault maps. Each random fault map indicates faulty cache cell locations and determines the disabled faulty cache blocks. The fault maps are used either to obtain the performance degradation of a program through cycle accurate simulation or to determine the impact on the miss-rate of a program's address trace. In general, however, the number of fault maps used in these studies is very small as compared to the number of all possible maps. Therefore, the accuracy of previous research articles in predicting expected performance has not been established.

Our proposal avoids fault-map-based analysis of block-disabling techniques by using instead an analytical model that calculates the Expected Miss Ratio (EMR) for a given application, a cache configuration, and a given random probability of permanent cell failure (p_{fail}). We also show how to obtain the standard deviation for the EMR (SD_MR), which provides an indication for the range of expected degradation of the cache. Additionally, we extend our model to take into account the variation of faults across a wafer and examine how this affects both performance and yield. Furthermore, we explain how to calculate a probability distribution for the EMR for a given number of faulty blocks. All of this is accomplished without producing and using fault maps.

[2] Disabling can be employed after spares have been exhausted.

[3] This logical bit needs to be resilient either through circuit design or extra protection, because if faulty it renders whole cache faulty.

Table I. Predicted p_{fail} for Different Types of Circuits and Technologies

Technology	Inverter	Latch	SRAM
45 nm	≈ 0	≈ 0	6.1e-13
32 nm	≈ 0	1.8e-44	7.3e-09
22 nm	≈ 0	5.5e-18	1.5e-06
16 nm	2.4e-58	5.4e-10	5.5e-05
12 nm	1.2e-39	3.6e-07	2.6e-04

The capabilities of the proposed model are demonstrated through an analysis of the trends of the mean and standard deviation of the cache miss rate with changing feature size (and p_{fail}). This analysis reveals that caches that disable blocks with faulty bits incur a minimal increase in the expected miss ratio when cell p_{fail} is up to 2e-4. Other analysis for the programs and cache configurations used in this study shows that the random fault map methodology provides highly accurate mean and standard deviation estimations when using 100 to 1,000 maps. A correlation analysis investigating these results reveals a very high degree of correlation between the number of accesses and the access distribution across sets for several benchmarks. This implies that a relatively small number of fault maps suffices to capture the mean and standard deviation of the cache miss rate.

This article also presents results regarding the implications of fault variation across a wafer on yield and performance. Assuming that fault clustering is given at the level of dies (caches), whereas the specific localization of faults for a given cache is randomly distributed [Cheng et al. 2011], we find the effect of fault variation to mainly depend on the cache associativity. If associativity is low, a higher clustering of faults improves both performance and yield, whereas high associativity affects negatively. One other study in this article examines how the number of threads of parallel applications affects the performance of faulty caches. showing that, in general, the performance impact of permanent faults increases with the number of threads. Finally, we show how our model can be used to compare two different graceful degradation techniques.

The rest of this paper is organized as follows: Section 2 reviews related work. Section 3 presents our model to calculate the EMR and SD_MR, and Section 4 extends our model to take into account variation effects. In Section 5 we describe the methodology followed in the evaluation, which is presented in Section 6. Finally, Section 7 summarizes the main conclusions of this work.

2. BACKGROUND AND RELATED WORK

2.1. Sparing and Circuit-Level Approaches

In the past, when variation issues were less dominant, it may have been acceptable during postmanufacturing tests to discard chips even with a single faulty cache bit. Nowadays, this is not a viable approach as reflected by the use of extensive spare columns and rows in contemporary cache SRAM arrays [Le et al. 2007]. However, the amount of spares needed to ensure a fault-free cache can grow faster than the area scaling rate and, consequently, diminish scaling benefits.

An approach that can reduce the amount of spares needed to address parametric variations and power constraints is the use of more robust cells [Agarwal and Nassif 2006]. SRAM cells are commonly implemented with a conventional six-transistor structure. This design provides enough stability at today's nominal voltage. However, limiting power consumption requires a reduction in the supply voltage. In this case, the Static Noise Margin (SNM) decreases significantly, which increases soft and parametric errors. Some studies reveal that voltage cannot be scaled down to 0.7V for an SRAM cell at 65nm to work properly [Yamaoka et al. 2004; Zhang et al. 2004]. To mitigate the effects of minimum supply voltage in SRAM cells, engineers have proposed new designs using more transistors per cell such as 8T [Verma and Chandrakasan 2008] and 10T [Kim et al. 2008] designs. These cells have lower $p_{fail,}$ but each one requires more transistors and a larger area, which results in longer overall cache access time. Another approach is frequency and voltage binning, which results in operating a chip at lower frequency or higher voltage than intended, so that all cells can be accessed correctly [Borkar et al. 2003]. This ensures functional correctness, but it either reduces performance or increases power/temperature.

The above approaches aim to provide fault-free caches by mitigating manufacturing and static parametric faults. This is desirable but not realistic to accomplish cost-effectively for future caches due to the mismatch between the cell p_{fail} versus area rate of scaling. Furthermore, wear-out faults that occur in the field are becoming more common [Nassif et al. 2010]. Consequently, we may be forced to ease the requirement of shipping only chips with fault-free caches and replacing parts that experience a wear-out fault. But this may require performance cost-effective mechanisms to deal with permanent faults in a cache during operation.

2.2. Architectural Solutions Based on Disabling

Another set of approaches is based on disabling of faulty cache portions, also referred to as graceful degradation techniques. One solution is to disable blocks [Patterson et al. 1983; Sohi 1989] that contain faulty bits upon permanent error detection (at manufacturing time or in the field). Such disabled blocks are not replaced with a spare and, therefore, the cache capacity is reduced. Block disabling is an attractive option because it has low overhead, but its reduced cache capacity could degrade performance.

Block disabling is not a new concept. It has been proposed and evaluated before [Ishihara and Fallah 2005; Lee et al. 2007a, 2007b; Pour and Hill 1993; Roberts et al. 2007; Shirvani and McCluskey 1999], for example, as a way to improve manufacturing yield [Lee et al. 2007b; Pour and Hill 1993; Sohi 1989] and to enable cost-effective operation below Vcc-min [Ladas et al. 2010].

Finer-grain disabling techniques have also been proposed, such as word disabling (wdis) and bit-fix [Wilkerson et al. 2008]. The *wdis* technique tracks faulty data cells at word granularity by means of fault masks. These masks are kept at every line's tag and contain as many bits as words, each one indicating whether the corresponding word is disabled or not. When the *wdis* technique is deactivated, the fault mask is ignored. However, when it is active, all pairs of consecutive blocks in a set are combined to form one logical block. The effect of this mechanism is that both the capacity and associativity of the cache are reduced by half. In order to obtain a logical block in aligned form, *wdis* introduces a shift-multiplexer network, which is controlled by each block's fault mask and is used to discard the defective words. This way, *wdis* tolerates up to $n/2$ faulty words in a logical block with n words. Unfortunately, the alignment network increases the access latency of the cache. Specifically, for eight-word blocks, *wdis* requires a line to pass through four different multiplexers (to discard up to four faulty words), something that increases the latency of the cache by one cycle [Wilkerson et al. 2008]. In *bit-fix* [Wilkerson et al. 2008], the granularity of disabling is extended to the bit level. In this scheme, a quarter of the ways in a cache are used to provide the repairs for the potentially faulty bits in the rest of ways, reducing the capacity of the cache by 25%. This approach requires a complex merging mechanism, which again can increase the cache access latency.

The buddy cache [Koh et al. 2009] is based on the same finer-than-block disabling principle but only groups faulty blocks. This way more cache blocks, as compared to *wdis*, remain usable and, therefore, it provides better performance. However, the buddy cache requires deep changes in the cache implementation such as the buddy map, which is used to identify which blocks can be combined, and this increases the access latency to the L2 cache. The ZerehCache [Ansari et al. 2009] can be considered as an optimization of *wdis*. Instead of merging blocks, the ZerehCache minimizes the amount of redundancy by selecting specific lines in a pool of spare lines. This way, multiple defective lines can be mapped to just one (faulty or not) spare line. The amount of cache space available remains unaltered (despite the faulty words), so the performance slowdown is limited. However, this proposal requires a mechanism to select the group of spare lines at manufacturing time, making the architecture not tolerant to permanent errors due to aging or voltage variations.

