Protecting Prediction Arrays against Faults

Yiannakis Sazeides, Constantinos Kourouyiannis, Nikolas Ladas
University of Cyprus

Veerle Desmet
Ghent University, Belgium

Abstract— **Continuous circuit and wire miniaturization increasingly exert more pressure on the computer designers to address the issue of reliable operation in the presence of faults. Virtually all previous work on processor reliability addresses problems due to faults in architectural structures, such as the register file or caches. However, faults can happen in non-architectural resources, such as predictors and replacement bits. Although non-architectural faults do not affect correctness they can degrade a processor performance significantly and, therefore, may render them as important to deal with as architectural faults.**

This paper quantifies the performance implications of faults in a line-predictor, and shows that performance can drop significantly when the line-predictor has faulty entries. In particular, a simulation based worst-case analysis of a high-end processor that experiences faults in 1% of the entries in the line-predictor, revealed an average performance degradation of 8% and up to 26%.

For solutions we point at no bit-interleaving as a more fault-tolerant design style for prediction arrays and to a hardware protection scheme based on address-remapping. This scheme is able to recover most of the performance loss when up to 5% of the line-predictor entries are faulty and when no faults exist it does not degrade performance.

I. Introduction

Current computer technology scaling trends are leading us toward smaller feature size and larger transistor budgets per chip. These developments present to the processor designer novel opportunities to improve performance and at the same time many challenges to overcome. One of these challenges is to provide reliable operation with little or no performance degradation in the presence of faults.

Sources of faults are mainly different types of variability [1] such as process variation due to manufacturing imperfections, temperature variation due to non-uniform activity and process variations, input variation due to workload composition and input data, environmental variation due to cosmic radiation and ambient temperature, and wear-out due to aging.

Techniques and processors that can provide reliable operation have been around for many years [2], [3]. What is distinct nowadays is that faults are becoming more frequent and challenging. For instance, due to the shrinking feature size, the design margins are narrowed down and the manufacturing process introduces both significant inter-die and die-to-die variations. Even in 130nm these variations are known to result in as much as 30% variation in maximum frequency and 20x variation in leakage power [4]. Furthermore, tight power envelopes and reduced voltage margins may necessitate operation below Vcc-min leading to otherwise well behaving components to become unreliable [5].

In the past, because faults were more rare, it was acceptable for low-end systems to offer little or no protection against them. As a result, mainly processors used in high availability systems employed advanced fault-tolerance techniques, such as using redundant and spare units [2], [3]. With future technology projections pointing to increased variability and faults, a more general use of fault-tolerance techniques is emerging. Furthermore, some of the known fault-tolerance techniques relevant to high-end systems may not be applicable to processors targeting markets where volume dictates profit and cost requirements are stringent.

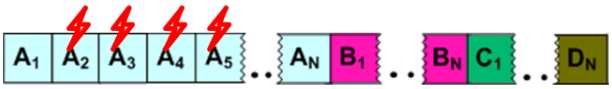
Virtually all previous microarchitectural studies on processor reliability and yield improvement aim to solve the problem for architectural resources such as a cache or an execution unit [6]–[8]. Faults in non-architectural resources, such as a predictor or a replacement array, received little attention because they do not affect correctness.

However, faults in these structures can affect performance and may need to be addressed to ensure acceptable performance levels, in particular for applications where performance is of paramount importance, e.g. real time systems that can not afford missing deadlines. Also, non-architectural faults can result in energy inefficiency for the extra work needed due to the additional mispredictions and/or cache misses they cause. Moreover, non-architectural resources constitute a significant fraction — according to our estimates around 10% for an EV6 like processor [9] — of the active area of the chip where temperature is higher and, therefore, susceptible to some wear-out and process variation induced faults [10], [11]. Also, if architectural resources are protected and the frequency of faults keeps increasing eventually non-architectural resources will experience more faults and potentially become a performance bottleneck.

One other reason to consider non-architectural faults is *frequency-binning* [4], [12] since non-architectural faults can cause non-uniform performance within a bin. One may argue that if non-architectural faults are not dealt with then an additional step in the binning process, with its associated extra cost, may be needed to classify chips according to their performance.

