Performance Implications of Hard-Faults in Non-Architectural Structures

Veerle Desmet* Yiannakis Sazeides+ Costas Vrioni+

*Dept. of Electronics and Information Systems
Ghent University, Belgium
veerle.desmet@elis.UGent.be

+Dept. of Computer Science
University of Cyprus, Nicosia
{yanos,cs03v1}@cs.ucy.ac.cy

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 hard-faults. Virtually all previous work on hard-fault reliability addresses problems that arise when a fault occurs in architectural resources, such as the register file or caches. However, hard-faults can happen in non-architectural resources, such as predictors and replacement bits. Although non-architectural hard-faults do not affect correctness they can degrade a processor performance significantly and, therefore, may render them as important to deal with as architectural hard-faults.

In this paper we determine, using previously proposed hard-fault models, the temperature conditions for which the frequency of hard-faults in non-architectural structures is in the same order of magnitude as in architectural structures. Furthermore, this paper quantifies the performance implications of hard-faults in two non-architectural resources: a line predictor and a return-address-stack. In particular, a simulation based analysis of a high-end processor that experiences a stuck-at fault in one of its most frequently used cells in the return-address-stack and the line predictor, revealed a degradation up to 9% and 3%, respectively. When a stuck-at hard-fault occurs in one of the output drivers the slowdown can be as high as 34% in the return-address-stack and 19% in the line predictor.

1 Introduction

1.1 Motivation

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 formidable challenges is to provide dependable operation with little or no performance degradation in the presence of faults.

Techniques and processors that can provide dependable operation have been around for many years [15, 18, 21]. What is distinct nowadays is that faults, due to shrinking feature size, occur more frequently. Srinivasan *et al.* [22] report that the failure rate due to hard-faults of a scaled 65nm processor is 316% higher than a similarly pipelined 180nm processor.

In the past, because faults were more rare, it was acceptable for low-end systems to offer little or no protection against faults. As a result, mainly processors used in high availability systems employed advanced fault-tolerance techniques, such as using redundant and spare units [15]. With technology projections pointing to a dramatic fault increase in processors [22] 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.

The above developments are creating an impetus for the development of processor fault-tolerance techniques that in the presence of faults can provide minimal or no performance degradation at low-cost. This paper represents a step toward that direction.

1.2 Fault Classification

Faults can be classified according to several criteria. One of the most widely used classification divides processor faults based on duration into *non-permanent faults* and *permanent faults*. Non-permanent faults temporarily affect the system, and may appear for a

very short time, e.g. alpha particles, or stay for an undetermined longer period, but eventually they can disappear. Permanent or *hard-faults* remain in existence for the lifetime of a system or until a repair fixes the fault. Hard-faults can occur during manufacturing due to design imperfections or during operation due to wear-out. The most dominant wear-out mechanisms are more likely at high temperature of operation [23].

Alternatively faults can be classified depending on their implications on correctness and performance. A fault may manifest into incorrect execution if it occurs in an *architectural resource*, such as a cache or an execution unit, or may degrade performance if it happens in a *non-architectural resource*, such as a predictor or a replacement array.

Virtually all previous studies on hard-faults due to wear-out aim to solve the problem for architectural resources [2, 3, 4, 5, 6, 8, 17]. Hard-faults in non-architectural resources 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.

1.3 Contributions

This paper performs an initial investigation of the performance implications of hard-faults in non-architectural structures to determine whether such structures merit protection from hard-faults due to wear-out. In particular, the contributions in this paper are:

- show that the frequency of hard-faults in non-architectural structures is in the same order of magnitude as in architectural structures.
- quantify the performance implications of hard faults in a line predictor and a return-address-stack. Our results show up to 19% slowdown for a hard-fault in a line predictor, and up to 34% performance degradation when a fault occurs in the return-address-stack.

The remaining paper is organized as follows. Section 2 demonstrates that the frequency of hard-faults in non-architectural structures is in the same order of magnitude as in architectural structures. Section 3 illustrates the different effects hard-faults may have on a physical array structure. Section 4 provides details on our methodology while Section 5 quantifies the performance implications of hard-faults in two non-architectural structures. After Section 6 with related work, Section 7 concludes the paper and provides direction for future work.