As far as we know, in all previously discussed block disabling-based schemes, the performance methodology relies on a specific number (small or large) of random fault maps. The random fault maps indicate the location of faulty cache cells and determine the disabled faulty cache blocks. They are used to either obtain the performance degradation of a program through cycle-accurate simulation or to determine the impact on miss-rate of a program's address trace through other analytical models [Agarwal et al. 1989; Pour and Hill 1993]. We claim that all these studies are limited because they do not provide a justification as to why the particular number of fault maps they use, which is typically a very small subset of the actual number of possible mappings, is representative of the expected behavior. Therefore, it remains unclear whether the conclusions reached in these studies are representative. In this work we propose an analytical methodology that completely avoids the use of random fault maps for the exact calculation of the EMR for a given application address trace, a cache configuration, and random probability of permanent cell failure (p_{fail}).

3. ANALYTICAL MODEL FOR CACHE MISS RATE BEHAVIOR IN THE PRESENCE OF UNCORRELATED FAULTS

In this section, we present an analytical model that can determine the EMR, SD_MR, and an approximate probability distribution of miss ratios (PD_MR) for a given program address trace, cache configuration, and random probability of permanent cell failure (p_{fail}) [Sánchez et al. 2011]. The EMR captures the average degradation due to random faulty cells. The SD_MR provides an indication of the range of the degradation, whereas PD_MR reveals the shape (distribution) of the degradation. These characteristics can be used to assess the implications of faults in a cache and compare different cache reliability schemes.

The model's key novelty is that it does not rely on fault maps and provides exact EMR and SD_MR rather than an approximation; that is, it determines them as if *all* possible fault maps for a given random p_{fail} have been considered. Previous studies that relied on fault maps may not have produced representative conclusions because they cannot generate and evaluate, in general, all possible cache fault maps for a given p_{fail} in a reasonable amount of time.

3.1. Assumptions and Definitions

The presented analytical model addresses the performance impact of block disabling in a cache architecture prone to permanent faults. Block disabling disables the use of a block in which at least one bit is detected as faulty. These faults are assumed to be detected with postmanufacturing and boot time tests, ECC, and built-in self-tests.

The model initially assumes that the permanent faulty cells occur randomly and are uncorrelated with probability p_{fail}. This random fault behavior is indicative of faults due to random dopant fluctuations and line-edge roughness, two prevalent sources of static variations.

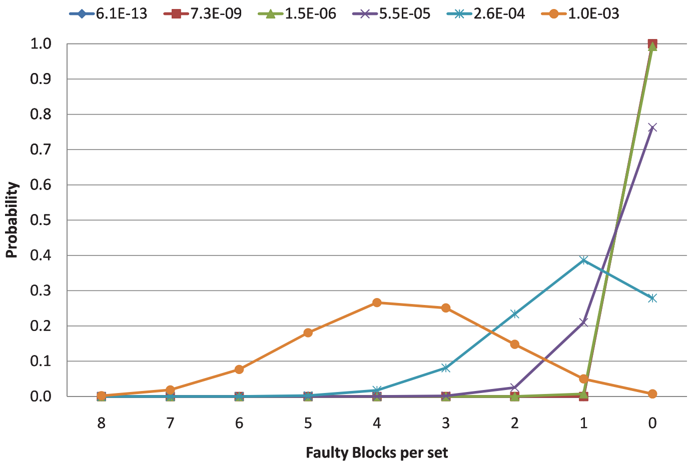


Fig. 1. pe_i with different p_{fails} for a 32 KB eight-way L1 cache.

A cache configuration is defined by the number of sets (s), ways per set (n), and block size in bits (k). We consider a block containing one or more permanent faulty bits as faulty. In that case, the faulty block is disabled and the cache capacity is reduced.

The model assumes LRU replacement policy. The possible extensions to other replacement policies are subject of future work.

Each program address trace is generated through a cache simulator to obtain for a given cache configuration the vector M. This vector contains $n + 1$ elements, one element more than the number of cache ways. M_i corresponds to the total misses when there are only $n - i$ valid ways in each set in the cache. More specifically, M_i equals the sum of all references that hit on the i least recently used blocks in each set, plus the misses of the fault-free cache. For example, M_0 equals the misses of a fault-free cache; M_n represents the misses of a cache in which all ways are entirely faulty, meaning that all accesses are misses; and M_1 equals the misses of the fault-free cache plus all the hits in the LRU position (refer to the example in Section 5.1).

3.2. EMR and SD_MR

This section shows how the model obtains the EMR and SD_MR given a cell p_{fail}, cache configuration, and the miss vector of an address trace. The model obtains the probability of block failure p_{bf} using the following expression (based on well-known binomial probability):

$$p_{bf} = 1 - (1 - p_{fail})^k \tag{1}$$

Although p_{bf} gives information about the fraction of blocks that are expected to fail in the cache, the impact on the miss ratio is still unknown as it depends on the fault's location and the amount of accesses to faulty blocks. However, with the p_{bf} we can obtain the probability distribution pe_i for the number of faulty ways in a set:

$$pe_i = \binom{n}{i} p_{bf}^i (1 - p_{bf})^{n-i} \tag{2}$$

which provides, for every possible value of i [0...n], the probability of having $n - i$ nonfaulty ways. This distribution is very useful because it provides the complete picture about how likely it is to lose a given number of ways in a set. We can see the pe_i distribution for a 32 KB, eight-way L1 cache with different p_{fails} in Figure 1. In the figure, we can see that the probability of having faulty blocks in an eight-way cache increases with higher p_{fails}. For example, we can see that for a p_{fail} 1.0e-03, the probability of having no faults in a set is close to 0, while the probability of having four faulty blocks is around 30%.

The expectation of a random variable $X = x_0, x_1 \ldots, x_m$ for which each possible value has probability $p = p_0, p_1, \ldots, p_m$ can be calculated as:

$$E[X] = \sum_{i=0}^{m} x_i \cdot p_i \tag{3}$$

In our case, the random variable X corresponds to the total number of misses for a cache with faults; x_i corresponds to the total misses when there are only $n - i$ valid ways in each set in the cache; and p_i is the probability of having i faulty ways in a set. Therefore, we can express the expectation of the number of misses for a cache with disabled blocks as:

$$E_{misses} = \sum_{i=0}^{n} M_i \cdot pe_i \tag{4}$$

and obtain the expected miss ratio of the cache using:

$$EMR = \frac{E_{misses}}{accesses} \tag{5}$$

This simple formula can be used to obtain the exact EMR without using fault maps. The key insight behind this formula, expressed better in Equation (2), is that caches have a useful property: for the same number of faulty blocks f in a set, the reduced associativity will be the same $n - f$. That is, for analyzing block-disabling approaches, what matters is the number of faulty ways in a set, not which specific ways in the set are faulty. As a result, this reduces the complexity of the problem.

The EMR provides the average case performance for a given p_{fail}. However, we have no information about the variation in the miss ratio. Variation information is useful for assessing whether disabled blocks lead to caches with a wide-variation (less predictable) miss rate.

One way to measure this variation is through the standard deviation of the miss ratio or SD_MR. Unfortunately, the standard deviation cannot be directly obtained for the whole cache. However, given that we already know the probability distribution of faulty blocks in a set, we can calculate its variance as follows:

$$\forall j[1 \ldots s], VAR\_E_{misses_j} = \sum_{i=0}^{n} pe_i \cdot (x_{ij} - E_{misses_j})^2 \tag{6}$$

where x_{ij} is the number of misses obtained when having $n - i$ nonfaulty ways in the j_{th} set.

Although the total EMR is equal to the sum of individual sets EMR_j:

$$EMR = \sum_{j=1}^{s} EMR_j \tag{7}$$

we cannot combine the variation of each set in the same way. Instead, we compute the deviation for the misses of the whole cache SD_MR by using the root mean square in the form:

$$SD\_MR = \frac{\sqrt{\sum_{j=1}^{s} VAR\_EMR_j}}{accesses} \tag{8}$$

3.3. EMR Approximate Probability Distribution

The SD_MR provides the range of deviation of the EMR. However, it may be useful also to know the probability distribution of cache misses (PD_MR) within the deviation range.

We propose to build an *approximate* probability distribution of misses in a stepwise manner. We first calculate the EMR for every possible number of faulty blocks (0 to the number of cache blocks) and then combine this information with the probability that a given number of faulty blocks occurs.

Equation (9), similar to Equation (2), gives the probability of x number of faulty blocks for a given block probability failure (p_{bf}):

$$\binom{s \cdot n}{x} p_{bf}^{x}(1 - p_{bf})^{s \cdot n - x} \tag{9}$$

This equation can be evaluated for different x to obtain a probability distribution. Then, we need to calculate the EMR for every possible number of faults. This problem has traditionally been solved by means of random fault maps [Pour and Hill 1993].