Previous work by Bower *et al.* [6] investigates the performance effects of up to 8 faulty entries in a branch history table. They conclude it is not worthwhile protecting it against hard faults as performance degradation is negligible. Our work considers the performance impact for a different prediction structure and shows that faults in a few bit-interleaved entries can lead to a significant performance penalty. In particular, this paper quantifies the performance implications of faults in a a line-predictor and shows that it may need protection against faults. This study also points out that a no bit-interleaving design style for prediction arrays is more resilient against

(a) Fault cluster in bit-interleaving design



(b) Fault cluster in no bit-interleaving design

Fig. 1. *Bit-Interleaving vs. No Bit-Interleaving with multi-bit faults.*

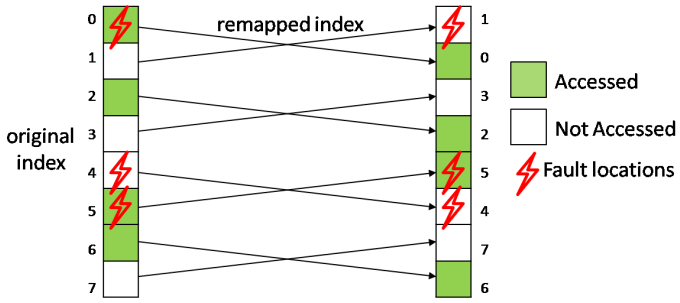


Fig. 2. *Principle of address-remapping.*

faults. Finally, we present and evaluate the effectiveness of a simple fault detection and repair scheme for prediction arrays.

The remainder of this paper is organized as follows. Section II provides details on fault modeling non-architectural array structures while Section III introduces a detection and repair scheme for predictors. Section IV overviews the used methodology, and Section V presents the experimental results. Section VI concludes the paper.

II. Fault Modeling

Non-architectural structures can be divided into arrays, such as a predictor array, and random logic, such as an adder used for computing the address of prefetched data. The focus in this paper is on faults in *prediction arrays* which are more area dominant within non-architectural units.

The impact of faults can vary widely according to their number, location and fault model. The more faults the more potential for degradation. Also, a fault on a frequently used entry is likely to have a bigger impact than a fault on a rarely accessed entry. Similarly, a cell stuck-at-1 may have more impact if the bits stored in the cell are more biased toward zero. Therefore, to investigate fault implications requires examining the following parameters: (i) the number of faults, (ii) the location of the faults in the array, and (iii) the fault model of each fault.

The physical principles that will determine the value of the above parameters in the future are poorly understood and difficult to model accurately. Regarding the number of faults it is important to consider scenarios covering a range of faults from small to large. Although currently, manufacturers do not expect to ship chips with 1,000's of faults, in the future power constraints may require operating circuits with voltage below Vcc-min at the expense of a larger number of unreliable devices [5]. Also, it is generally known that faults, at least due to manufacturing and process variation, are distributed according to a *random* component—expressing the non-determinism of fault locations—and a *spatial* component—expressing the clustering of faults. Furthermore, it is understood that various fault behaviors can be observed in arrays and several fault models have been proposed in the literature to capture them, such as random, inverse, stuck-at, etc.

Another issue that affects the resiliency of arrays is bit-interleaving. Specifically, for caches the bits from different words are usually interleaved together to increase both area efficiency and tolerance against soft-errors [13]. The former is possible since a single sense amplifier can be used for multiple columns and the latter is realized because multiple neighboring bits can flip while still being able to detect and correct them since neighboring bits are protected by distinct error correction codes.

The bit-interleaving design style is applicable to non-architectural arrays but only offers an area advantage since predictors are, as far as we know, not error protected. However, with respect to performance in the presence of faults, bit-interleaving may not be suitable since a multi-bit fault cluster can affect multiple predictions and, therefore, can degrade performance to a larger extent as compared to no bit-interleaving. This is illustrated in Fig. 1 where a wordline contains four predictions, *A*, *B*, *C* and *D* each *n* bits wide. A four bit multi-bit cluster fault affects four predictions in the bit-interleaved design but merely a single prediction in the no bit-interleaving design.

III. Address Remapping for Prediction Arrays

This section introduces a mechanism to mitigate the performance degradation due to faults in prediction arrays. Our scheme is based on *address-remapping* and is applicable to any prediction array though this study only considers its application to a line-predictor [14].