2 Architectural vs Non-Architectural Hard-Faults

Hard-faults due to wear-out are mainly influenced by operating conditions, such as temperature or activity. Since these conditions can be different between non-architectural and architectural structures, one of the key questions this study attempts to answer is whether non-architectural structures, like a branch predictor, are equally vulnerable to wear-out as architectural structures, such as a register file.

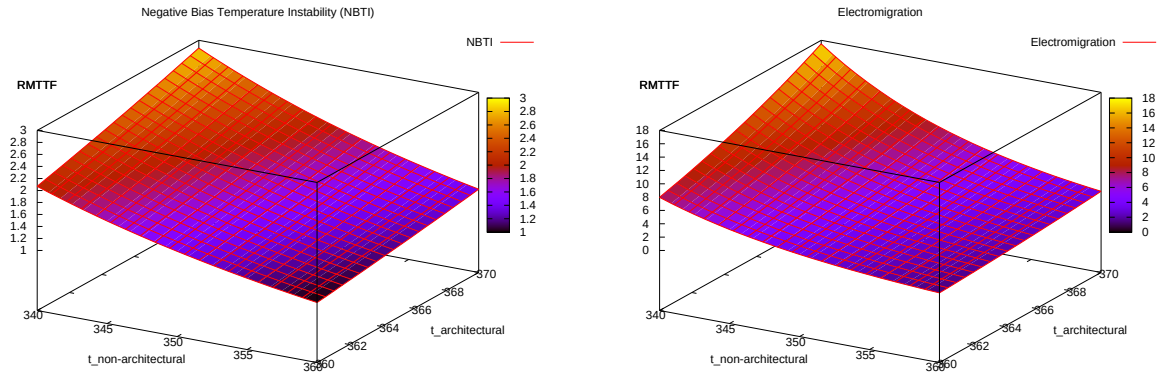
To answer whether we should worry about non-architectural hard-faults, we employ analytical models proposed by IBM researchers for some of the most important wear-out mechanisms that lead to permanent hard-faults: electromigration, gate oxide breakdown, stress migration, thermal cycling, and negative bias temperature instability (NBTI) [11, 24, 25]. These analytical models express a transistor's or wire's mean time to failure (MTTF) as a function of temperature. For some of the wear-out mechanisms their models consider additional parameters, such as activity, voltage, and current density.

Since the value of temperature, and of other parameters of interest, may vary across the units of a processor, the MTTF due to hard-faults is expected to vary across different structures. To determine quantitatively the difference between the MTTF of architectural and non-architectural structures we compute the *ratio of MTTF* (RMTTF) of a non-architectural over an architectural structure for a range of temperatures. Also, we assume that—as shown in previous temperatures studies [19] —the hottest architectural resource has usually the same or higher operating temperature than any non-architectural structure.

The RMTTF equation for NBTI is given below and is derived assuming that t_a and t_{na} correspond to the temperature of the architectural and non-architectural structures, respectively.

$$\begin{aligned}\text{RMTTF} &= \frac{MTTF_{non-arch}}{MTTF_{arch}} \\ &= \frac{\left(\ln(\frac{A}{1+2e^{\frac{B}{kt_{na}}}}) - \ln(\frac{A}{1+2e^{\frac{B}{kt_{na}}}} - C))\frac{t_{na}}{e^{\frac{-D}{kt_{na}}}}\right)^{\frac{1}{\beta}}}{\left(\ln(\frac{A}{1+2e^{\frac{B}{kt_a}}}) - \ln(\frac{A}{1+2e^{\frac{B}{kt_a}}} - C))\frac{t_a}{e^{\frac{-D}{kt_a}}}\right)^{\frac{1}{\beta}}}\end{aligned}$$

where A, B, C, D, and β are fitting parameters, and k is Boltzmann's constant. According to Zafar



(a) Negative Bias Temperature Instability

(b) Electromigration

Figure 1. Ratio of MTTF (RMTTF) for Architectural versus Non-Architectural Wear-Out

et al. [25] we used the values $A = 1.6328$, $B = 0.07377$, $C = 0.01$, $D = -0.06852$, and $\beta = 0.3$. In a similar way the RMTTF can be derived for all other wear-out models.

