The performance vulnerability of architectural and non-architectural arrays to permanent faults

Damien Hardy*†, Isidoros Sideris*, Nikolas Ladas*, Yiannakis Sazeides*
*University of Cyprus †University of Rennes 1, IRISA, France

Abstract

This paper presents a first-order analytical model for determining the performance degradation caused by permanently faulty cells in architectural and non-architectural arrays. We refer to this degradation as the performance vulnerability factor (PVF).

The study assumes a future where cache blocks with faulty cells are disabled resulting in less cache capacity and extra misses while faulty predictor cells are still used but cause additional mispredictions.

For a given program run, random probability of permanent cell failure, and processor configuration, the model can rapidly provide the expected PVF as well as lower and upper PVF probability distribution bounds for an individual array or array combination.

The model is used to predict the PVF for the three predictors and the last level cache, used in this study, for a wide range of cell failure rates. The analysis reveals that for cell failure rate of up to 1.5e-6 the expected PVF is very small. For higher failure rates the expected PVF grows noticeably mostly due to the extra misses in the last level cache. The expected PVF of the predictors remains small even at high failure rates but the PVF distribution reveals cases of significant performance degradation with a non-negligible probability.

These results suggest that designers of future processors can leverage trade-offs between PVF and reliability to sustain area, performance and energy scaling. The paper demonstrates this approach by exploring the implications of different cell size on yield and PVF.

1. Introduction

For the past 50 years, technological advances have enabled 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 device area has been accompanied by at least two negative consequences: a slowdown of voltage scaling and frequency, due to slower scaling of leakage current as compared to area scaling [34], and a shift to probabilistic design and less reliable silicon primitives due to variations [6, 9].

Technology trends suggest that in tomorrow's computing world failures will become commonplace due to many and/or frequent causes such as: static [6] and dynamic [9] variations, latent-faults due to limited burn-in [35], higher temperatures and accelerated wear-out [33], near subthreshold operation [36], etc. A recently published resilience roadmap [24] underlines the expected increase of probability of failure (*pfail*) with scaling. Table 1 shows the pfail predicted in [24] for inverters, latches and SRAM cells due to random dopant fluctuations as a function of technology node (the trends for negative-bias-temperature-instability are similar). The table shows that the pfail of all types of devices increases dramatically with scaling. However, the table clearly indicates that not all types of devices are equally vulnerable; SRAM cells that are usually aggressively sized are distinctively more problematic.

A variety of well known approaches, such as column/row sparing and error correcting codes [29], have been adopted to provide fault-free chips by mitigating manufacturing faults and static parametric variations while preserving an important abstraction: *performance-invariability*. This is defined to be the expectation that two identical chips, or even two cores within a chip, when each is operating stand-alone under identical conditions have identical cycle by cycle behavior and, therefore, identical performance.

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

Table 1: Predicted probability of failure (pfail) for different types of circuits vs technology node [24]

The central hypothesis of this paper is that the abstraction of performance-invariability is unrealistic to preserve for future processor chips due to the non-scalable cost of existing techniques to mitigate the mismatch between the scaling rates of area, voltage and static and dynamic variations. Replacing a chip and coarse grain disabling (e.g. core, cache bank/ways) may be acceptable when variability phenomena are rare but with increasing static and dynamic variations finer disabling and deconfiguration (e.g. individual functional units, cache blocks) will become economically necessary. We are going, therefore, to enter an era of *performance-variability*: functionally correct chips with variable degraded performance, intra-chip and across-chips, from the time when parts are shipped. The performance-variability will be due to lower resource capacity caused by the disabling of faulty cache blocks or units [32, 21, 36], timing misspeculation in the datapath due to timing variations [13], and extra mispredictions caused by faulty predictor cells [20].

In this paper, we propose a first-order analytical model for understanding the implications on performance of permanently faulty cells in architectural and non-architectural arrays. The model for a given program execution, micro-architectural configuration, and probability of cell failure, provides rapidly the *Performance Vulnerability Factor (PVF)*. PVF is a direct measure of the performance degradation due to permanent faults. In particular, the model can determine the expected PVF as well as the PVF probability distribution bounds for caches and prediction arrays without using an arbitrary number of random fault-maps.

PVF is analogous to the Architectural Vulnerability Factor (AVF) [23] proposed for assessing the vulnerability of architectural structures to soft-errors. PVF like AVF can be used by designers/researchers to appreciate and compare vulnerability of different structures and perform reliability driven trade-offs. However, PVF is complementary to AVF as it measures performance degradation due to permanent errors.

Virtually all previous micro-architectural work aiming to assess the performance implications of permanently faulty cells relies on simulations with random fault-maps, assumes faulty blocks are disabled [32, 21, 4, 1], and focuses on architectural arrays such as caches. These studies are, therefore, limited by the fault-maps they use that may not be representative for the average and distributed performance. Moreover, they are incomplete by ignoring faults in non-architectural

arrays, such as predictors, that do not affect correctness but can degrade performance. A narrow understanding of the consequences of permanently faulty cells can lead to a processor fault-reliability strategy that neither addresses key reliability challenges nor leverages reliability driven trade-offs.

A recent relevant study [27] proposes a method that given a cache, random probability of permanent cell failure, and an address trace, can calculate *without any fault-maps* the expected miss ratio when blocks with permanent faults are disabled. The model proposed in our paper, augments the method in [27], to predictors, can produce for caches and predictors the miss-rate and misprediction distribution bounds respectively, and provides the expected performance and performance distribution bounds for individual or combination of faulty arrays.

The paper establishes that for three predictors (a return address stack, a gshare direction predictor and an indirect jump predictor) and the last level cache used in this study, the proposed model assumptions are valid. The model is used to predict, for the same arrays and processor, the expected PVF and PVF distribution bounds at different technology nodes using projected cell failure rates due to random dopant fluctuations.

The experimental analysis shows that for cell pfail of up to 1.5e-6 the expected PVF is very small for any array or array combination. For higher pfail the PVF grows considerably mainly due to the extra misses in the last level cache. The expected PVF of all predictors remains small even at high pfail, but the PVF distribution reveals cases of significant performance degradation with small but noticeable probability. These results suggest that in the future designers can leverage trade-offs between PVF and reliability to improve efficiency. The paper demonstrates this approach by investigating the impact of different cell size on yield and PVF.

The remainder of the paper is organized as follows. Section 2 introduces the notion of PVF. The model that determines PVF is presented in Section 3. Experimental results are given in Section 4. Section 5 reviews related work and Section 6 concludes the paper and gives direction for future work.

2. Performance Vulnerability Factor - PVF

The *Performance Vulnerability Factor (PVF)* is a measure of the performance degradation that a processor will experience for a given benchmark run due to permanently faulty cells in its arrays. PVF takes values in the range $[0,1)$, with zero meaning no vulnerability at all (100% of performance) and for values near 1 almost zero performance. To define PVF, we rely on the notion of *Computation Capacity* (CC) as follows:

$$PVF = 1 - CC \tag{1}$$

where the computation capacity [5] is the fraction of the original pristine machine's performance that is still available given current hardware conditions and defined in our work as:

$$CC = \frac{C_{base}}{C_{base} + overhead * \#failures} \tag{2}$$

where C_{base} corresponds to the number of cycles of a fault-free run of a program by the microarchitecture under study, $\#failures$ is the number of failures, e.g. misses or mispredictions, resulting from permanently faulty cells and, *overhead* represents the mean increase in cycles of the overall execution time per failure.

For mathematical convenience, we introduce the notation ETV (Execution Time Vulnerability) which represents the normalized execution cycle increase of a program due to faults in arrays and defined as:

$$ETV = penalty * \#failures \tag{3}$$

where $penalty = \frac{overhead}{C_{base}}$ and corresponds to the normalized overhead of a single failure to the overall cycle count of a fault free run. By substituting ETV in Eq. 2 the computation capacity becomes:

$$CC = \frac{1}{1 + ETV} \tag{4}$$

PVF and expected PVF (noted $\overline{PVF}$) can be determined directly from ETV and $\overline{ETV}$ by using Eq. 1 and 4.

In the following section we describe a model that can determine the ETV, $\overline{ETV}$ and ETV distribution bounds for caches and prediction arrays.

3. ETV for non-architectural and architectural arrays

This section presents a first-order analytical model for predicting the ETV of an individual or combination of non-architectural and architectural arrays. First we present the model assumptions and then introduce the basic model. We then present how to use the model for non-architectural and architectural arrays. In particular, we focus on how to obtain the mean penalty per failure and the number of failures for each array. After, we describe a method to determine lower and upper ETV probability distribution bounds. At the end of the section, we discuss model limitations, extensions and uses.