The principle of address-remapping is illustrated in Fig. 2 for an 8 entry prediction array. Assume that the array has 3 faulty entries and that only the 4 filled entries are accessed during the execution of a program. In the original situation, in the left of Fig. 2, two out of the three faulty entries are accessed. The problem this paper attempts to solve is to detect during execution whether there are significant accesses to faulty entries and consequently remap the accessed entries to different locations to reduce the number of accesses to faulty entries. The right of Fig. 2 shows the situation if the accesses are permuted such that the same number of entries are accessed after remapping but now only 1 of those is faulty.

Note that address-remapping does not merge accesses from different entries, it merely redistributes them. This highlights a key difference between protecting architectural and non-architectural arrays. For the latter a protection mechanism does not need to detect and avoid/repair all accessed faulty entries as long as most of the performance degradation is recovered. This property enables the design of simple detection and repair schemes.

Fig. 3 shows how prediction arrays can be protected using *index remapping* and *remapping search* units.

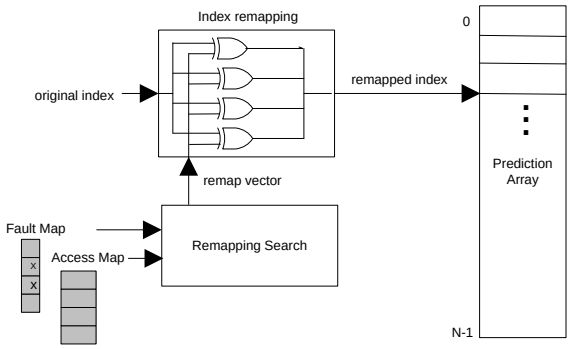


Fig. 3. *Prediction Array Protection Scheme.*

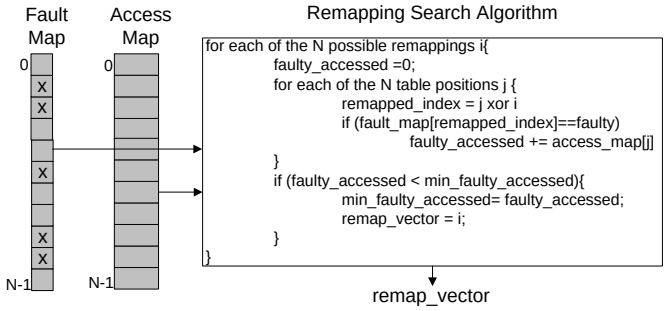


Fig. 4. *Remapping Search.*

A. Index Remapping

The purpose of the index-remapping is to provide the means to dynamically change the logical to physical mapping of each location. The idea of remapping logical to physical locations to avoid faults has been proposed before, but, as far as we know, in all previous work the remapping function mapped a previously faulty entry to either a non-faulty entry or to a spare [6]–[8], [15]. The remapping function we employ simply redistributes the accesses, i.e. there are no spares to remap to, and one or more entries may get remapped to faulty locations.

An analysis of the access distributions of prediction structures provides the rationale behind this remapping approach. Experimentally we found that for SPEC CPU 2000 benchmarks very few entries are responsible for the majority of correct predictions for the line-predictor. Therefore, when accesses to faulty entries are detected it may be possible, through the remapping function, to remap frequently accessed faulty entries to rarely accessed entries without a fault. In this case, the performance will be as if there were no faults and no further remapping is needed. Of course, it is possible for remapping not to improve or even make the problem worse, i.e. increase the number of faulty accessed entries. In the latter case, the mechanism will detect that many accesses are mapped to faulty entries and will try to determine a better remapping.

There are plenty of remapping functions that can be considered and in this study we choose to use the *XOR*. The *XOR* is a simple remapping function, and can be implemented as shown in Fig. 3 by XOR-ing the original index with a *remap-vector*. *XOR-remapping* only provides symmetric permutations of the logical to physical location mapping. Note that the remapping shown in Fig. 2 can be obtained by XOR-ing the index with the value 1. Simplicity of the remapping function is an important asset since the remapping function lies in the critical path to access the array.

The success of *XOR-remapping* depends on choosing well the *remap-vector* to redistribute the accesses to avoid faulty entries. For an n bit remap-vector there exist 2^n possible permutations and it is the job of the *remapping search* unit to find out which permutation will work best.