Figure 1 shows the RMTTF for negative bias temperature instability and electromigration when the temperature of the architectural resource ranges from 360 to 370 degrees K, and the temperature of the non-architectural resource ranges from 340 to 360 degrees K. The maximum temperature difference of 30 degrees (370-340) is larger than what is typically observed between processor units in thermal simulations [19]. For electromigration the RMTTF equation used in Fig. 1(b) assumes that the activity of the architectural resource is 1.6 times that of the non-architectural resource. This activity ratio is typical between an instruction cache and a conditional branch predictor.

The data reveal that for NBTI the RMTTF varies from 1 to 2.6 and for electromigration from 1 to 17. A ratio of one means hard-faults are equally likely to occur in an architectural and a non-architectural structure. A higher ratio indicates that a hard-fault is less frequently to occur in the non-architectural structure, e.g. an RMTTF of 2 means that there is 50% less chance of having a hard fault in a non-architectural structure. For the other failure mechanisms we found similar ratios as in Figure 1(a) for NBTI.

The above analysis shows that hard-faults in non-architectural structures are likely to occur in the same order of magnitude as in architectural structures. Therefore, if the performance degradation they cause is substantial, they may need to be protected against faults. In Section 5 we investigate the performance impact of hard-faults in two non-architectural structures.

3 Modeling Hard-Faults in Non-Architectural Array Structures

Non-architectural structures can be divided into arrays, such as a predictor array, or random logic, such as an adder used for computing predicted address. The focus of the remaining paper is on faults in array structures which are more dominant between non-architectural units.

The various wear-out mechanisms cause physical faults that may occur in different locations in an array structure such as cell, bitline, wordline, decoder and driver as shown in Figure 2.

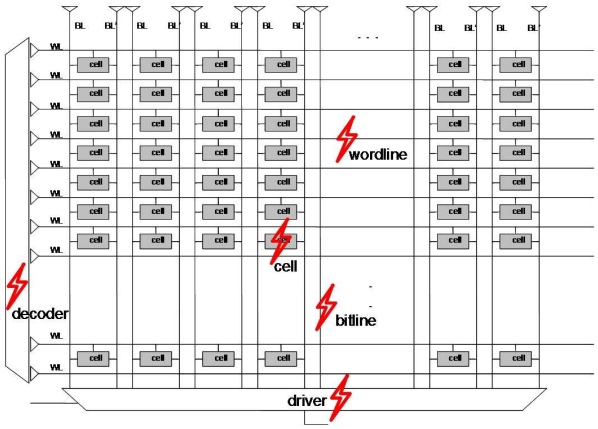


Figure 2. Physical hard fault locations.

Studying faults at the physical level requires detailed and slow simulations that are not practical. What is done usually instead, is to abstract the physical faults

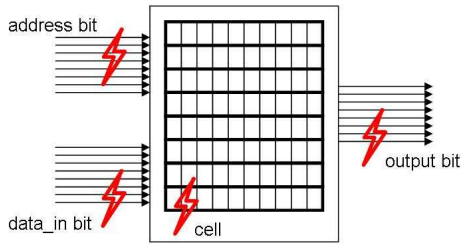


Figure 3. Possible stuck-at locations in logical array view.

into functional faults in a logical model of an array that is easier to model and study.

The most popular class of functional faults are stuck-at. A stuck-at-0 or stuck-at-1 fault causes a signal to permanently hold the value zero or one, respectively. More sophisticated functional fault models include transition faults and coupling faults. Transition faults behave as stuck-at faults, but it takes until a specific transition before effectively manifesting as a fault. Coupling faults represent relations between multiple erroneous signals, e.g. always the same or opposite value as another signal, and the range of possibilities is enormous. Therefore, we focus on the most typically used models of stuck-at-0 and stuck-at-1 faults.