For a given number of faults, this problem is analogous to selecting at random n balls from an urn that contains dk balls without replacement, where d is the number of unique colors and k is the number of balls of each color. The urn represents the cache, the variable n the faults, d the number of blocks, and k the number of bits in each block. The mean number of distinct blocks, u, that contain at least one faulty cell in a cache with n faulty cells can be approximated with very high accuracy [Yao 1977]:

$$u = d - d(1 - p_{fail})^{k} \tag{10}$$

This means that we can obtain an approximation of PD_MR analytically, without using fault maps. Simply use Equation (10) to convert the number of faulty blocks to p_{fail}. This gives the expression:

$$p_{fail_i} = 1 - \sqrt[k]{\frac{s \cdot n - x_i}{s \cdot n}} \tag{11}$$

This way, we can calculate the p_{fail} for a cache with s sets, n ways, and x_i faults. Then, every p_{fail_i} can be used to calculate the EMR associated to each number of faulty blocks.

Although a PD_MR provides more information than simply expected values, due to space limitations and for the sake of visibility, in this paper we will only show results for the EMR. Deriving more accurate PD_MR with bounded error is the subject of an ongoing research effort.

4. MODELING VARIATION EFFECTS OF THE P_{FAIL}

The model, so far, presents the p_{fail} as a homogeneous effect. However, several previous studies [Bowman et al. 2002; Unsal et al. 2006; Bowman et al. 2007] report nonnegligible variations of p_{fail} in memory cells.

In this section, we address this variation effect by a two-level model in which faults are clustered at coarse granularity (intradie), whereas the specific location of faults within the SRAM structures is randomly distributed as suggested by Cheng et al. [2011]. This means that, in every chip, faults are homogeneously distributed. However, for different chips in a wafer the number of faults is different according to a clustering parameter. To model this clustering effect we make use of the negative binomial distribution.

4.1. Negative Binomial Distribution

When the number of components n is very large and the probability of fault p_{fail} is small, the binomial distribution presented in Equation (2) can be approximated by the Poisson distribution as follows:

$$P(k) = \frac{\lambda^k}{k!}e^{-\lambda} \tag{12}$$

where λ is the expected number of faults given by $n * p_{fail}$ and k is the number of occurrences of a given event. The Poisson distribution is not well suited to model defects or faults when they do not occur independently. Therefore, the compound Poisson distribution is introduced:

$$P(k) = \int_0^{\infty} \frac{\lambda^k}{k!}e^{-\lambda}P(\lambda)d\lambda \tag{13}$$

As proposed in Stapper et al. [1983], $P(\lambda)$ can be expressed as a gamma distribution as follows:

$$P(\lambda) = \frac{\beta^{\alpha}\lambda^{\alpha-1}e^{-\beta\lambda}}{\Gamma(\alpha)} \tag{14}$$

Finally, applying Equation (14) to Equation (13), we obtain the negative binomial distribution, which is able to adjust variance and mean with two parameters α and β:

$$P(k) = \frac{\Gamma(k+\alpha)\beta^{\alpha}}{k!\Gamma(\alpha)(1+\beta)^{k+\alpha}} \tag{15}$$

In Equation (15), $\Gamma(x)$ is calculated as $(x-1)!$, whereas α and β are adjustment parameters satisfying the following:

$$\mu = \frac{\alpha}{\beta} \tag{16}$$

and

$$\sigma^2 = \frac{\alpha}{\beta^2}(1+\beta) \tag{17}$$

in which μ is the average number of faulty components per chip and σ is its standard deviation.

Given those equations and a known μ and σ, it is easy to find parameters α and β.

Applying Equation (15) to the particular case of $k = 0$, we obtain the Yield, formulated as:

$$Y = P(k=0) = \frac{1}{\left(1+\frac{\mu}{\alpha}\right)^{\alpha}} \tag{18}$$

4.2. Applying the Negative Binomial Distribution

In this work, we follow an approach similar to the one presented by Koren et al. [1993]. The total area of the wafer is divided into segments where the number of faults follows a negative-binomial distribution. Faults within a segment are modeled by means of a uniform distribution. With this approach, each of the segments corresponds to the size of a cache. This modeling approximation is supported by previous studies that show highly correlated regions to be significantly larger than the total size of caches [Bowman et al. 2007; Cheng et al. 2011].

We use Equation (15) to calculate the probability $P(k)$ for each value $k = \{0..n\}$. This means that, for a given mean p_{fail} (μ) and a clustering parameter α, we can provide the fault density function (or p_{fail} distribution).

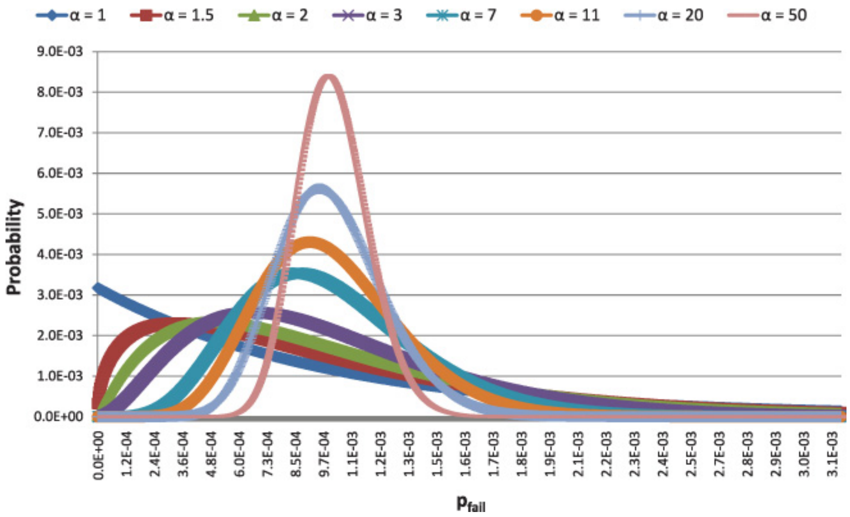


Fig. 2. p_{fail} distribution for a 32 KB, eight-way cache with $\mu = 1.0\text{e-}03$.

The effect the alpha parameter has over the p_{fail} distribution can be observed in Figure 2. When the α value is small, the clustering effects are larger, and faults concentrate in specific areas that are heavily damaged, whereas other areas remain unaffected. Therefore, the shape of the curves for small α values is wider. On the contrary, a large α value means that faults are not clustered but homogeneously distributed; therefore, the shape of the probability distribution is narrower and centered around μ (similar to Figure 1 where no variation is considered).

5. METHODOLOGY

Sections 3 and 4 introduced the proposed model. In this section, we explain the methodology used to extract the per application information needed by our model. We also explain how we produce random fault maps used for comparing the proposed model and the fault-map methodology.

5.1. Generating Maps of Accesses

The input to the analytical model is a map of accesses to the cache for every application. One way to accomplish this is to run the application for every possible cache configuration. In order to avoid this cost, we have used an algorithm called all-associativity simulation [Hill and Smith 1989], previously used in Pour and Hill [1993].

This algorithm takes as input a trace of memory accesses, which is converted to a map of accesses to a cache of any desired configuration (sets and ways) following a deterministic replacement policy (LRU in our case). This allows us to obtain the number of accesses per way and per set without the need for new simulation runs. The output of the algorithm is a matrix in which each row corresponds to a set and each column to a position in the LRU stack. Each value of the matrix indicates the number of accesses to every position in the LRU sequence for every set. This information is highly useful for offline analysis given that we can extract the number of misses for a given number of ways w in our cache by simply adding the accesses for the last $n - w$ columns of the matrix.

We can see an example of how this method works in Table II for a cache with four ways and four sets. The first column from the left (*way3*) refers to the accesses to the first (most recently used) position in the LRU stack for each set. The last column (*misses*) refers to accesses to data not previously accessed or with a position in the LRU sequence

Table II. EMR Calculation after Executing the All-Associativity Algorithm

(a)

	way3	way2	way1	way0	misses
set0	120	100	110	60	100
set1	150	140	110	55	90
set2	180	134	80	50	200
set3	220	200	100	30	180

(b)

	way3	way2	way1	way0	misses
set0	490	370	270	160	100
set1	545	395	255	145	90
set2	644	464	330	250	200
set3	730	510	310	210	180

(c)

	way3	way2	way1	way0	misses
cache	2409	1739	1165	765	570

greater than our threshold, meaning a miss. In order to calculate the number of misses of the cache in a fault-free environment, we simply sum the accesses that appear in the last column. The number of misses of this example would be 570 $(100+90+200+180)$.