B. Remapping Search

The proposed address-remapping scheme monitors the accesses in faulty entries per interval and if their number is above a certain threshold it triggers a search to determine the remapping vector that will minimize the accesses to faulty entries. In particular, this search uses a *fault-map* that indicates for each predictor entry whether is faulty or not and an *access-map* that contains the number of accesses in each entry in the previous interval. The search determines for each possible remapping the number of accesses to faulty entries in the previous interval and selects the remapping with the minimum number of faulty accesses as the new remap vector for the next interval. The functionality of the remapping-search unit is illustrated in Fig. 4. It is noteworthy that the proposed scheme does not degrade performance when no faults are present or when all faults occur in non-accessed entries.

In the above discussion we assumed that the fault map is available, next we describe how it is determined.

The fault map is a bit vector indicating for each entry whether is faulty or not. The fault map can be determined using a built-in-self-test (BIST) unit that applies to each predictor entry various test patterns to detect possible faulty behavior [16]. The BIST can be activated selectively, instead of doing so continuously, during periods when the predictor is idle, at boot-up, or at regular time intervals.

The state cost of the BIST, is roughly equal to the counters used to sequence through the predictor table and the test-patterns table, and the ROM that stores the patterns [16]. The number of test patterns depends on the type of faults that need to be detected but for a memory this is typically small. Therefore, the state cost of the BIST is a very small as compared to the predictor cost.

IV. Methodology

We extended the validated cycle accurate simulator *sim-alpha* [17] to measure the performance implications of faults in the line-predictor of a high performance out-of-order superscalar processor that its key parameters are summarized in Table I. We simulated all the SPEC CPU 2000 benchmarks using reference inputs for a 100M committed instructions. An in-house SimPoint-like tool is used to select the regions to simulate.

Pipeline depth	15 stages
Superscalarity	4
Line-Predictor	6 KB, 4096 entries, 11 bit prediction + 1 bit hysteresis/entry
Branch Predictor	4 KB meta, 4 KB bimodal, 8 KB gshare
Reorder buffer	128
L1 instr. cache	64 KB, 2-way, 64 B blocks, 1-cycle
L1 data cache	64 KB, 2-way, 64 B blocks, 3-cycle
L2 unified cache	2 MB, 8-way, 64 B blocks, 12-cycle hit latency, 255 cycles miss latency

TABLE I

BASELINE PROCESSOR

The 4096 line-predictor entries are assumed to be mapped in a row-major fashion using 256 wordlines each holding 16 11-bit predictions. This configuration makes the line-predictor more square and area efficient for layout. For the bit-interleaving design style, which is the default configuration, the predictions on a wordline are bit-interleaved and an entry consists of 16 consecutive bits coming from 16 different logical predictions.

The study focuses on cell faults modeled as stuck-at faults and investigates the effect on performance with increasing number of faults. To capture the spatial component of fault distribution we assume that whole line-predictor entries are faulty. We perform experiments for two scenarios: worst-case and random-case. Assuming n faults, the worst-case scenario injects n faults in the top n entries that gave the most correct predictions. For random experiments, with a given number of faults, 10 random fault maps are evaluated. For the worst-case each faulty entry is assigned a stuck-at value at each bit position that will maximize the mispredictions. For the random study when an entry is assumed to be faulty each of its cells is assigned randomly a stuck-at-1 or stuck-at-0 fault.

Next we describe the specific values used in the experimentation for key parameters of the protection mechanism presented in Section III.

To keep the area overhead of the proposed protection scheme low, the fault-map and access-map track faults and accesses at the granularity of wordlines and therefore each table consists of 256 entries. Each entry in the fault-map contains one bit indicating whether the corresponding line-predictor wordline contains a fault. In this paper we assume that the fault-map is always up to date since faults occur infrequently. The access-map entries contain 8-bit saturating counters that count the accesses to the corresponding line-predictor wordline per interval. The interval length between checks as to whether remapping is needed is 100000 committed instructions. Such interval length allows the access-map table counts to be small while keeping to minimal the cold effects due to remapping. The number of accesses, to faulty line-predictor entries, that will trigger a search for new remapping is set to 1000 per interval.

The remapping search only explores wordline remapping. This reduces the remapping search space to 256 options but still requires some time to compute. We account for the delay to perform the search by assuming that it requires 65536 cycles to be completed.

The above cost optimizations and simplifications were found to have minimal impact on the quality of the proposed solution. The overall state cost of the proposed scheme is mostly dominated by the access-map counts and is 325 bytes. This corresponds to 5% of the line-predictor state cost.