Figure 3 shows a logical view of an array and some of its possible stuck-at fault locations: address bit, data_in bit, cell, and output bit. The impact of a single stuck-at fault can widely vary according to where exactly the fault manifests itself. A cell stuck-at is reasonable to cause less degradation than a stuck-at output bit.

We like to point out that it is necessary to consider both stuck-at types because their impact might be different. Indeed, a stuck-at fault is only manifested when it is read and is supposed to contain the opposite value. In the other case, when e.g. a stuck-at-0 is read and has to be a zero the hard fault is logically masked. Thus, dependent on the value that it was supposed to be a stuck-at-0 will be more severe than a stuck-at-1, or vice-versa.

In the experimentation we will consider stuck-at faults for a single cell and a single output bit for non-architectural array structures.

4 Methodology

We extended the validated cycle accurate simulator *sim-alpha* [9] to measure the performance of a high performance out-of-order superscalar processor with and without faults. Table 1 summarizes the parameters for our baseline processor.

The experiments we perform aim to measure the impact of a stuck-at fault on a single cell and a single output bit in two non-architectural structures, namely the line predictor and return-address-stack. In the near future we plan to extend our study to other non-architectural structures.

For each unique experimental configuration with a hard-fault we perform two experiments, one injecting a stuck-at-0 and another with a stuck-at-1, and we report the worst case of the two. Therefore, our results have the potential to represent the worst case performance degradation under our assumptions.

For the single cell stuck-at faults, for each program we select the most frequently used entry, according to a previous program run, again aiming to determine the implications for a worst case scenario. Note that due to subtle interactions of the different microarchitectural structures worst case fault scenario may not always appear with a fault in the most accessed entry. Also is important to mention that for the return-address-stack we selected the entry that was popped most frequently, including wrong-path pops.

The stuck-at fault is injected in the most significant bit position of the predicted addresses by the line predictor and the return-address-stack. The upper bits of addresses usually exhibit less variance than lower bits and a hard-fault may cause more degradation since the potential for disagreement can be higher.

We simulated all the SPEC CPU 2000 benchmarks using reference inputs for a 100M committed instruction interval. An in-house SimPoint [10] like tool is used to select the region to simulate. Table 2 shows the basic statistics of the benchmarks used in this study.

4.1 Line Predictor and Return-Address-Stack

A line predictor [7, 12] is a non-architectural structure that is used in the fetch stage to provide every cycle a next line prediction to enable continuous instruction fetch. The price for fast line prediction is lower accuracy. As a result a line predictor is backed-up in the next stage with a more accurate branch predictor. When the branch predictor disagrees with the line predictor the processor trusts the branch predictor and re-steers the fetch to the branch predicted program counter. Every time such a disagreement occurs there

Parameter description	Setting
Line Predictor	6 KiB, 2-way
RAS	64 entries
Pipeline depth	15 stages
Fetch/Decode/Issue/ Commit width	up to 4 instructions per cycle
Issue Queue	40 INT entries, 20 FP entries
Functional Units	4 INT ALUs, 4 INT mult/div, 1 FP ALUs, 1 FP mult/div
Reorder buffer	128
Branch Predictor	hybrid: 4 KiB meta, 4 KiB bimodal, 8 KiB gshare (15 bits history)
L1 instruction-cache	64 KiB, 2-way, 64 B blocks, LRU, 1-cycle latency
L1 data-cache	64 KiB, 2-way, 64 B blocks, LRU, 3-cycle latency
L2	unified: 2 MiB, 4-way, 64 B blocks, LRU, 10-cycle hit latency, 255 cycles miss latency