3.1. Model Fault Assumptions

The model assumes that permanently faulty SRAM cells locations are random, each cell has equal probability of failure, and broken cells behave as stuck-at faults (the stuck-at assumption is only needed for predictors). The random behavior aims to capture some major causes of uncorrelated faults, such as line edge roughness and random dopant fluctuation. Also, the granularity of spatial correlation is large and within a chip the failures can be treated as uncorrelated [10].

The cells in processor can be divided into the following categories:

- can be faulty and not disabled - e.g. predictor bits
- can be faulty and must be disabled - e.g. cache block data bits
- cannot be faulty (fault free or replaced with a spare) - e.g. PC bits, block disable bit

The third category affects correctness and yield. Our model focuses on the first two categories, which affect performance. Based on the above cell classification, faulty entries in prediction arrays are used normally and can lead to extra mispredictions. In caches, on the other hand, blocks with faults are disabled and thus reduce the array capacity and lead to extra misses. For caches, an entry (i.e. a cache block) with at least one permanent fault is considered as faulty. Faulty cache blocks are assumed to be detected with post-manufacturing and boot-time tests, ECC, and built in self tests. Cache block disabling has been proposed for commercial processors in [22].

The model does not address the effect of rarely occurring transient errors, e.g. due to particle strikes because their effects are rare and short lived. In fact, they will cause at most a handful of mispredictions in prediction arrays and a repair when captured by ECC in caches.

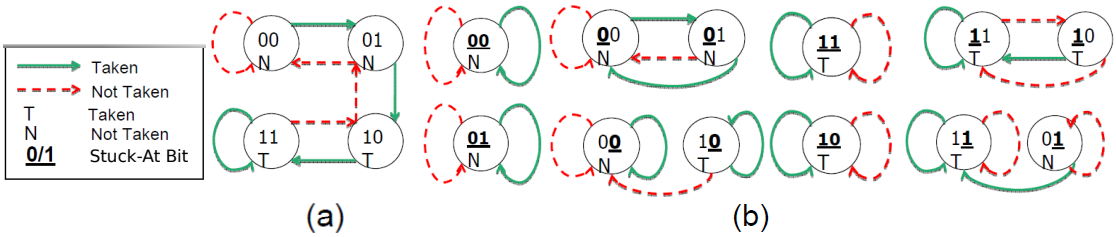

Figure 1: Operation of a 2-bit saturating counter used for prediction (a) Fault Free, (b) With Faults

3.2. Basic Model

At the core of determining PVF is the computation of ETV (Eq. 3) which can be obtained for a given program run by:

$$ETV = penalty * \#failures = \sum_{i=1}^{u} \sum_{j=1}^{M_i} P_{ij} \quad (5)$$

where u is the number of arrays with faults, M_i is the number of failures caused by faults in array i, and P_{ij} is the penalty for the jth failure in the program from array i. M_i corresponds to additional mispredictions (faulty-mispredictions) in case the unit i is a predictor, and additional cache misses (faulty-misses) in case i is a cache. Eq. 5 can be simplified to:

$$ETV = \sum_{i=1}^{u} ETV_i = \sum_{i=1}^{u} M_i \overline{P_i} \quad (6)$$

where ETV_i is the ETV of unit i and $\overline{P_i}$ is the mean normalized penalty for each failure in array i. Finally, the expected ETV_i of unit i can be obtained from:

$$\overline{ETV_i} = \overline{M_i} * \overline{P_i} \quad (7)$$

This is the same equation as Eq. 6 except $\overline{M_i}$ that represents the expected number of faulty-mispredictions or faulty-misses due to unit i.

The above model assumes that ETV is additive, i.e. we determine the ETV_i of each array i separately and simply sum them together to obtain the overall ETV. We discuss in Section 3.5 why this can be accurate.

Next we describe how to determine for non-architectural and architectural arrays: (i) $\overline{P_i}$, (ii) M_i when given a fault-map, (iii) $\overline{M_i}$ when given a random cell pfail but without fault-maps, and (iv) distribution bounds of ETV when given a random cell pfail without fault-maps.

3.3. ETV for non-architectural arrays

The focus of this part is on the performance implications of *permanently faulty SRAM cells* in non-architectural arrays. Today's high-end processors employ several non-architectural arrays for improving performance. Such arrays are used to predict: next instruction line, direction for conditional branches, target for branches (especially for return and indirect branches), addresses for prefetching, etc. In addition, non-architectural arrays are used to guide the updating of predictors (hysteresis bits).

Although faults can occur in any of these arrays, the performance implications of a faulty cell vary depending on how the value in a faulty cell is used in a pipeline and, in particular, on how a faulty cell influences performance. For example, a faulty cell in a conditional predictor, a return-address-stack and an indirect-jump predictor can cause a misprediction and a pipeline flush and, therefore, have a large misprediction penalty. In contrast, a faulty line-predictor cell has a smaller misprediction penalty because a line-predictor is usually corrected by a more accurate predictor within one or two cycles.

A faulty cell can have different implications depending on its semantic functionality. For example, a stuck-at fault in an 1-bit hysteresis, used to guide replacement on a misprediction, can lead to two behaviors: always replace or never update. Both behaviors can degrade performance but the second can be more grave. Fig. 1 illustrates how different combinations of stuck-at fault(s) at different bit positions transform the operation of a 2-bit saturating counter used for direction prediction. It is interesting to observe that such faults mainly result in always taken or always not-taken behavior.

Another key parameter that influences the vulnerability of a non-architectural array to faults is the distribution of accesses. A permanent fault in a frequently accessed entry is likely to cause more degradation than a fault in an infrequently accessed entry. Also, a program with many "hot" entries has higher probability to have a faulty hot entry. However, for the same number of accesses the more the hot entries the lower the impact from each.

The bias of the values stored in a non-architectural array also influences the performance with faults. For instance, an entry that contains bits that are highly biased will experience worst case degradation when at least one of these bits is stuck-at in the opposite value of the bias.

The above discussion reveals that performance is not equally vulnerable to faults across non-architectural arrays and even within an array. Furthermore, there is variation due to differences in the dynamic behavior between programs, such as the access distribution, instruction mix, predictability and misprediction penalty. For example, it is obvious that a program with or with low branch predictability will not suffer significantly from faults in the predictors.

Below we define the basic analytical model for determining the performance impact of permanently faulty cells in non-architectural arrays.

3.3.1. Model for non-architectural arrays

According to Eq. 6, the ETV_i, of a non-architectural array i, can be obtained from M_i and $\overline{P_i}$. $\overline{P_i}$ can be obtained by linear regression of the additional cycles and additional mispredictions from runs that inject randomly stuck-at faults in unit i[1]. It is known from previous work [15] that misprediction penalty is sensitive to the number of mispredictions, but our approximation for $\overline{P_i}$ is found to be accurate for most benchmarks and non-architectural arrays we considered in this study. If more accuracy is needed future work can consider models that correlate penalty to other parameters [15].

The M_i for a non-architectural array i and a program run, can be obtained using the *access-map* and *bias-maps* of the array. An *access-map* represents the number of correct predictions from each entry

[1]We used ten runs for each of the following fractions of faulty bits in unit i: 0.5, 2 and 8%

	Bias0	Bias1
0	30	12
1	6	47
2	23	30
3	15	15

	Fmap0	Fmap1
0	0	0
1	0	1
2	1	0
3	0	0

Figure 2: M_i computation example for a 4 entry 1bit/entry predictor. The predictor has a stuck-at-1 bit in entry 1 resulting in 6 extra misspredictions (Bias0 map) and a stuck-at-0 bit in entry 2 resulting in 30 extra misspredictions (Bias1 map).

during a fault free run of the program. The access map is further broken into *bias-maps*, *bias0* and *bias1*, that indicate for each bit in the array, during the fault free run, how many times it is part of a correct prediction with value 0 and 1 respectively.

Next we explain how to use the access and bias maps to obtain the M_i when given a fault-map, and to determine the expected $\overline{M_i}$ and $\overline{ETV_i}$ when given a cell pfail.

3.3.2. M_i from a fault-map

We can obtain M_i for a specific fault-map, that indicates whether a bit is faulty and its stuck-at value, by taking effectively the dot product of the fault-map and the bias-maps. This is illustrated in Figure 2 for a single-bit per entry predictor. For a 2-bit saturating predictor, this approach works as well because, as shown in Figure 1, a stuck-at bit in such a predictor, can lead to always taken or always not-taken predictions.

We limit the fault-maps used for non-architectural arrays to at most one faulty-bit per entry to keep the bias-maps simple, as multiple faulty bits in an entry are very unlikely for the pfail range considered in this work.