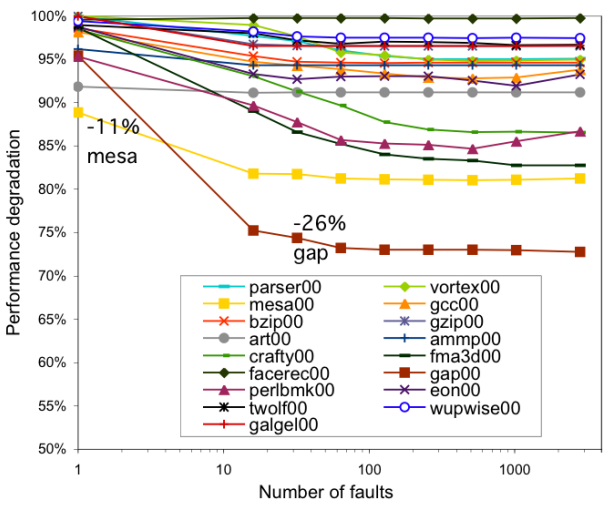


Fig. 5. *Performance vs Number of Faults.*

V. RESULTS

In this section we present our results starting with the performance degradation in the line-predictor with increasing number of faults. Next, we discuss how two different array design styles influence fault tolerance in prediction arrays, and finally we examine the effectiveness of our address-remapping to protect a line-predictor against faults.

Performance Implications. To assess the performance implications of faults in a line-predictor, we measure its performance with increasing number of faults in a worst-case setup. Fig. 5 illustrates the performance degradation compared to a baseline which contains no faults with the different curves corresponding to different benchmarks. Note the logarithmic scale for the number of faults. A single fault can degrade up to 11% for `mesa`, and 2.4% on average. With a few faults in the line-predictor, processor performance drops significantly, with merely 1% faulty entries (i.e. 32 entries) an average degradation of 8% is measured, and up to 26% for `gap`. It can also be seen that the curves saturate quickly, indicating that few entries are responsible for the majority of correct predictions.

Bit-interleaving. Fig. 6 compares the worst-case performance degradation of a bit-interleaved versus a no bit-interleaved line-predictor with varying number of faults. For each number of faults, 10 runs with a random fault map are evaluated for each benchmark and the results are sorted in descending order of performance for the scheme with no bit-interleaving. The latter curve is consistently above the bit-interleaved curve, clearly supporting that a bit-interleaved design style will suffer significantly more performance degradation for the same number of faults. This suggests that a non-interleaved design style for predictors is more resilient against faults.

Address-remapping. Fig. 7 shows the effectiveness of our address remapping scheme to recover the performance loss in the presence of faults. It compares the performance for a line-predictor without and with address-remapping. Again, for each

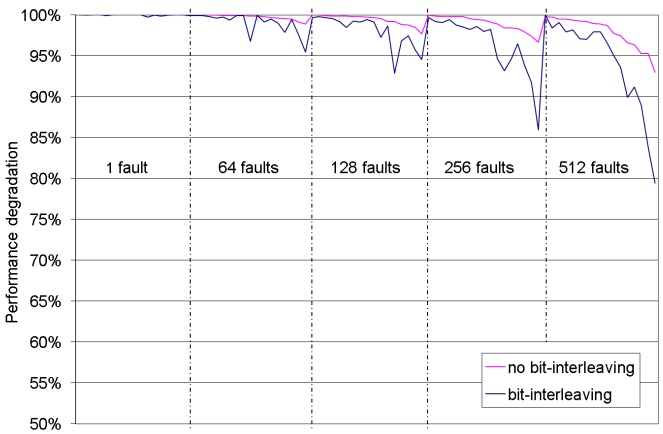


Fig. 6. *Bit-interleaving vs. no bit-interleaving.*

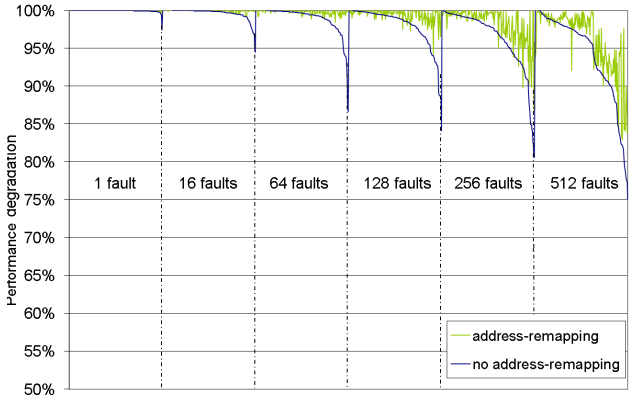


Fig. 7. *Remapping vs. No Remapping.*

number of faults, we measured performance for 10 random fault maps for each benchmark and this time the results are sorted in descending order of performance for the scheme with faults and no address-remapping.