Table 1. Baseline Processor

Benchmark	Fast fwd (billion)	IPC baseline	LinePred usage	RAS usage
ammp	4.95	1.493	8937878	20623
applu	2.10	0.401	650121	100
apsi	1.65	2.164	3635749	57742
art	3.15	1.879	8617394	110
bzip2	42.95	1.160	12731599	329579
crafty	0.95	1.900	11591903	1091659
eon	26.40	1.191	11757704	2024150
equake	19.25	0.378	3791795	1061219
facerec	36.55	1.152	7138174	166354
fma3d	10.25	1.424	19922052	1434416
galgel	4.45	2.404	5849044	0
gap	20.60	1.103	14550003	2051862
gcc	8.40	0.895	8445648	478339
gzip	19.40	0.989	13857405	315047
lucas	2.65	0.513	1333339	0
mcf	13.40	0.107	27992306	3331666
mesa	0.45	1.739	9015327	1187486
mgrid	0.15	0.566	397340	327
parser	1	1.056	16810721	1987964
perlbmk	13.8	1.239	14383221	1351772
sixtrack	8.20	1.860	2289412	128
swim	1.15	0.287	2302776	66
twolf	7.20	1.108	12673833	704713
vortex	18.55	1.594	16970902	2054990
vpr	25.55	1.515	11041773	646494
wupwise	7.95	1.493	10615089	651842

Table 2. Benchmark summary: number of fast-forwarded instructions in billions, number of committed instructions that used line predictor and return-address-stack, respectively.

is a cycle penalty. A line predictor fault can degrade performance because it can increase the number of disagreements between the line predictor and the branch predictor. Note that in the next section we report line mispredictions but actually we refer to disagreements between the line predictor and the branch predictor for committed branch instructions.

A return-address-stack is used to push the return-address every time we encounter a call in the front-end of the processor and pop a return-address every time we encounter a return. Clearly, a return-address-stack fault can increase the number of return-address mispredictions. The return misprediction penalty is significantly larger than a line misprediction because it can be detected only at writeback and for the pipeline used in this work this will be at least a 12 cycles.

5 Results

In this section we present our results about the performance implications of stuck-at faults in the line predictor and the return-address-stack.

5.1 Line Predictor

Figure 4 shows the speedup in the presence of a single stuck-at hard-fault as compared to a baseline without hard-faults. For the cell stuck-at small degradations are observed, with maximum 3.3% for *art00*. When the single stuck-at fault occurs in the output of the line predictor, the performance impact is more detrimental. The results show that more benchmarks suffer from a significant degradation, half of the benchmarks slow down by more than 3%, with the worst case, for *gap00*, experiencing a 19% slowdown.

Figure 5 shows the effect of hard-faults on the mispredictions per kilo instructions (MPKI) in the line predictor. By comparing the MPKI with the performance degradation results in Figure 4, and the benchmarks baseline IPC in Table 2 the following observations can be made.

A single cell fault typically increases misprediction marginally over a baseline scheme with no hard-faults. This indicates that usually no few entries in a line predictor are responsible for many of the line-predictions in a program. This explains why the performance with a cell fault only degrades minimally.

As expected, output bit faults cause more mispredictions than cell faults. Unexpectedly, the data show that a large increase in misprediction may not imply large performance loss. For *mcf00* this is due to very low baseline IPC, and a low misprediction penalty, that make the added overhead insignificant as compared to the overall execution time of *mcf00*. The small degradation can also be attributed to subtle interactions between the different microarchitectural units where a