3.3.3. Expected ETV_i and M_i for a given cell pfail

The expected ETV_i, for a benchmark running on a processor with faulty cells in a non-architectural array i, can be obtained from Eq. 7. The term $\overline{M_i}$ represents the expected number of mispredictions due to faults in unit i. $\overline{M_i}$ can be obtained probabilistically for a given cell pfail without using fault-maps as follows:

$$\overline{M_i} = \frac{\sum_{j=1}^{entries_i} access\_map_j}{2} * \overline{pfail\_entry_i} \tag{8}$$

where $entries_i$ represents the total number of entries in predictor array i, $access\_map_j$ is the number of correct predictions of entry j during a fault free run, and $\overline{pfail\_entry_i}$ is the expected probability of having a faulty entry determined as follows for an n bit entry:

$$1-(1-pfail)^n \tag{9}$$

The halving in Eq. 8 captures the probability of a fault being stuck-at 0 or 1.

The non-architectural PVF model is valid as long as the access-map and bias-maps of an array, for a given program run, are virtually insensitive to pipelining effects, i.e. the number of correct predictions is insensitive to pipelining effects. Fortunately, this is the case for prediction arrays that are not speculatively updated or when they are speculatively updated they are repaired from wrong path effects [18, 31].

Fault Map		LRU+3 (MRU)	LRU+2	LRU+1	LRU
1	SET 0	120	100	110	60
0	SET 1	150	140	110	55
3	SET 2	180	134	80	50
2	SET 3	220	200	100	30

Figure 3: M computation example for a 4 set 4 ways LRU cache. $M = 454$ resulting from 1 faulty block in set 0, 3 faulty blocks in set 2 and 2 faulty block in set 3.

3.4. ETV for architectural arrays

Nowadays, caches take most of the real-estate in processors and contain numerous SRAM cells. As explained before, we assume that blocks that contain permanently faulty cells are disabled and thus reduce the cache capacity.

Architectural arrays vulnerability, similar to non-architectural, depends on the distribution of hit accesses across sets and within sets and cache miss latency. Consequently, the PVF for architectural arrays can vary across programs and across sets, and, it depends on the dynamic program behavior. The model detailed hereafter is proposed to assess the performance impact of permanently faulty SRAM cells in architectural arrays.

3.4.1. Notations and Assumptions

A cache configuration is defined by the number of sets s, ways per set n, and block size in bits k. The model is defined for the LRU replacement policy, the generalization of the model to other replacement policies is left for future work.

3.4.2. Model for architectural arrays

The ETV_i, caused by faulty blocks in cache i, can be obtained using Eq. 6. For caches, M_i corresponds to the number of additional misses caused by faulty blocks, and $\overline{P_i}$ is the average normalized penalty per additional miss. $\overline{P_i}$ can be determined by linear regression using additional cycles and additional misses from handful runs[2].

The number of additional misses M_i for a program run is determined by using the *access-map* of the cache during a fault free run of the program. An *access-map* represents the number of hits per set and per LRU position during a fault free run. In the access-map, a row corresponds to a set and each column corresponds to a position in the LRU stack.

Next we present how to use cache access maps to obtain the M_i when given a cache fault-map, and to determine the expected $\overline{M_i}$ and expected $\overline{ETV_i}$ when given a cell pfail.

3.4.3. M_i from a fault-map

Given an access-map, we can obtain M_i for a specific fault-map of cache i, which indicates the number of faulty blocks w per set, by simply adding the accesses to the last w columns as illustrated in Figure 3 and suggested by [25]. It can be noticed that, thanks to the LRU replacement policy, the exact position of the faulty-blocks are not needed because the LRU stack is reduced by the number of faulty blocks in each set.

3.4.4. Expected ETV_i and M_i for a given cell pfail

The expected ETV_i, for a benchmark running on a cache with faulty cache blocks, can be obtained from Eq. 7. The term $\overline{M_i}$ represents the expected number of additional misses due to faulty blocks. $\overline{M_i}$

[2]We used n (associativity of the cache) runs by considering 1 way faulty in each set, 2 ways faulty etc.

can be obtained analytically for a given cell pfail without using fault-maps as proposed in [27][3]. To determine $\overline{M_i}$, we first compute the probability of a cache block failure p_{bf} with Eq. 9 for k bits. Then, the probability pe_i for i faulty ways in a set can be determined based on the well-known binomial probability:

$$pe_i = \binom{n}{i} p_{bf}^i (1-p_{bf})^{n-i} \tag{10}$$

which can provide, for every value of $i[0\ldots n]$, the probability of having i faulty ways. This distribution provides insight about how likely it is to have a given number of faulty ways in a set and, can be used to obtain the expected number of additional misses. The expectation of a random variable $X = x_1, x_2, \ldots, x_n$ for which each possible value has probability $P = p_1, p_2, \ldots, p_n$ is given by:

$$E[X] = \sum_{i=1}^{n} x_i * p_i \tag{11}$$

In our case, the random variable X corresponds to the total number of additional misses in a cache set with faults, x_i corresponds to the total number of additional misses when there are i faulty ways in a set, noted hereafter $m_{i,set}$, and p_i the probability of having i faulty ways in a set. $m_{i,set}$ is simply determined by adding the last $n-(i-1)$ columns of the access map of the set:

$$m_{i,set} = \sum_{j=n-(i-1)}^{n} access\_map[set][j] \tag{12}$$

Therefore, the expectation of the number of additional cache misses $\overline{M_{set}}$ for a given set can be expressed as follows:

$$\overline{M_{set}} = \sum_{i=1}^{n} m_{i,set} * pe_i \tag{13}$$

Finally, we sum this expectation for each set to get the expected number of additional misses $\overline{M_i}$.

We note that the ETV model requires that the access-map and the number of cache hits per set and LRU position, for a given program run, to be rather insensitive to pipelining effects. Our empirical analysis presented in Section 4 reveals that this assumption is true.

3.5. Why are ETV_is additive?

Eq. 6 assumes that ETV_i for each array i is additive when computing the overall ETV. It is known from previous work [30, 14] that, the penalty of different units can overlap, thus, how can this additive claim be accurate?

The proposed method for the penalty estimation of an array i considers the overall execution increase due to additional misses or mispredictions while it captures the overlapping and interactions with other pipelining effects and events as well as the non-constant main memory latency. Specifically, the penalty $\overline{P_i}$, for an array is obtained by activating faults while in the pipeline there can be other concurrent misses, mispredicts, stalls, memory accesses etc.

The simple summation of ETV_is is found to be accurate for the benchmarks, faulty arrays, and microarchitecture used in this study. If more accuracy is needed, future work can consider the detail interaction between units [30, 14].

[3]The model used in this section to estimate $\overline{M_i}$ is from [27] and is presented here for completeness.

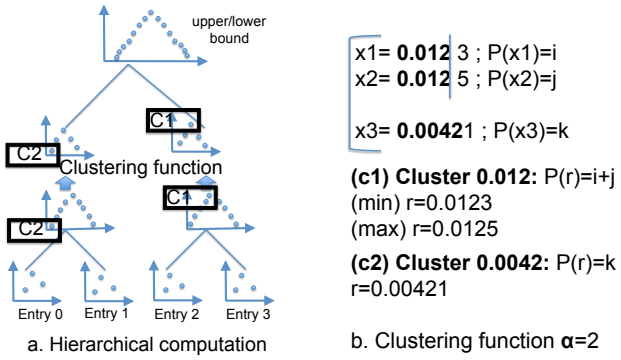

Figure 4: Hierarchical distribution computation and clustering function example.

3.6. ETV probability distribution bounds

To draw more insight about the PVF, we introduce an analysis that determines the lower and upper PVF probability distribution bounds for an individual or any combination of arrays for a given cell $pfail$. These bounds are computed in two steps: (i) estimation of a discrete ETV probability distribution per entry for each unit i, (ii) combining the multiple discrete ETV probability distributions by using a heuristic to determine a lower bound and an upper bound of the distribution. Although, not discussed further, the heuristic can be trivially used to produce the M_i probability distribution bounds by using Eq. 6.

Estimation of a discrete probability distribution. For predictors, a discrete ETV probability distribution is first determined per entry. For each entry, this distribution is determined by the sum of probabilities that can cause a given ETV by considering the following cases:

- $ETV = 0$ which corresponds to a fault free entry. The corresponding probability is $1 - \overline{pfail\_entry_i}$
- ETV_{bias0}[4] which corresponds to the ETV caused by a stuck-at-1 bit in the entry. The corresponding probability is $\overline{pfail\_entry_i}/2$. The ETV_{bias0} can be obtained using Eq. 6.
- ETV_{bias1} which corresponds to the ETV caused by a stuck-at-0 bit in the entry. The corresponding probability is $\overline{pfail\_entry_i}/2$.