But this table can be used to compute the misses in a faulty environment. For this, and according to Equation (5), we need to calculate the number of misses with a given number of ways (from 0 to 4 in our example) disabled in the cache due to permanent faults. First, we accumulate the number of accesses in every position per set. The result of this is in Table II(b). Finally, we perform the same operation per set to get the cumulative vector in Table II(c). This vector indicates exactly the number of misses the cache would suffer as a result of losing from 0 to w ways. Finally, this vector can be used by the model to obtain the EMR and SD_EMR for a faulty cache.

This methodology is similar to the methodology used in DEFCAM [Lee et al. 2011]. DEFCAM uses a greedy algorithm over the matrix of accesses to determine the minimum and maximum impact of disabling lines and sets. However, DEFCAM still relies on the simulation of multiple faults maps to produce average miss rates. Our approach is different from DEFCAM in that it avoids completely the evaluation of different fault maps while providing a precise value for the EMR and its standard deviation.

5.2. Misses Based on Random Fault Maps

Part of the analysis in the paper examines how accurate the random fault-map-based methodology is in estimating EMR and SD_MR. Each random fault map reflects a number of disabled ways for every set in our architecture, regardless of the location of these faults in the set. This is so because the position does not alter the number of misses in the cache because of associativity and LRU replacement policy.

To generate these random maps, we have used the GNU Scientific Library (GSL). We generate random numbers in the interval [0, 1] to mark the faulty blocks in terms of a given p_{fail}. Finally, we compute the number of misses produced by a fault map by using the access maps obtained with the all-associativity algorithm.

6. EVALUATION

For the experiments, we simulate a processor architecture by means of Virtutech Simics [Magnusson et al. 2002] and GEMS [Martin et al. 2005]. We have performed several modifications to the simulator to extract memory address traces. Then, we use these traces to generate the map of accesses for every possible cache configuration by means of the all-associativity algorithm as explained in Section 5.

We conducted experiments by executing both sequential applications from SPECcpu-2006 [Henning 2006] (*bzip2, gcc, hmmer, mcf, perlbench, sjeng*) and parallel applications from SPLASH-2 [Woo et al. 1995] (*Barnes, Cholesky, EM3D, FFT, Ocean, Radix, Raytrace, Tomcatv, Unstructured, Waternsq, Watersp*) and PARSECv2.1 [Bienia et al. 2008] (*Blackscholes, Fluidanimate, Swaptions*). In the case of parallel benchmarks, the experimental results reported correspond to the parallel phase of each program. Benchmarks are executed in a CMP with private L1 caches and a shared L2. The number of cores in each case depends on the number of threads. In all cases, the warming up of caches has been taken into account. For the evaluation of L1 caches, benchmarks have been executed for 300 million cycles, whereas for the evaluation of the L2 cache, benchmarks have been run to completion.

The different p_{fails} used for the evaluation of the caches are those shown in Table I with the exception of the 6.1e-13 p_{fail}, which produces virtually no faulty blocks in our experiments. All these p_{fails} are predictions for the fault probability of SRAM cells for different scales of integration. Additionally, we have used p_{fail} 1e-03, which is usually studied in related work.

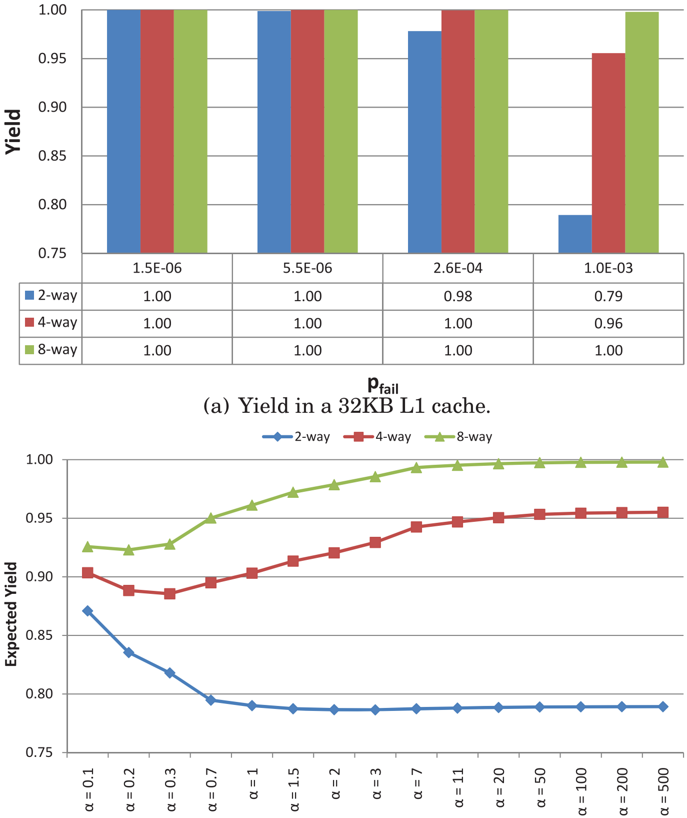
6.1. Yield Analysis

One important tool that helps to understand the behavior of a cache prone to permanent faults is the Yield analysis. The Yield is a statistical metric referred to as the proportion of valid caches (or other devices) from a population. In this article, we consider that a cache with permanent faults cannot be used (i.e., its Yield is 0) when all the blocks in a specific set have been disabled. In this section, we assume that a cache block is disabled if it has one or more faulty bits.

Figure 3(a) presents the Yield for a 32 KB L1 cache with different associativities and p_{fails}. As depicted, Yield is unaffected when the number of ways is eight. However, if the associativity is reduced, then a significant number of caches become unusable. This effect is more clearly seen for higher p_{fails} such as 1.0e-03, for which the Yield is reduced to 96% and 79% for four-way and two-way caches, respectively.

However, this first study does not take into account the effect of clustering. As stated before, models that do not address clustering make skewed predictions in relation to Yield. Figure 3(b) shows the Yield for a p_{fail} of 1.0e-3 for different degrees of clustering. A small value of the clustering parameter α indicates that faults tend to group in specific locations on a wafer, whereas a high value indicates that faults are homogeneously distributed. It is noteworthy that when increasing the α value, the expected Yield converges to the Yield without clustering. This is expected since a higher value of α implies higher dispersion. In other words, a higher fault dispersion means more homogeneous fault distribution caches, whereas less dispersion leads to caches with a high density of faults or to caches with few or no faults.

As we can see in Figure 3(b), fault dispersion has a different effect depending on the cache associativity. On the one hand, medium to high associative caches (four- and eight-way caches) are able to mitigate a moderate density of faults without affecting the usability of the cache; that is, rarely all blocks of the same set are disabled. Therefore, a homogeneous distribution of the faults results in an increased Yield. On the other hand, the density of faults that low associative caches (two-way in Figure 3(b)) are able to absorb is limited. Consequently, the Yield of the entire population is higher because most faults occur in few caches (which become not usable). In conclusion, when the associativity is moderately high, Yield improves with fault dispersion, whereas low associativity is better with fault clustering.



(a) Yield in a 32KB L1 cache.

(b) Clustering with $p_{fail} = 1.0e-3$.

Fig. 3. Yield behavior without (a) and with (b) clustering effects.