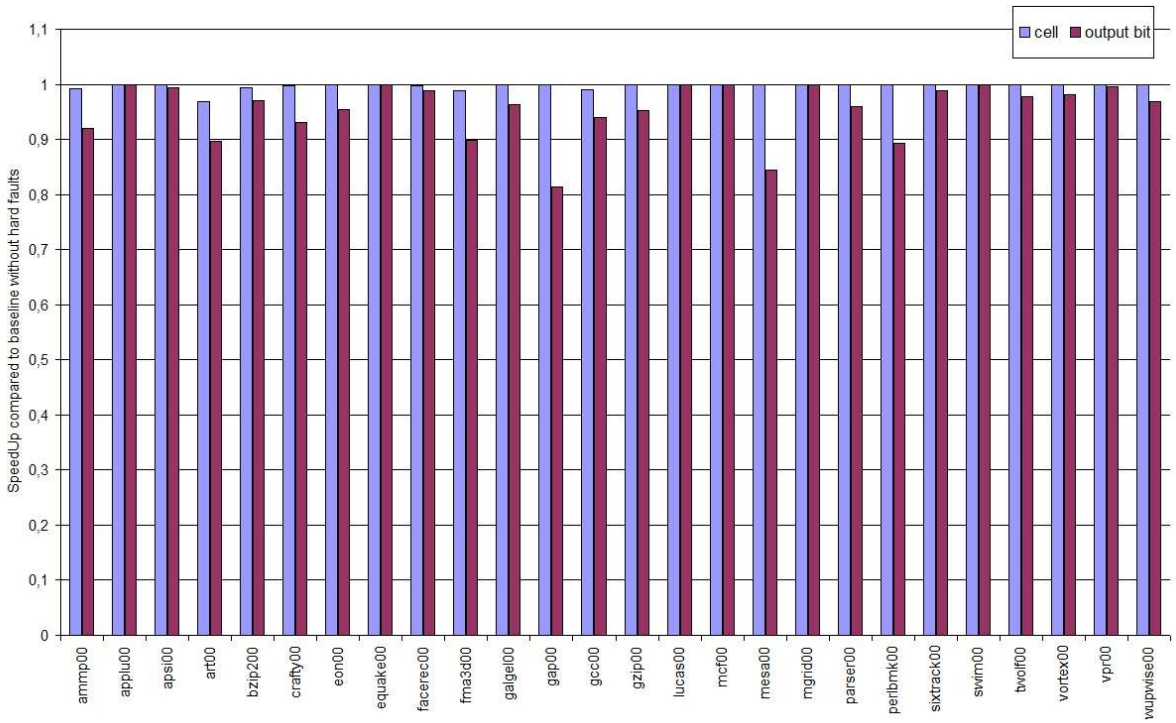


Figure 4. Speedup compared to baseline without hard faults in line predictor.

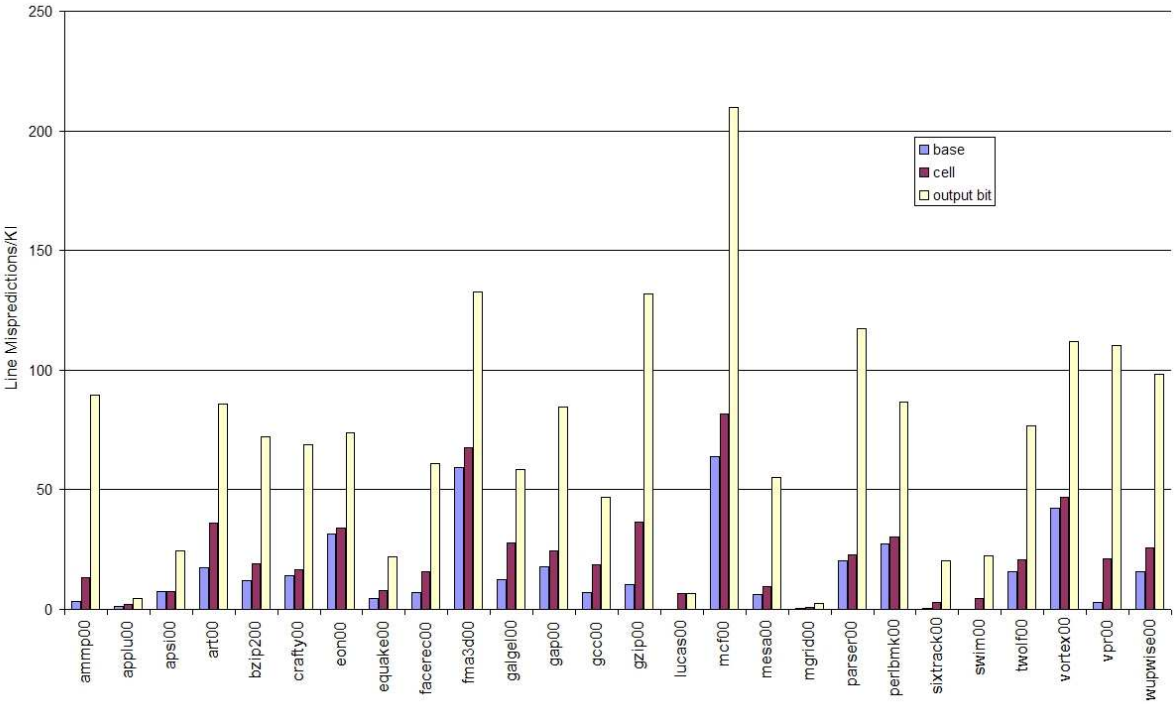


Figure 5. Line Mispredictions per kilo instructions.

degradation in the one can be useful to the other's performance. For example, additional line mispredictions may allow branch predictor state to be more up-to-date and improve its overall accuracy.