For a cache, the discrete ETV probability distribution is determined at the granularity of a set. We obtain at most $associativity+1$ different ETVs that correspond to 0 up to all blocks faulty in a set with the probability for each case given by pe_i.

Combining multiple discrete probability distributions. To determine the resulting ETV probability distribution of an array or combination of arrays, we need to combine together all the previously computed distributions. It is well known that exhaustive combining is infeasible due to a combinatorial explosion [16]. To avoid this problem we propose a parametric (α) heuristic to determine a lower bound and an upper bound of the ETV distribution.

A hierarchical approach [16] is used based on a binary tree representation as illustrated in Figure 4.a. Each leaf represents the distribution of an entry for predictors or the distribution of a set in a cache. The root represents the resulting upper/lower distribution bound and each intermediate node an intermediate joint distribution.

[4]For arrays that predict multi-bit values, like the return address-stack, ETV is determined for each bit and the probability is further divided by the number of bits per entry. This approach is similarly used for ETV_{bias1}

To avoid the combinatorial explosion, after obtaining each intermediate distribution (at each intermediate node), we use a clustering approach by summing probabilities when ETV values are close enough.

This strategy uses a clustering function as illustrated in Figure 4.b. It starts with a truncation of each value (i.e. ETV) after the first α non-zero digits[5]. Equal resulting values are then grouped together and their respective probability are summed. The value r representative of each group is the minimal/maximal value of the group before the truncation. This ensures (see the proof in appendix) a lower/upper bound ETV probability distribution.

3.7. How many runs are needed to determine the input values of the analytical model?

In this section, we summarize the number of runs needed by the analytical model to determine the expected PVF and the PVF distribution bounds. We like to note that the PVF analysis assumes a fixed microarchitecture. Therefore, for each distinct microarchitecture a separate PVF analysis is required.

For each array under study the model needs information provided by a single run without faults for each program that is analyzed. This fault-free run is performed to collect the *Cbase*, program baseline performance, and the access-maps of each array. The access-maps obtained for predictors and caches are then used to compute the expected PVF and PVF distribution bounds.

The mean penalty, $\overline{P_i}$, for a unit i is obtained per program by running it on the given microarchitecture with increasing number of faults in the unit i. Typically k_i (=30 in this study[6]) runs for each predictor and as many runs as the number of ways, w_i, in cache i are sufficient to observe how the overall execution increases as a function of additional faulty mispredictions and misses.

Therefore, for one benchmark the total number of runs needed to determine the input values for the analytical model is given by:

$$1 + \sum_{i=1}^{\#arch\_units} w_i + \sum_{i=1}^{\#non\_arch\_units} k_i$$

So assuming 26 benchmarks, 3 predictors and an 8 way cache we need to perform 2574 simulations to completely characterize the microarchitecture.

The model key strengths is that once the characterization is complete, the model does not require performing new simulations when changing cell pfail. It can very fast and accurately obtain the expected PVF (few seconds) and the PVF distributions (1 minute on a current machine) for different pfails.

To put in perspective, to produce the model results using random fault maps and simulations will require exponential to the number of entries and sets runs for each unique pfail. The benefits of our analytical model as compared to current practice depend on the number of distinct pfail to consider and the number of fault-maps to simulate. The proportion of runs needed between our model and current practice for a benchmark is:

$$\frac{1 + \sum_{i=1}^{\#arch\_units} w_i + \sum_{i=1}^{\#non\_arch\_units} k_i}{\#pfails * \#faultmaps}$$

The denominator is the number of runs for current practice and the numerator is the number of runs needed to determine the input values of the model (100 is an upper bound in our study). Our cost is thus a small number of runs whereas a random fault map methodology may require huge amount of runs to obtain the expected PVF and its distribution.

3.8. Model limitations

The model is a first-order approximation and can underestimate the PVF in some cases. In particular, for prediction arrays when multiple bits of the same entry happen to be faulty and their combined mispredictions is more than their individual contributions, then the expected PVF is underestimated. For arrays with 1-bit entries or 2-bit saturating counters this is not a problem since the faulty mispredictions are determined uniquely by the faulty bits, the faulty value and the bias (see Fig. 1) and, therefore, M_i can be estimated at the granularity of an entry. For arrays that predict multi-bit values, like the return address-stack and the indirect jump predictor, this is not a problem as long as the fault-maps are random and the number of faults is small enough that render very unlikely to have more than one faulty bit in a frequently accessed entry. These assumptions are reasonable for the configuration and parameters used in this study (in particular for pfail $\leq$0.001).

It is possible that faults in the prediction arrays and caches can result in a significant increase in the cache accesses and misses that may not be captured by our model. We have not observed such behavior for the pfails considered in this study.

All these limitations are directions for future model improvements if higher model accuracy is required.

3.9. Model Extensions

The model, in its current version, is defined for three predictors and a cache of a single core. The model, however, should be applicable to other arrays as long as we can derive a method to obtain the number of extra events due to faults and their corresponding penalties. For non-arrays, like functional units, we believe that the model can be extended based on the probability of timing violations [13, 19]. Finally, for multi-cores the impact on performance of faults in shared resources, such as caches and interconnect, will need to be modeled while also considering the interactions between benchmarks.

With the currently modeled structures, we observe that the performance degradation depends on the penalty and on the number of accesses to a structure. The more accessed an entry is and the higher the penalty of an extra miss or misprediction, the higher the performance degradation when the entry is faulty. We believe that this observation will also hold for other structures.

3.10. Model Uses

The model, once derived, can be used to explore processor behavior with different cell pfail. This can be helpful to forecast how processor performance may be affected by faults in the future. Additionally, this information can be useful to explore the use of different cell sizes that enable a trade-off between area and PVF. Another use of the model is to determine which arrays have significant PVF and make design decisions to reduce their PVF, for example through a protection mechanism, using larger cells, or even by selecting a different array organization. The PVF distributions bounds can help establish how a population of chips is affected due to faults in predictors and caches. Such binning provides an indication about

[5] α is a parameter that determines the precision and allows to trade-off accuracy for computation time.

[6] k_i is determined empirically. Furthermore, k_i has been validated by a sensitivity analysis over the penalty values, which reveals that in most of the cases there is a range of values around the determined penalties that provides the same PVF.

Parameter description	Setting
Pipeline depth	15 stages
Fetch/Decode/Issue/Commit	up to 4/4/6/4 instr. per cycle
Line Predictor	4096 entries
RAS	16 entries (31 bits per entry) x 2 (fetch and commit)
Indirect Jump Predictor	512 entries (31 bits per entry)
Branch Predictor	8 KB gshare (32768 entries - 2 bits per entry, 15 bits history)
Branch Resolution	In-order
Issue Queue/Reorder buffer	40 INT entries, 20 FP entries/128 entries
Functional Units	4 INT ALUs, 4 INT mult/div, 1 FP ALUs, 1 FP mult/div
L1 instr./data cache	64 KB, 2-way, 64 B blocks, 1-cycle, LRU / 64 KB, 2-way, 64 B blocks, 3-cycle, LRU
L2 unified cache	2 MB, 8-way, 64 B blocks, 12-cycle hit latency, 255 cycles miss latency, LRU