6.2. Faulty Block Distribution

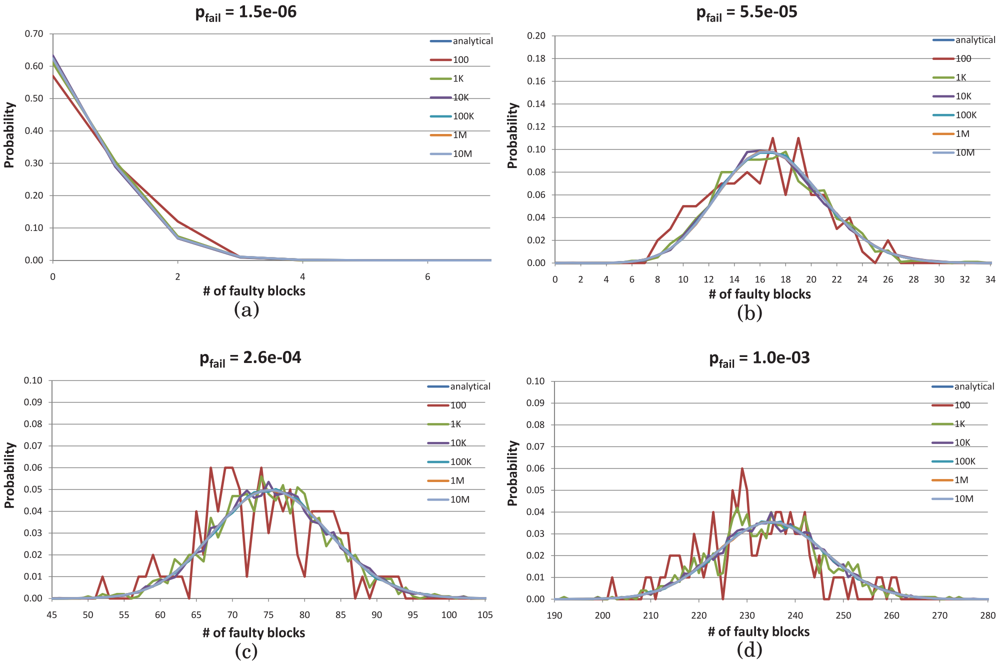
We validate the number of faulty blocks produced using randomly generated fault maps by comparing against those obtained using Equation (1).

In Figure 4, we can see the probability distribution of the number of faulty blocks for different p_{fails} (we omitted 6.1e-13 because it has almost perfect yield) in a 32KB, eight-way associative cache with 615 bits per block.[4] Results show the estimated faulty blocks provided by Equation (1) (analytical) and by different numbers of random fault maps (from 100 to 10 million). As can be observed, few fault maps are not able to catch the exact behavior. However, when the number of maps is increased (1K maps or more), the number of faulty blocks per cache is obtained with high accuracy. Nonetheless, this analysis cannot show how adequate approximation random fault maps are for estimating the expected misses of a cache since it does not consider cache access distribution.

6.3. EMR and SD_MR for Sequential Benchmarks

In this section, we show the estimated EMR for several sequential benchmarks for an eight-way 32KB L1 with different permanent cell p_{fails}.

[4] We consider L1 cache blocks composed of 64 bytes for data and eight bytes for its ECC; 28 bits for the tag; three control bits for valid, disable, and dirty states; and one byte for its ECC.



(a) (b) (c) (d)

Fig. 4. Probability distribution for the number of faulty blocks per cache obtained analytically and by randomly generated maps.

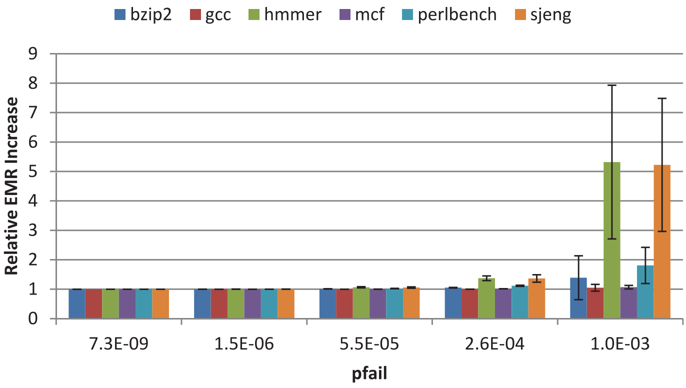
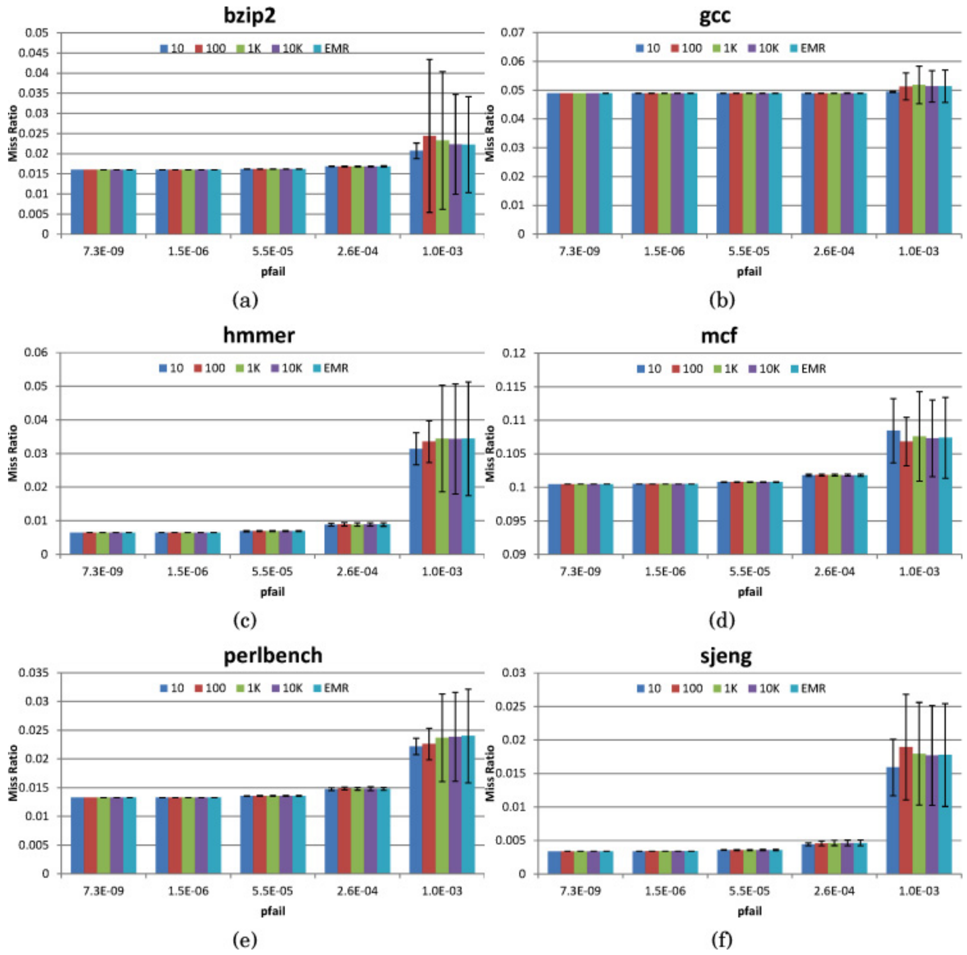


Fig. 5. EMR and SD_MR relative increase for sequential applications in an eight-way 32KB L1 cache.

6.3.1. Uncorrelated Faults. Figure 5 shows the relative EMR increase with respect to a fault-free scenario for different applications and p_{fails}. As we can see, the impact of permanent faults in EMR varies among different traces. In particular, applications such as *hmmer* and *sjeng* are affected even at very low p_{fails} such as 2.6e-04, whereas the impact on other benchmarks is negligible. We can notice that SD_MR also grows in relation to the p_{fail} and that this growth is application dependant as well.

In Section 6.2, we showed how fault maps can approximate the number of faulty blocks per cache. Nonetheless, it is still unknown how many maps are needed to approximate the EMR. The answer can be found in Figure 6, which shows the EMR for



(a) (b)

(c) (d)

(e) (f)

Fig. 6. EMR and SD_MR for different applications in an eight-way associative 32KB L1 cache with different p_{fails}.

different applications for an eight-way 32 KB L1 cache. As we can see, using 10 and 100 random fault maps is not enough to approximate the results provided by our model.

For the studied benchmarks, 10 and 100 fault maps underestimate or overestimate the EMR for p_{fail} 1.0e-03, while differences are barely noticeable with the smallest p_{fails}. This is due to the fact that the number of affected blocks is very small and the effect over the total number of misses is negligible. However, with a higher number of maps, the EMR and SD_MR converge to the one provided by the analytical model. In general, we can state that 1,000 to 10,000 maps are needed to produce very accurate values for both EMR and SD_MR. Therefore, we can state that our analytical model, besides producing accurate results, requires much less time to provide them since we only require one execution of the benchmark.