For the majority of the runs the proposed address-remapping recovers the performance loss, almost perfectly up to 128 faults or 5% faulty entries. When more faults are present, address-remapping can typically recover a fraction of the performance loss.

VI. Conclusion

This work argues that it is important to protect prediction arrays against faults because they can degrade performance. The paper quantifies the performance implications of faults in the line-predictor and shows that performance can drop significantly when the line-predictor has few faulty entries. More specifically, a simulation based worst-case analysis of a high-end processor that experiences faults in 1% of the entries in the line-predictor, revealed an average performance degradation of 8% and up to 26%. For solutions we suggest no bit-interleaving as a more fault-tolerant design style for prediction arrays and a hardware protection scheme based on address-remapping. This scheme is able to recover most of the performance loss when up to 5% of the line-predictor entries are faulty, and when no faults exist it does not degrade performance.

Acknowledgments

This work is partially supported by the University of Cyprus, Ghent University, HiPEAC, and Intel. The authors would like to recognize Constantinos Vrionis for the preliminary studies leading to this work.

References

[1] O. S. Unsal, J. W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin, "Impact of parameter variations on circuits and microarchitecture," *IEEE Micro*, vol. 26, no. 6, pp. 30–39, 2006.

[2] F. J. Aichelman, "Fault-tolerant design techniques for semiconductor memory applications," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 177–183, Mar. 1984.

[3] P. J. Meaney, S. B. Swaney, P. N. Sanda, and L. Spainhower, "IBM z990 soft error detection and recovery," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 419–427, Sept. 2005.

[4] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter variations and impact on circuits and microarchitecture," in *Proceedings of the 40th Design Automation Conference*, June 2003, pp. 338–342.

[5] 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 *Proceedings of the 35th Annual International Symposium on Computer Architecture*, June 2008, pp. 203–214.

[6] F. A. Bower, P. G. Shealy, S. Ozev, and D. J. Sorin, "Tolerating hard faults in microprocessor array structures." in *Proceedings of the 34th Annual International Conference on Dependable Systems and Networks*, June 2004, pp. 51–60.

[7] A. Das, S. Ozdemir, G. Memik, J. Zambreno, and A. Choudhary, "Microarchitectures for managing chip revenues under process variations," *Computer Architecture Letters*, vol. 6, June 2007.

[8] M. Mutyam and V. Narayanan, "Working with process variation aware caches," in *Proceedings of the 2007 Design, Automation and Test in Europe Conference*, Apr. 2007, pp. 1152–1157.

[9] R. Kessler, E. McLellan, and D. Webb, "The Alpha 21264 microprocessor architecture," in *Proceedings of International Conference on Computer Design*, Oct. 1998, pp. 90–105.

[10] S. Ozdemir, D. Sinha, G. Memik, J. Adams, and H. Zhou, "Yield-aware cache architectures," in *Proceedings of the 39th Annual International Symposium on Microarchitecture*, Dec. 2006, pp. 15–25.

[11] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "Exploiting structural duplication for lifetime reliability enhancement," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005, pp. 520–531.

[12] A. Datta, S. Bhunia, J. H. Choi, S. Mukhopadhyay, and K. Roy, "Speed binning aware design methodology to improve profit under parameter variations," in *Asia South Pacific Design Automation Conference*, Jan. 2006, pp. 712–717.

[13] K. Ghose and M. B. Kamble, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation," in *ISLPED*, Aug. 1999, pp. 70–75.

[14] B. Calder and D. Grunwald, "Next cache line and set prediction," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, May 1995, pp. 287–296. [Online]. Available: citeseer.nj.nec.com/calder95next.html

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

[16] B. T. Murray and J. P. Hayes, "Testing ICs: Getting to the core of the problem," *Computer*, vol. 29, no. 11, pp. 32–38, Nov. 1996.

[17] R. Desikan, D. Burger, S. Keckler, and T. Austin, "Sim-alpha: a validated execution driven Alpha 21264 simulator," Department of Computer Sciences, University of Texas at Austin, Tech. Rep., 2001.