Table 2: Processor Configuration

Benchmark	Baseline	Conditional	Returns	Indirect	L2 cache	Accuracy			Overheads			
	IPC	branches (M)		jumps	MPKI	Gshare	RAS	Ijump	Gshare	RAS	Ijump	L2 cache
ammp00	1.54	8.8	21K	0	1.19	0.97	1.00	0	20	25	0	58
applu00	0.52	0.6	100	94	22.04	0.99	0.99	0.68	33	41	0	51
apsi00	2.35	3.5	58K	0	0.82	0.99	1.00	0	25	29	0	187
art00	1.82	8.6	110	0	0.25	0.99	0.99	0	45	27	0	70
bzip00	1.29	14.4	353K	0	0.82	0.95	1.00	0	17	18	0	62
crafty00	1.90	8.7	1.09M	209K	0.17	0.96	0.99	0.66	24	24	25	66
eon00	1.26	7.0	2.04M	571K	0.01	0.99	1.00	0.76	14	16	13	48
equake00	0.64	1.7	1.06M	0	12.73	0.97	1.00	0	39	28	0	55
facerec00	1.58	6.7	166K	0	4.22	0.98	1.00	0	23	24	0	43
fma3d00	1.43	16.3	1.43M	278K	0.04	0.96	1.00	0.83	19	25	25	76
galgel00	2.42	5.8	0	0	0.24	0.99	0	0	29	0	0	55
gap00	1.56	9.5	2.05M	1.53M	1.01	0.99	0.99	0.77	23	25	26	38
gcc00	1.17	6.9	478K	213K	4.02	0.97	1.00	0.59	17	27	34	74
gzip00	1.54	7.0	1.05M	13	0.17	0.94	1.00	0.77	20	20	0	94
lucas00	0.70	1.3	0	0	13.39	1.00	0	0	9	0	0	40
mcf00	0.13	20.4	3.33M	0	86.42	0.96	1.00	0	58	68	0	60
mesa00	1.73	5.9	1.19M	547K	0.20	0.97	1.00	0.99	29	25	27	35
mgrid00	0.82	0.4	327	154	10.58	0.99	0.99	0.90	16	0	0	66
parser00	1.19	12.3	1.99M	240	1.37	0.96	0.99	0.90	20	21	0	82
perlbmk00	1.28	9.2	2.11M	1.54M	0.19	0.99	1.00	0.77	21	16	23	48
sixtrack00	1.96	2.3	128	24K	0.29	0.99	1.00	0.80	28	34	38	50
swim00	0.40	2.3	66	46	26.36	0.99	1.00	0.57	24	23	0	45
twolf00	1.06	10.6	705K	0	0.25	0.90	1.00	0	20	24	0	86
vortex00	1.86	10.8	2.06M	79K	0.34	0.99	0.99	0.95	23	24	23	81
vpr00	0.72	9.7	647K	49	6.13	0.94	1.00	0.93	23	31	0	72
wupwise00	1.78	8.8	652K	18	2.79	0.99	1.00	0	25	17	0	85

Table 3: Benchmark Characteristics

how many chips will be affected by failures and up to what extent. Finally, the PVF analysis can provide a first order timing analysis for systems with performance constraints such as real time systems.

The above types of studies are facilitated by the proposed methodology, since being analytical it allows for quick exploration, without long micro-architectural simulation or fault maps generation.

4. Experimental Results

4.1. Experimental setup

In our experiments, the validated cycle accurate simulator *sim-alpha* [12] is used. We have extended it to measure the performance implications of faults in three prediction arrays: a return address stack, a gshare direction predictor and an indirect jump predictor; and the L2 cache of a high performance out-of-order superscalar processor. Key parameters of the processor configuration are summarized in Table 2. For L2 cache, we consider blocks comprised of 64 bytes for data, 11 bits for its ECC, 25 bits for the tag, 3 control bits for valid, disable and dirty states and 7 bits for the tag ECC. The experiments are conducted using SPEC CPU2000 benchmarks. The applications characteristics are summarized in Table 3 that also shows the estimated overhead per failure for each benchmark and structure. An in-house SimPoint [17]-like tool is used to select the regions to simulate and run them for 100M committed instructions.

Two types of experimental results are reported: *simulation based* (Section 4.2.1) and *model-based* (Section 4.2.2).

The simulation based results are used to validate the accuracy of our model. The validation compares the values obtained by simulations against the values predicted by the model presented in Sections 3.3.1 and 3.4.2. The validation runs used 1000 fault maps. A *fault-map* represents the location of the faulty entries. Each fault-map represents a processor with faults and contains the locations of faults in the prediction arrays and the L2 cache. The validation fault-maps are randomly generated with cell probability failure of 1e-3 (lower probabilities have also been used to validate the model). For prediction arrays, unlike architectural arrays, it is not sufficient to know that a cell is faulty but we also need to know what is the faulty value. Therefore, for the prediction arrays, each fault-map is paired with a value-map that contains, for each fault location, a randomly generated fault value. For each fault map, the PVF of the predictions arrays and the L2 cache are estimated for each of the benchmarks using the model. If the PVF is 1% or more for both the prediction arrays and the L2 cache, a detailed performance simulation is performed. Out of the 26000 possible runs, only 1847 are predicted to produce PVF

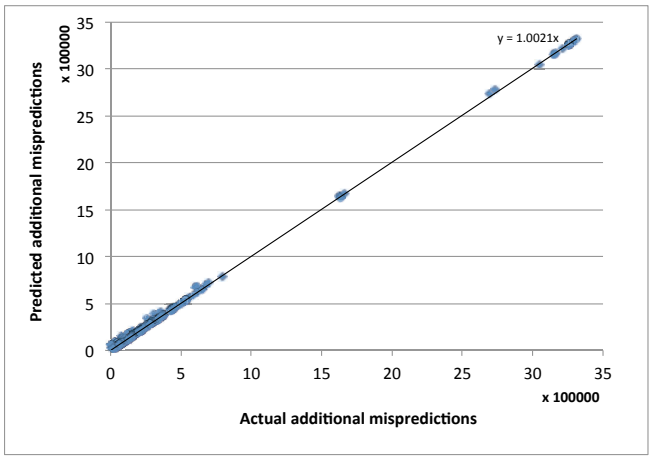

Figure 5: Actual vs. Predicted number of additional mispredictions with pfail=0.001.

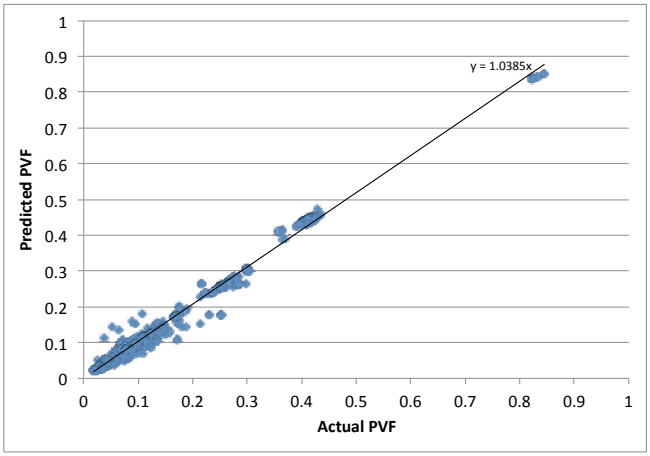

Figure 7: Actual vs. Predicted PVF with pfail=0.001.

more than 1% for both type of arrays.

For the model based results, we determine the expected PVF and the distribution bounds. The pfails values in [24] for technologies ranging from 32nm to 12nm (7.3e-09 to 2.6e-04) are used in this study. To account for aging phenomena and lower voltage, we use cell pfail of 1.0e-03, which also falls in the range considered by many recent studies [11, 1, 4, 36, 26, 2]. Finally, we explore for the L2 cache the impact of more robust but larger cells on the expected PVF and yield.

4.2. Experimental results

4.2.1. Simulation based results

The simulation based results are used to validate the model. Each point in Figures 5 and 6 shows the additional mispredictions (respectively additional misses) predicted from the model and the actual obtained through simulation for different benchmarks and fault-maps when cell pfail is 0.001. On each figure, the linear regression of all points and its corresponding equation is shown. We can observe from Figure 5 that the prediction model for additional mispredictions is very accurate (3.79% error on average) with virtually all predicted and actual values being equal. Figure 6 shows the same trends for the additional misses (3.34% error in average) except for two cases which are underestimated. We further analyzed these cases and found that for both cases the underestimation comes from the interaction

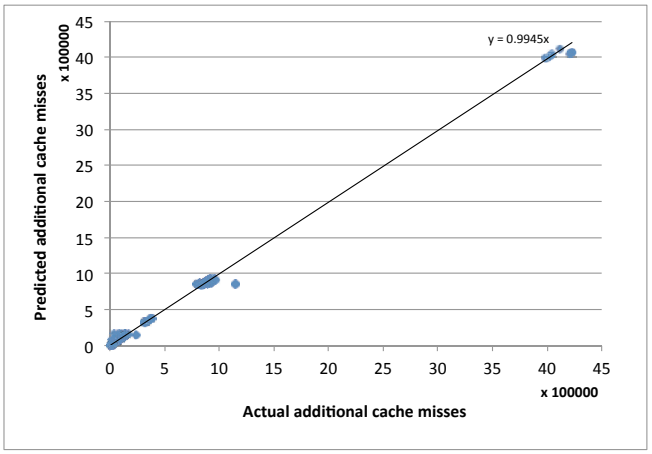

Figure 6: Actual vs. Predicted number of additional L2 misses with pfail=0.001.

with the faulty predictors which increase significantly the number of accesses to the L2 cache. We have validated this hypothesis by using only the cache fault-map and found that the predicted and actual values match in that case.