6.3.2. Clustering Effects. When calculating the EMR in the presence of a clustering parameter, the result is a distribution of EMR rather than a single value. This means we obtain a scalar value plus a probability of that EMR to happen. In order to make

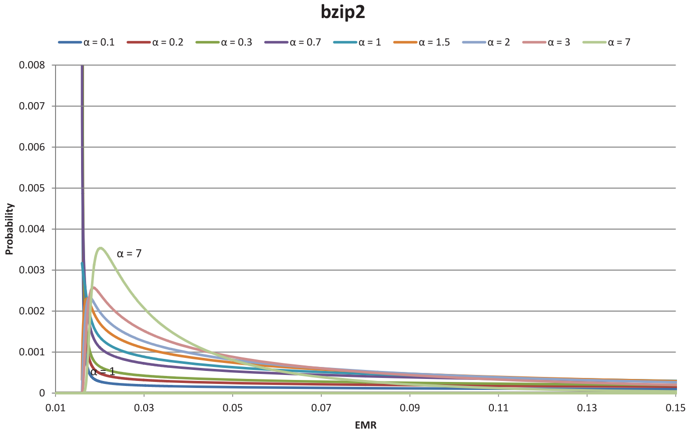


Fig. 7. EMR probability distribution for an eight-way 32 KB L1 cache with different α values for 1.0e-3 p_{fail}.

these EMR distributions simpler to understand, we produce the "Expected EMR" or "EEMR," which is the result of the addition of every EMR multiplied by its probability.

Figure 7 shows the EMR distribution for *bzip2* in the presence of fault clustering for a fixed average number of faults per device μ given by $\mu = p_{fail} * k$ for different α values. For a high clustering degree, $\alpha = 0.1$, the curve behaves as an asymptote. This means that the distribution of faults among different caches is erratic; that is, some of them are not affected at all, whereas others are deeply affected. On the contrary, when clustering degree is lower, $\alpha = 7$, faults are homogeneously distributed among all the caches and the curve for the EMR behaves as a Gaussian bell. For the sake of visibility, in the rest of the article we will only show the expected values for the EMR distributions (or EEMR), given by Equation (3).

Figure 8 shows the EMR for different degrees of fault clustering for a 32KB, two-way L1 cache with 1.0e-03 p_{fail}.[5]

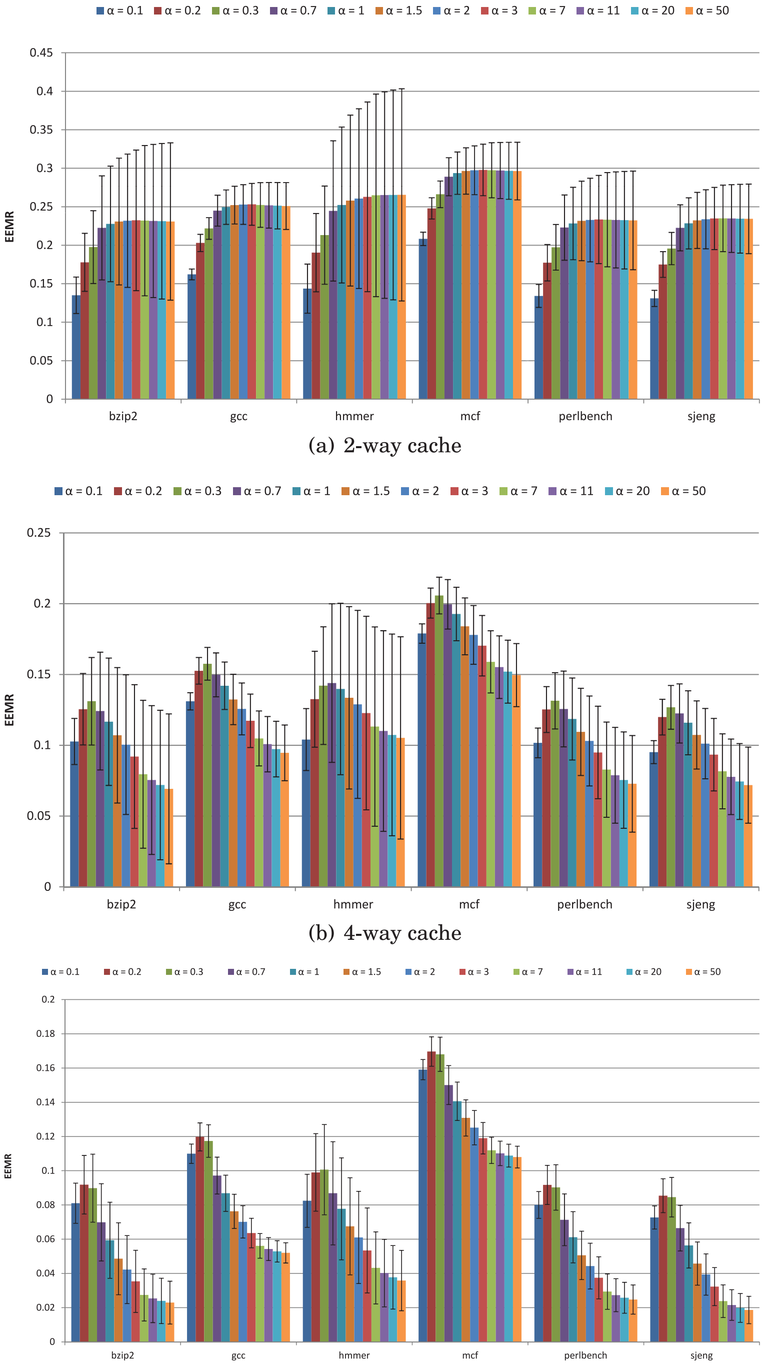
As depicted in Figure 8(a), for two-way caches a higher level of clustering (i.e., a lower value of α) provides a reduced EMR than with lower levels of clustering for all the studied benchmarks. However, this trend is different for four-way and eight-way caches as shown in Figure 8(b) and Figure 8(c), respectively.

These results are consistent with the Yield analysis presented in Section 6.1. As we can see, the effect of faults is more noticeable in low-associativity caches when using block disabling. This way, if faults are concentrated in some chips on a wafer and the rest of them remain unaffected, the impact on the EMR is lower. On the other hand, higher cache associativities are more tolerant to faults. Therefore, if faults are more homogeneously distributed, the expected EMR is reduced.

Finally, as depicted in Figure 8(c), when $\alpha >= 7$, the negative-binomial distribution behaves similarly to the binomial distribution and EMR results converge, as shown in Figure 6.

6.3.3. Correlation Analysis. Surprisingly, this study shows that a moderately small number of random fault maps is enough to approximate the EMR. The reason is the uniform accesses to the different cache sets. The reason for this is that there are no particular sets in which the accesses per way are clearly more accessed than others along the overall execution of the benchmark. This makes the EMR virtually independent of the

[5] We have reduced this analysis for this specific p_{fail} due to space limitations.



(a) 2-way cache

(b) 4-way cache

(c) 8-way cache

Fig. 8. Expected EMR for a 32KB L1 cache with different α values for $p_{fail} = 1.0e\text{-}3$.

Table III. Average for the Pearson Coefficient Matrix for Every Benchmark

Benchmark	Mean Pearson Coeff.	DEV
bzip2	.999	.0005
gcc	.999	.0001
hmmer	.992	.0197
mcf	.999	.0002
perlbench	.997	.0039
sjeng	.994	.0139

allocation of faults, and that is the reason why 1,000 to 10,000 fault maps are able to provide such accurate estimations.

In order to measure the cache access uniformity, we have performed a study of the correlation of accesses between all the sets in the L1 cache used in this study. We have used the Pearson correlation coefficient. Given X and Y, two different sets of our cache with different accesses per way

$$X = x_1 \dots x_n, Y = y_1 \dots y_n \tag{19}$$

the Pearson correlation coefficient is calculated as:

$$r_{xy} = \frac{\sum_{i=1}^{N}(x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{20}$$

When r_{xy} is close to 0, it means that there is no correlation between variables, whereas when it is close to 1, there exists a relation between them. We have calculated the matrix of correlations for the number of accesses to an eight-way 32KB L1 cache for the SPEC benchmarks. Table III reflects the average value for the Pearson coefficients as well as its standard deviation. As we can see, all coefficients are very close to 1 and deviations are small. This means that the accesses among sets are highly correlated.

But still we can study the significance of the Pearson correlation coefficient to assess if the correlation among the variables is true as it was not produced arbitrarily. For that, we can calculate Student's t as:

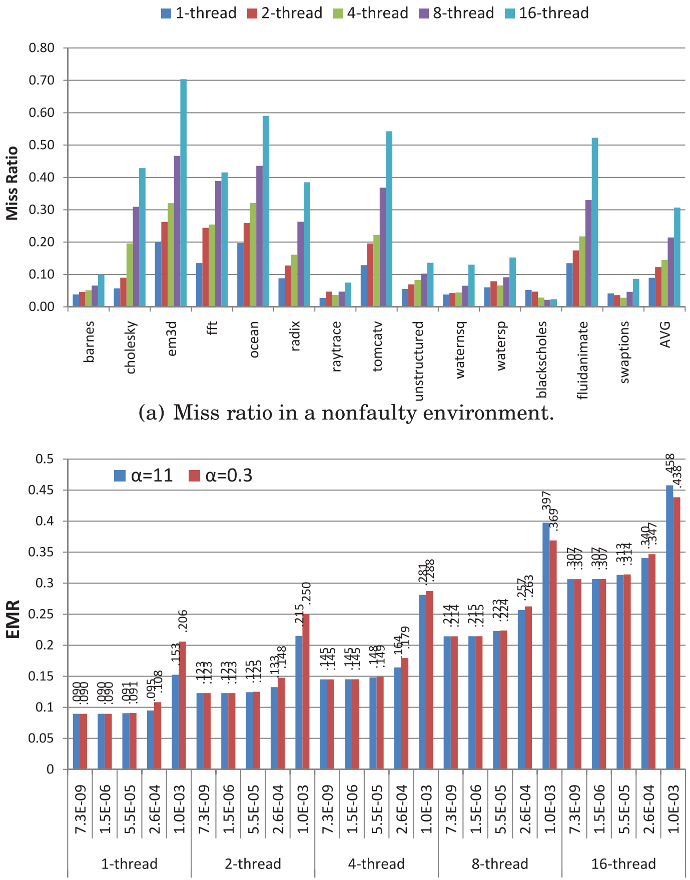
$$t = \frac{r_{xy}}{\sqrt{\frac{1-r_{xy}^2}{N-2}}} \tag{21}$$

in which N is the number of values for the variables. In our example, for an eight-way cache, we have that $N = 8 + 1$, taking into account the number of cold misses. For the worst correlation exhibited by the studied benchmarks, which is *hmmer* with .992, we have that $t = 10.21$. By means of a calculator for the t distribution, for a degree of liberty $N = 7$, we can determine that our sets are correlated with a maximum risk of 1e-05 corroborating our hypothesis.

The lesson from this study is that when accesses across sets are correlated, the amount of random fault maps needed to approximate the EMR is moderately low. In cases in which the correlation is not high, only a larger number of fault maps will be needed.

6.4. EMR for Shared-Memory Applications

The model can also be used to study how the EMR is affected when varying the number of threads for a parallel application. In this study, each thread runs in a different core of a CMP. This way, both the number of cores and the number of private L1 caches increase with the number of threads. The size of the shared L2 cache remains constant.



(a) Miss ratio in a nonfaulty environment.

(b) EMR for different p_{fails}.

Fig. 9. Results for an eight-way 512 KB L2 cache for parallel applications.

The general trend in a nonfaulty environment for the parallel applications we have executed can be seen in Figure 9(a). The miss ratio increases with the number of threads. This is due to two main reasons. First, the number of L2 accesses decreases with the number of threads/cores as a result of the increased total available L1 cache capacity. Second, the absolute number of L2 misses increases with the number of threads due to the increased cache contention, a result consistent with previous studies such as Song et al. [2007].

Therefore, it is expected that the impact of permanent faults on L2 cache performance is higher when increasing the number of threads due to the increased pressure, that is, higher miss ratios. Indeed, this behavior can be observed in Figure 9(b). Let us focus the attention on p_{fail} = 5.5e-05. When the number of threads is small (i.e., one or two threads), the impact on EMR is barely noticeable in comparison to a nonfaulty environment.[6] However, when the number of threads is moderately high (i.e., four threads and more), the EMR grows accordingly.

[6] From the point of view of EMR, both p_{fail} = 7.3e-09 and p_{fail} = 1.5e-06 provide similar results as the base case without faults.

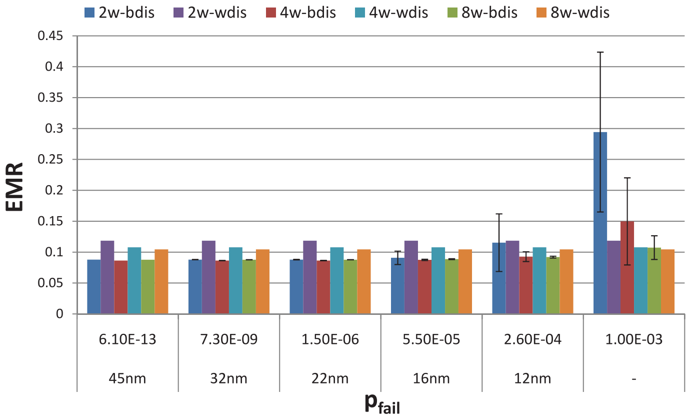


Fig. 10. EMR for block disabling and *wdis* in a 32KB L1 cache.

This trend suggests that, in a future in which applications will employ more and more threads/cores, the impact of permanent faults when using fault-tolerant techniques such as block disabling will start increasing even at small p_{fails} such as 5.5e-05.

Regarding the effect of the fault clustering parameter, in general, a higher α value provides a reduced EMR for a moderately high associativity. However, as shown in Figure 9(b), when the cache pressure is too high, that is, eight threads and 16 threads with $p_{fail} = 1.0\text{e-}03$, a lower α value provides better performance. Overall, when the number of threads and, therefore, the cache pressure, increases, the impact of p_{fails} is more noticeable and the fault clustering results in a better performance.

6.5. EMR Impact of Block Disabling and Word Disabling

In this section, we evaluate the performance implications of block disabling and word disabling (*wdis*) [Wilkerson et al. 2008]. In *wdis*, faults are tracked at word granularity by means of fault masks in the tag array. In order to provide fault tolerance, two potentially faulty cache lines are combined to obtain a fault-free one. This reduces the overall capacity and associativity of the cache by 50%.

In Figure 10, we can see the EMR of block disabling and word disabling provided by our model, for a 32KB L1 cache with a different number of ways for the SPEC applications. It is worth noticing that for word disabling EMR remains constant for the different p_{fails}.

We can also observe in Figure 10 that for p_{fails} up to 2.6e-04, block disabling obtains lower average EMR than word disabling for each configuration. However, with a p_{fail} of 1e-03, word disabling results in a lower EMR for all associativities. Furthermore, the deviation starts to grow noticeably in block disabling, whereas in word disabling it remains constant, as explained before. This result confirms the findings of Ladas et al. [2010] that are based on random fault-map methodology.

This analysis highlights how the proposed analytical model can be used as a tool for evaluating different fault-tolerant cache techniques.

7. CONCLUDING REMARKS

Technology scaling is enabling continuous miniaturization of circuits and wires, offering designers the opportunity to place more functionality per unit area. Furthermore, the increased device density is allowing the integration of large caches and many cores into the same chip. However, this trend is challenged by the decrease in resiliency with every new technology node. In particular, SRAM cells are very susceptible to decreasing voltage, higher frequencies and temperature, and other events such as power supply noise, signal cross-talking, and process variation, which, combined, severely affect the reliability of caches. Block disabling is a low-cost approach to provide correctness in the presence of faulty cache cells.

Previous proposals study the impact of block disabling due to permanent faults using an arbitrary number of random fault maps. In this article, we present an analytical model to determine the Expected Miss Ratio (EMR) and its standard deviation (SD_MR) for a given application address trace, cache configuration, and random probability of cell failure (p_{fail}) that avoids the use of random fault maps.

This analytical model enables designers to determine the implication of faults (both uncorrelated and clustered) in caches in terms of EMR and its deviation, without the need of performing any iterative analysis with random fault maps. Our experiments suggest that to provide accurate results, 100 to 1,000 fault maps are needed.

We have evaluated our model with both L1 and L2 caches with several sequential and parallel benchmarks from SPECcpu2006, SPLASH-2, and PARSECv2.1. Results show that L1 cache Yield improves with a higher clustering (due to variation) in the case of low associative caches (two ways), whereas for medium to high associative caches (four and eight ways), the best Yield is obtained when faults are uniformly dispersed across the wafer. The EMR is lower for low associative caches if there exists a high degree of clustering, whereas fault dispersion provides better results in medium to high associative caches. We have also shown that EMR has a noticeable impact on performance when p_{fail} is greater than 2.6e-4. With respect to shared-memory applications, we have shown that the EMR increases according to the number of threads. Finally, we compared block- and word-disabling techniques and found block disabling to be better performing except when p_{fail} is very high.

REFERENCES

Agarwal, A., Hennessy, J., and Horowitz, M. 1989. An analytical cache model. *ACM Trans. Comput. Syst. 7*, 2, 184–215.