Figure 7 compares the PVF predicted from the model and the actual obtained through simulations. We observe again that the proposed model is quite accurate (7.15% error in average) with most of the predictions being close to the actual experimental outcome. However, there is more deviation in Figure 7 as compared to Figures 5 and 6. Considering that the additional misprediction and misses are predicted very accurately, the deviation is attributed to the variability in the penalties which are not captured by the proposed approach.

Overall, the results suggest that the model is quite accurate and also validate the model assumption that the ETV of different prediction arrays and the L2 cache are additive.

4.2.2. Model based results

This set of experiments estimates the expected PVF for cell pfails that correspond to different technology nodes [24]. Results are shown in Table 4. In this table, the first two columns show the technology node and the corresponding pfail, the next six columns show the maximal expected PVF in at least one benchmark for gshare, ras, ijump, all the predictors together, L2 cache and the global expected PVF. The last column shows the expected PVF for a composite benchmark which treats the consecutive benchmark execution as a single benchmark.

As shown in Table 4, the expected performance degradation from permanent faults in predictors is small and at most 2% when pfail equals to 1e-03. For the L2 cache, the performance degradation observed at current technology nodes is small, but, at pfail=2.6e-04 and pfail=1e-03 the maximal expected PVF is close to 30% and 84% respectively. Further analysis, not shown, reveals one benchmark, art00, experiences large PVF. The composite expected PVF for the different benchmarks at the two highest pfails is low but still significant at 2% and 14% respectively. Our model, therefore, can be useful to estimate at which point the performance degradation due to permanent faults can be tolerated and when it starts to become problematic and needs to be addressed with fault tolerance techniques.

The lower PVF of the predictors as compared to the L2 cache is mainly due to the size and the organization of each structure: number of total bits and number of bits per entry. Specifically, for the same cell pfail the expected number of faulty bits in the L2 cache will be $L2\_size\_in\_bits/Predictor\_size\_in\_bits$ times more. For example,

Technology	pfail [24]	Max Expected PVF						Composite Bench.
		Gshare	RAS	ijump	Predictors	L2 cache	**global**	Expected PVF
32nm	7.3e-09	0.00001%	0.00001%	0.00001%	0.00002%	0.00007%	0.00007%	0.00003%
22nm	1.5e-06	0.00106%	0.00213%	0.00111%	0.00347%	0.01441%	0.01533%	0.00621%
16nm	5.5e-05	0.03874%	0.07792%	0.04055%	0.12697%	1.70053%	1.73797%	0.25661%
12nm	2.6e-04	0.18286%	0.36615%	0.19083%	0.59581%	29.96965%	30.05939%	1.95337%
-	1.0e-03	0.69939%	1.37861%	0.72203%	2.23256%	83.71776%	83.73641%	14.60214%

Table 4: Expected Performance Vulnerability Factors (PVFs) for different technology nodes

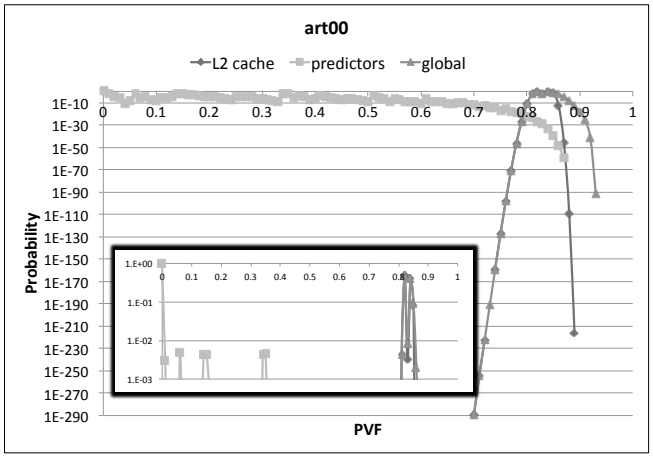

Figure 8: art00: average of the lower and upper distribution bounds. pfail=0.001 ; $\alpha = 2$.

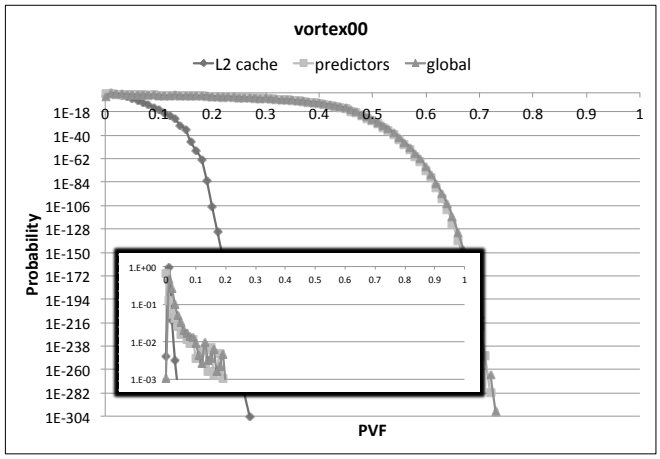

Figure 9: vortex00: average of the lower and upper distribution bounds. pfail=0.001; $\alpha = 2$.

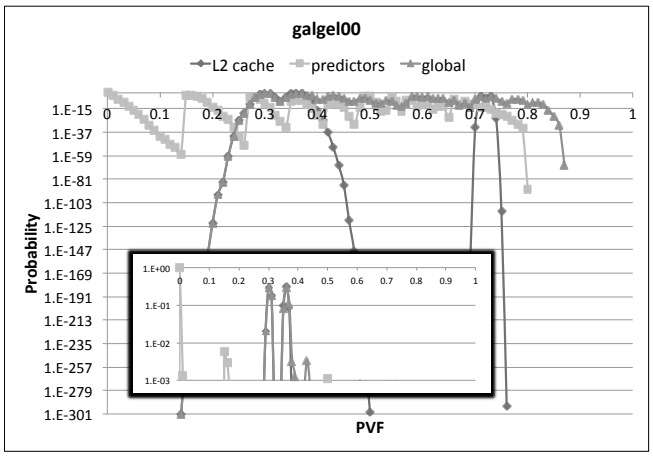

Figure 10: galgel00: average of the lower and upper distribution bounds. pfail=0.001; $\alpha = 2$.

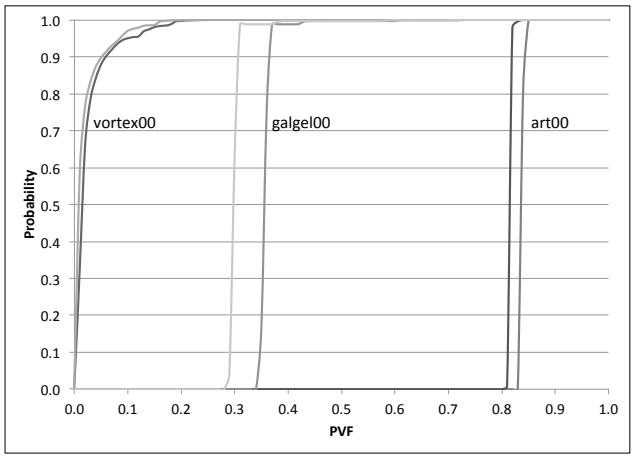

Figure 11: Cumulative distribution of the lower and upper distribution bounds for all units together. pfail=0.001 ; $\alpha = 2$.

for pfail=1.0e-3 the L2 cache used in this study will experience about 18000 faulty bits whereas the gshare predictor 65. Furthermore, the impact of a faulty bit in the L2 is significantly more pronounced since the size of a block is typically much larger than a predictor entry. We observe that the performance degradation due to a structure depends on the penalty and on the number of accesses to that structure. The more accessed an entry is and the higher its penalty, the higher the performance degradation when the entry is faulty.

PVF distribution bounds: To draw more insight about the expected PVF and to show that the range of possible PVF values can be quite distant from the expectation, we determine the lower and upper bounds of the PVF distributions. For reading ease, we show only three benchmarks representative of the different cases we observed.

Figures 8, 9, and 10 present the average of the two distribution bounds for the last level cache, the predictors and all units together. Figure 11 shows the cumulative distribution of the two bounds for the three benchmarks when all units are considered together to highlight the small distance between the bounds (i.e. small error).

The first observation is that the combined distribution of all units is dominated by the distribution of the unit that has the highest expected PVF (the cache for art00 and galgel00, the predictors for vortex00).

Moreover, the distribution results help to reveal the shortcoming of analysis based only on expected values and limited number of random fault maps: they cannot reveal the shape of the distribution. For instance, even if the highest PVF probability is close to the expectation, there is a significant probability to suffer a much higher PVF. In terms of population of processors, this will mean that a significant number will experience a PVF higher than the expectation. For example, with vortex00, 1 per 1000 processors will experience a PVF near 0.2 while the expected PVF is only 0.02.