Agarwal, K. and Nassif, S. 2006. Statistical analysis of SRAM cell stability. In *Proceedings of the 43rd Annual Design Automation Conference*. ACM, New York, 57–62.

Ansari, A., Gupta, S., Feng, S., and Mahlke, S. 2009. ZerehCache: Armoring cache architectures in high defect density technologies. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'09)*. 100–110.

Bienia, C., Kumar, S., Singh, J. P., and Li, K. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. ACM, New York, 72–81.

Borkar, S. 1999. Design challenges of technology scaling. *IEEE Micro 19*, 4, 23–29.

Borkar, S., Karnik, T., Narendra, S., Tschanz, J., Keshavarzi, A., and De, V. 2003. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th Annual Design Automation Conference*. ACM, New York, 338–342.

Bowman, K. A., Alameldeen, A. R., Srinivasan, S. T., and Wilkerson, C. B. 2007. Impact of die-to-die and within-die parameter variations on the throughput distribution of multi-core processors. In *Proceedings of the 2007 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'07)*. 50–55. DOI: http://dx.doi.org/10.1145/1283780.1283792

Bowman, K. A., Duvall, S. G., and Meindl, J. D. 2002. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE J. Solid-State Circuits 37*, 2, 183–190.

Bowman, K., Tschanz, J., Wilkerson, C., Lu, S.-L., Karnik, T., De, V., and Borkar, S. 2009. Circuit techniques for dynamic variation tolerance. In *Proceedings of the 46th Annual Design Automation Conference*. ACM, New York, 4–7.

Cheng, L., Gupta, P., Spanos, C. J., Qian, K., and He, L. 2011. Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 30*, 3, 388–401.

Frank, D. J. 2002. Power-constrained CMOS scaling limits. *IBM J. Res. Dev. 46*, 2/3, 235–244.

HENNING, J. L. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Comput. Archit. News 34*, 4, 1–17.

HILL, M. D. AND SMITH, A. J. 1989. Evaluating associativity in CPU caches. *IEEE Trans. Comput. 38*, 12, 1612–1630.

ISHIHARA, T. AND FALLAH, F. 2005. A cache-defect-aware code placement algorithm for improving the performance of processors. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 995–1001.

KIM, T.-H., LIU, J., KEANE, J., AND KIM, C. H. 2008. A 0.2 V, 480 kb subthreshold SRAM with 1 k cells per bitline for ultra-low-voltage computing. *IEEE J. Solid-State Circuits 43*, 2, 518–529.

KOH, C.-K., WONG, W.-F., CHEN, Y., AND LI, H. 2009. Tolerating process variations in large, set-associative caches: The buddy cache. *ACM Trans. Archit. Code Optim. 6*, 2, 8.

KOREN, I., KOREN, Z., AND STEPPER, C. H. 1993. A unified negative-binomial distribution for yield analysis of defect-tolerant circuits. *IEEE Trans. Comput. 42*, 6, 724–734.

LADAS, N., SAZEIDES, Y., AND DESMET, V. 2010. Performance-effective operation below Vcc-min. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems Software*. 223–234.

LE, H. Q., STARKE, W. J., FIELDS, J. S., O'CONNELL, F. P., NGUYEN, D. Q., RONCHETTI, B. J., SAUER, W. M., SCHWARZ, E. M., AND VADEN, M. T. 2007. IBM POWER6 microarchitecture. *IBM J. Res. Dev. 51*, 6, 639–662.

LEE, H., CHO, S., AND CHILDERS, B. R. 2007a. Exploring the interplay of yield, area, and performance in processor caches. In *Proceedings of the 25th International Conference on Computer Design*. 216–223.

LEE, H., CHO, S., AND CHILDERS, B. R. 2007b. Performance of graceful degradation for cache faults. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*. 409–415.

LEE, H., CHO, S., AND CHILDERS, B. R. 2011. DEFCAM: A design and evaluation framework for defect-tolerant cache memories. *ACM Trans. Archit. Code Optim. 8*, 3, 17.

MAGNUSSON, P. S., CHRISTENSSON, M., ESKILSON, J., FORSGREN, D., HALLBERG, G., HOGBERG, J., LARSSON, F., MOESTEDT, A., WERNER, B., AND WERNER, B. 2002. Simics: A full system simulation platform. *Computer 35*, 2, 50–58.

MARTIN, M. M. K., SORIN, D. J., BECKMANN, B. M., MARTY, M. R., XU, M., ALAMELDEEN, A. R., MOORE, K. E., HILL, M. D., AND WOOD, D. A. 2005. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Archit. News 33*, 4.

MCNAIRY, C. AND MAYFIELD, J. 2005. Montecito error protection and mitigation. In *Proceedings of the 1st Workshop on High Performance Computing Reliability Issues, in Conjunction with HPCA'05*.

NASSIF, S. R., MEHTA, N., AND CAO, Y. 2010. A resilience roadmap. In *Proceedings of the Design, Automation, and Test Conference in Europe*. 1011–1016.

PATTERSON, D. A., GARRISON, P., HILL, M., LIOUPIS, D., NYBERG, C., SIPPEL, T., AND VAN DYKE, K. 1983. Architecture of a VLSI instruction cache for a RISC. In *Proceedings of the 10th Annual International Symposium on Computer Architecture*. ACM, New York, 108–116.

POUR, A. F. AND HILL, M. D. 1993. Performance implications of tolerating cache faults. *IEEE Trans. Comput. 42*, 3, 257–267.

ROBERTS, D., KIM, N. S., AND MUDGE, T. 2007. On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*. 570–578.

SÁNCHEZ, D., SAZEIDES, Y., ARAGÓN, J. L., AND GARCIA, J. M. 2011. An analytical model for the calculation of the expected miss ratio in faulty caches. In *IOLTS*. 252–257.

SHIRVANI, P. P. AND MCCLUSKEY, E. J. 1999. PADded cache: A new fault-tolerance technique for cache memories. In *Proceedings of the 17th IEEE VLSI Test Symposium*. IEEE Computer Society, 440–445.

SOHI, G. S. 1989. Cache memory organization to enhance the yield of high performance VLSI processors. *IEEE Trans. Comput. 38*, 4, 484–492.

SONG, F., MOORE, S., AND DONGARRA, J. 2007. L2 cache modeling for scientific applications on chip multiprocessors. In *Proceedings of the 2007 International Conference on Parallel Processing (ICPP'07)*. 51.

STAPPER, C. H., ARMSTRONG, F. M., AND SAJI, K. 1983. Integrated circuit yield statistics. *Proc. IEEE 71*, 4, 453–470.

TAUR, Y. 2002. CMOS design near to the limit of scaling. *IBM J. Res. Dev. 46*, 2/3, 213–222.

UNSAL, O. S., TSCHANZ, J. W., BOWMAN, K., DE, V., VERA, X., GONZALEZ, A., AND ERGIN, O. 2006. Impact of parameter variations on circuits and microarchitecture. *IEEE Micro 26*, 6, 30–39. DOI: http://dx.doi.org/10.1109/MM.2006.122.

VERMA, N. AND CHANDRAKASAN, A. P. 2008. A 256 kb 65 nm 8T subthreshold SRAM employing sense-amplifier redundancy. *IEEE J. Solid-State Circuits 43*, 1, 141–149.

WILKERSON, C., GAO, H., ALAMELDEEN, A. R., CHISHTI, Z., KHELLAH, M., AND LU, S.-L. 2008. Trading off cache capacity for reliability to enable low voltage operation. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*. 203–214.

WOO, S. C., OHARA, M., TORRIE, E., SINGH, J. P., AND GUPTA, A. 1995. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*. ACM, New York, 24–36

YAMAOKA, M., OSADA, K., TSUCHIYA, R., HORIUCHI, M., KIMURA, S., AND KAWAHARA, T. 2004. Low power SRAM menu for SOC application using Yin-Yang-feedback memory cell technology. In *Proceedings of the Symposium on VLSI Circuits*. 288–291.

YAO, S. B. 1977. Approximating block accesses in database organizations. *Commun. ACM 20, 4,* 260–261.

ZHANG, K., BHATTACHARYA, U., CHEN, Z., HAMZAOGLU, F., MURRAY, D., VALLEPALLI, N., WANG, Y., ZHENG, B., AND BOHR, M. 2004. SRAM design on 65nm CMOS technology with integrated leakage reduction scheme. In *Proceedings of the Symposium on VLSI Circuits*. 294–295.

Received February 2013; revised June 2013; accepted July 2013