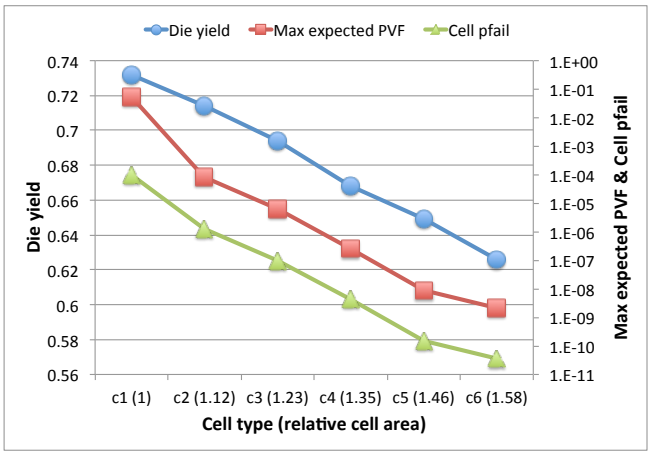

Figure 12: Die yield vs. Max Expected PVF for different L2 cell size (32nm).

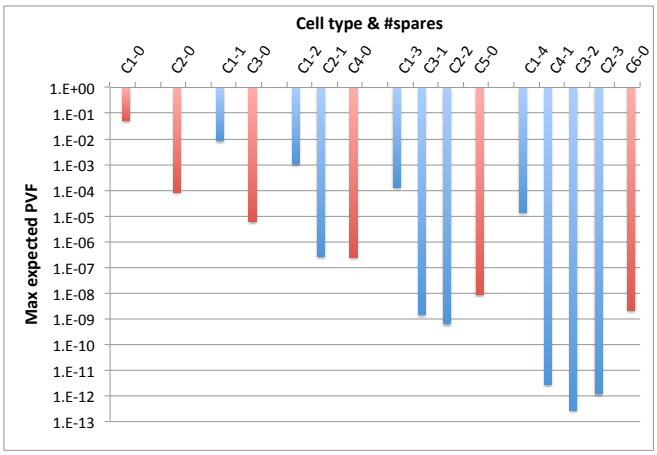

Figure 13: Max Expected PVF for different L2 cell size (x) and spare blocks (s). The configurations Cx-s are sorted in decreasing order according to their corresponding estimated die area.

The main cause for this difficult to detect behavior, when using fault maps, is that accesses are not evenly distributed across entries and, in general, it will require many fault-maps to produce a representative distribution.

Finally, the cumulative distributions in Figure 11 show that the bounds are accurate for all benchmarks with a small error, 0.027 on average and up to 0.07, when α is set to 2. In terms of computation time, computing the two bounds for all units together takes 1 minute per benchmark on average, when running on a typical desktop.

4.2.3. Case study: design trade-offs using PVF

As shown above, for the same cell pfail the L2 cache has higher PVF than the predictors. An approach that can reduce the L2 PVF is to use more robust cells. On the other hand, robust cells require more die area. This is especially true for large size L2 cache which occupies a significant part of the total processor's area and thus will lower the die yield. To assess this trade-off, we use six different cells sizes with their corresponding pfail from [37] for 32nm technology. Figure 12 shows the die yield[7], the maximal expected PVF for the different L2 cell size and their corresponding pfail. The results show that by choosing a less robust cell (c2 in this case) instead of the most robust one (c6), the PVF remain low (1.0e-4 vs. 1.0e-9) with a significant improvement in die yield (0.72 vs. 0.63). Furthermore, the cell with higher yield (c1) results in the largest PVF (close to 0.05). This first order analysis helps identify that c2 can be a good compromise.

Another approach to reduce the PVF is to use spare blocks (for instance s per set). The notion of spares can be easily incorporated in our model by changing Equation 10 as follows:

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

which provides, for every value of i $[1...n]$, the probability of having $i+s$ faulty blocks. To assess the benefits of sparing, Figure 13 shows the expected maximal PVF when using different cell size and number of spare blocks. Each configuration is noted Cx-s where x is the cell type and s is the number of spare blocks per set. In the figure, the different configurations are sorted in decreasing order according to their estimated area/yield. As shown in the figure, using spares with the less robust cell (c1) is not a good solution because it decreases significantly the die yield to achieve the same PVF as c2 without spares (PVF(C1-3)$\sim$PVF(C2-0)). Nevertheless depending on the targeted PVF threshold, it can be better to use relatively robust cells with some spares to maximize the die yield instead of using only the most robust cells. C2-2, for instance, gives a lower PVF and a better die yield as compared to C5-0 and C6-0.

The two cases studies highlight how the rapid exploration of PVF and area trade-offs can help designers configure and optimize their array designs.

5. Related Work

Previous work investigating the implications of permanently faulty cells in processor arrays focused on caches and considered yield and performance analysis.

Yield analysis provides the probability distribution in terms of number of faulty blocks expected in a cache given a specific cache configuration and a random probability for permanently faulty cells. Such analysis is usually based on binomial probability and helps determine the expected fraction of fault-free caches and number of spares that may be required by the caches with faults [2, 26, 36, 4, 1, 11]. The cell pfail depends on several parameters including technology node, failure sources, such as static process-variations and below Vcc-min operation, operating conditions and fault-model.

Performance analysis is useful to assess the performance of a processor that operates with disabled faulty blocks that have not been replaced with spares. Sohi [32] studied the impact on miss-rate of a cache organization with randomly disabled portions, such as ways and sets. The aim of [32] is to improve yield without noticeable miss-rate increase. Related research performed by Pour and Hill [25] also studied the impact of manufacturing faults on cache miss-rate. The work by [25] quantified analytically the expected miss-rate implications of different fault scenarios using a single run through an address trace. This approach aims to eliminate the need for long simulations but uses large number of random-fault maps. They estimate the expected miss ratio for a fixed number of faults by generating all the possible distributions of these faults over the cache sets. In [27], a methodology is proposed to estimate the expected miss rate and its expected distribution by using pfail without the need to generate fault

[7] $die\ yield = wafer\ yield * (1 + \frac{die\ area * defects\ per\ unit\ area}{\alpha})^{-\alpha}$, where $wafer\ yield = 1$, $defects\ per\ unit\ area = 0.4$, $\alpha = 4$ and the die area for the processor is estimated by using hotfloorplan [28] in our experiments.

maps. Our work shares similarities to [27] but, as highlighted in the introduction, also some key differences.

A number of recent studies consider the performance implications with disabled cache blocks assuming random fault-maps [21, 4, 1]. Our model does not rely on fault-maps but rather on probability analysis.

An earlier work is concerned with the testing and validation of mechanisms aiming to enhance performance [7]. This underlines the importance of mitigating faults in prediction arrays. However, this work does not evaluate the performance implications of faults. Bower *et al.* [8] investigate the performance effects of up to 8 permanently faulty entries in a branch history table. Their conclusion is that it is not worthwhile to protect this table against hard faults as performance degradation is negligible. Also, Makris *et al.* [3] evaluated the effect of a single fault in the most frequently accessed entry of a conditional branch predictor. Our paper considers the performance impact more rigorously using an analytical approach.

6. Conclusions and Future work

This work proposes a model for predicting PVF: the expected performance degradation in the presence of permanently faulty cells in architectural and non-architectural arrays. The model for a given program execution, micro-architectural configuration, and cell pfail, provides rapidly the PVF. PVF can be used by designers/researchers to evaluate and compare vulnerability of different structures and perform reliability driven trade-offs.

The model assumptions are validated and shown to be correct by comparing the predicted values of our model against actual values obtained by simulations with many fault-maps.

Predictions using the model reveal that the expected PVF for predictors is small even with high pfail. However, the PVF distribution reveals cases where processors in a given population will experience a significant performance degradation. Consequently, future processor reliability strategies may need to consider predictors. For the last-level cache, the PVF becomes increasingly prominent with technology scaling. This suggests that last level cache PVF mitigation techniques will become essential for future processors.

Design trade-off analysis using the model reveals that choosing appropriately the cell size in an array can help maintain PVF low with a small impact on die yield.

Future work, will extend and validate the proposed PVF model for other non-architectural arrays and architectural arrays as well as to multi-cores. We also plan to investigate low-cost detection and repair schemes for both architectural and non-architectural arrays to ensure a given PVF bound for all processors in a population. Developing an integrated AVF-PVF analysis will help measure the interactions between the two types of vulnerability when making design decisions.

Acknowledgements

The research leading to this paper is supported by the European Commission FP7 project "Energy-conscious 3D Server-on-Chip for Green Cloud Services (Project No:247779 "EuroCloud")". Damien Hardy was also supported by a mobility grant by HiPEAC (FP7 Network of Excellence). We like to thank the reviewers for their critique and comments that helped improve significantly the quality of this manuscript. The last author likes to acknowledge Veerle Desmet, Babak Falsafi, Emre Özer, Ronny Ronen, and André Seznec for their encouragement to pursue this line of work.

References

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, and A. González, "Low vccmin fault-tolerant cache with highly predictable performance," in *MICRO42*, 2009, pp. 111–121.

[2] A. Agarwal, B. C. Paul, H. Mahmoodi, A. Datta, and K. Roy, "A process-tolerant cache architecture for improved yield in nanoscale technologies," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 13, no. 1, pp. 27–38, Jan. 2005.

[3] S. Almukhaizim, T. Verdel, and Y. Makris, "Cost-effective graceful degradation in speculative processor subsystems: The branch prediction case," *Computer Design*, p. 194, 2003.

[4] A. Ansari, S. Gupta, S. Feng, and S. Mahlke, "Zerehcache: armoring cache architectures in high defect density technologies," in *MICRO42*, 2009, pp. 100–110.

[5] M. D. Beaudry, "Performance-related reliability measures for computing systems," *IEEE Trans. Comput.*, vol. 27, no. 6, pp. 540–547, Jun. 1978.

[6] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *DAC40*, Jun. 2003, pp. 338–342.

[7] P. Bose, "Testing for function and performance: Towards anintegrated processor validation methodology," *J. Electron. Test.*, vol. 16, no. 1-2, pp. 29–48, 2000.

[8] F. A. Bower, P. G. Shealy, S. Ozev, and D. J. Sorin, "Tolerating hard faults in microprocessor array structures." in *DSN34*, Jun. 2004, pp. 51–60.

[9] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De, and S. Borkar, "Circuit techniques for dynamic variation tolerance," in *DAC46*. New York, NY, USA: ACM, 2009, pp. 4–7.

[10] L. Cheng, P. Gupta, C. J. Spanos, K. Qian, and L. He, "Physically justifiable die-level modeling of spatial variation in view of systematic across wafer variability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 30, no. 3, pp. 388–401, 2011.

[11] Z. Chishti, A. R. Alameldeen, C. Wilkerson, W. Wu, and S.-L. Lu, "Improving cache lifetime reliability at ultra-low voltages," in *MICRO42*, 2009, pp. 89–99.

[12] R. Desikan, D. Burger, S. Keckler, and T. Austin, "Sim-alpha: a validated execution driven Alpha 21264 simulator," CS Dept., University of Texas at Austin, Tech. Rep. TR-01-23, 2001.

[13] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings of the 36th International Symposium on Microarchitecture*, Dec. 2003, pp. 7–18.

[14] S. Eyerman, K. Hoste, and L. Eeckhout, "Mechanistic-empirical processor performance modeling for constructing cpi stacks on real hardware," in *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, april 2011, pp. 216 –226.

[15] S. Eyerman, L. Eeckhout, and J. E. Smith, "Characterizing the branch misprediction penalty," in *Proceedings of the 2006 IEEE International Symposium on Performance Analysis of Systems and Software*, Mar. 2006.

[16] D. Fass, "Approximation of discrete multivariate probability distributions: Recursive and hierarchical approaches," 2005.

[17] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "Simpoint 3.0: Faster and more flexible program analysis," in *Journal of Instruction Level Parallelism*, 2005.

[18] E. Hao, P.-Y. Chang, and Y. N. Patt, "The effect of speculatively updating branch history on branch prediction accuracy, revisited," in *MICRO27*, Nov. 1994, pp. 228–232.

[19] E. Krimer, P. Chiang, and M. Erez, "Lane decoupling for improving the timing-error resiliency of wide-simd architectures," in *ISCA*, 2012, pp. 237–248.

[20] N. Ladas, Y. Sazeides, and V. Desmet, "Performance Implications of Faults in Prediction Arrays," in *2nd HiPEAC Workshop on Design for Reliability*, 2010.

[21] H. Lee, S. Cho, and B. R. Childers, "Performance of graceful degradation for cache faults," in *IEEE Computer Society Symposium on VLSI*, Mar. 2007, pp. 409–415.

[22] C. McNairy and J. Mayfield, "Montecito error protection and mitigation," in *HPCRI '05: 1st Workshop on High Performance Computing Reliability Issues, in conjunction with HPCA '05*, 2005.

[23] S. S. Mukherjee, C. Weaver, J. S. Emer, S. K. Reinhardt, and T. M. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," in *MICRO36*, Dec. 2003, pp. 29–42.
[24] S. R. Nassif, N. Mehta, and Y. Cao, "A resilience roadmap," in *DATE*, 2010, pp. 1011–1016.
[25] A. F. Pour and M. D. Hill, "Performance implications of tolerating cache faults," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 257–267, Mar. 1993.
[26] D. Roberts, N. S. Kim, and T. N. Mudge, "On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 32, no. 5-6, pp. 244–253, May 2008.
[27] D. Sánchez, Y. Sazeides, J. L. Aragón, and J. M. Garcia, "An analytical model for the calculation of the expected miss ratio in faulty caches," in *IOLTS*, 2011, pp. 252–257.
[28] K. Sankaranarayanan, S. Velusamy, M. Stan, C. L, and K. Skadron, "A case for thermal-aware floorplanning at the microarchitectural level," *Journal of ILP*, vol. 7, 2005.
[29] D. P. Siewiorek, R. S. Swarz, and A. K. Peters, *Reliable computer systems (3rd ed.): design and evaluation*. Ltd, 1998.
[30] L. J. Simonson and L. He, "Micro-architecture performance estimation by formula." in *SAMOS'05*, 2005, pp. 192–201.
[31] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark, "Improving prediction for procedure returns with return-address-stack repair mechanisms," in *MICRO31*, Nov. 1998, pp. 259–271.
[32] G. S. Sohi, "Cache memory organization to enhance the yield of high performance VLSI processors," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 484–492, Apr. 1989.
[33] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Lifetime reliability: Toward an architectural solution," *IEEE Micro*, vol. 25, no. 3, pp. 70–80, May 2005.
[34] Y. Taur, "CMOS design near to the Limit of Scaling," *IBM Journal of Research and Development*, vol. 46, no. 2/3, pp. 213–222, Mar./May 2002.
[35] A. Vassighi, O. Semenov, M. Sachdev, S. Member, A. Keshavarzi, and C. Hawkins, "Cmos ic technology scaling and its impact on burn-in," *IEEE Trans. on Devices and Materials Reliability*, vol. 4, 2004.
[36] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S.-L. Lu, "Trading off cache capacity for reliability to enable low voltage operation," in *ISCA35*, Jun. 2008, pp. 203–214.
[37] S.-T. Zhou, S. Katariya, H. Ghasemi, S. Draper, and N. S. Kim, "Minimizing total area of low-voltage sram arrays through joint optimization of cell size, redundancy, and ecc," in *Computer Design (ICCD), 2010 IEEE International Conference on*, oct. 2010, pp. 112 –117.

Appendix

Sketch of proof: correctness of the discrete distribution lower/upper bounds.

Let's consider the min clustering function (similar reasoning can be applied to the max clustering function).

1 Let's assume a discrete distribution P. By applying the clustering function on P, we obtain distribution P'.
By construction of the min clustering function, the following property (pa) is ensured:

$$pa : \forall X, P'(x < X) \geq P(x < X)$$

The correctness is ensured for the first step of the algorithm.

2 Let's assume two distributions P1 and P2 and their corresponding P'_1 and P'_2 distributions resulting from the min clustering function. The next step of the algorithm consist of combining P'_1 with P'_2, noted $P'_{1,2}$ and we want to be sure that the following condition (ca):

$$ca : \forall Z, P'_{1,2}(x < Z) \geq P_{1,2}(x < Z)$$

is always valid to ensure the correctness of this step.
With property *pa*, we have:

$$\forall Z, P'_1(x < Z) \geq P_1(x < Z)$$

$$\forall Z, P'_2(y < Z) \geq P_2(y < Z)$$

Thus,

$$\forall (x+y) < Z,$$
$$P'_1(x < Z-y) \geq P_1(x < Z-y)$$
$$\wedge$$
$$P'_2(y < Z-x) \geq P_2(y < Z-x)$$

Thus,

$$\forall (x+y) < Z,$$
$$P'_1(x < Z-y) * P'_2(y < Z-x) \geq P_1(x < Z-y) * P_2(y < Z-x)$$

And the combination of distributions is a multiplication of probabilities thus, ca is verified.

3 By induction we obtain a safe distribution bound.

■