Overall, the data suggest that protection against hard-faults in a line predictor may be important for output bits but not for cells.

5.2 Return-Address-Stack

Now, lets consider how stuck-at hard-faults influence the performance of a return-address-stack. Figure 6 illustrates the degradation compared to a baseline without errors and Figure 7 shows the effect of hard-faults on the MPKI in the return-address-stack.

As expected, only benchmarks with a significant number of committed returns are affected by faults (see Table 2). The impact can be very high even with a single cell fault, up to 9% for *mesa*. This is due to the small number of entries of a return-address-stack that results in many of the entries to have frequent accesses. The performance decrease for output bit stuck-at is even more pronounced, up to 34% for *vortex00*.

A comparison of the MPKI of the line predictor and the return-address-stack reveals that line mispredictions are larger and yet their performance degradation is smaller. The reason for this is the longer misprediction penalty of return instructions as compared to line predictions. Recall that this penalty is at least 12 times longer for a return instruction.

Overall, the data suggest that return-address-stack is much more vulnerable to performance degradation and therefore may need to be protected against all kind of hard-faults in future designs.

6 Related Work

As stated in the introduction most of previous hard-fault research has concentrated on the impact of correctness. However, a few papers touched on performance implications before.

Sohi [20] studied the performance impact of cache organization with disabled portions, such as ways and sets. The goal of that work was to improve yield without noticeable performance degradation. Related research was performed by Pour and Hill [16] to study the performance impact of manufacturing faults in caches. The work by [16] quantified the performance impact in an isolated way through the cache miss ratio. Lee *et al.* [13] also explored various masking strategies for manufacturing hard-faults in caches. The authors measure performance degradation by disabling cache lines, sets, ways, ports or even the complete cache. In our work, we go beyond architectural structures, and quantify the degradation of performance due to wear-out in non-architectural structures.

Bower *et al.* [4] described a technique to tolerate hard faults in array structures. Their mechanism is able to detect faults, and once detected it maps out the faulty array elements through an additional level of indirection. This achieves almost fault-free performance with a single hard-fault, and they reported a slight speedup when 8 hard-faults were injected. For non-architectural structure they focus on a branch history table, and they examine stuck-at-1 cell faults only. In this respect, we augment their study by considering faults in output bits of a line predictor and a return-address-stack.

A recent study by Li and Yeung approaches the implications of faults from an application perspective [14]. They show somce applications, e.g. multimedia applications, can tolerate up to a certain amount of faults, and still be acceptable to the user.

Recently, Abella *et al.* [1] augmented an adder tree with a transistor called fuse to anticipate faults in arithmetic units. The idea is to stress the fuse so that it will fail short before the first transistor in use.

7 Conclusion

This paper investigates the performance implications of hard faults in non-architectural structures. Non-architectural hard-faults do not affect correctness but they can degrade a processor performance at a low level and therefore may make them as important to deal with as architectural hard-faults.

This work shows, using previously proposed analytical models, under what temperature conditions hard-faults in non-architectural structures are likely to occur in the same order of magnitude as hard-faults in architectural units.

A worst-case analysis of the performance degradation of a high performance processor due to a single cell stuck-at fault in two prediction structures—return-address-stack and line predictor—revealed in some cases a degradation up to 9%. The performance decrease goes up to 34% when considering a hard-fault on one of the output bits.

The above findings underline the importance to continue the study of the performance implications of non-architectural hard-faults. Our future work will perform a comprehensive quantification of the performance impact of different types of hard-errors (such as cell or line stuck-at-X) on various structures including branch direction, branch target and memory dependence predictors, lru bits used for replacement, and units used

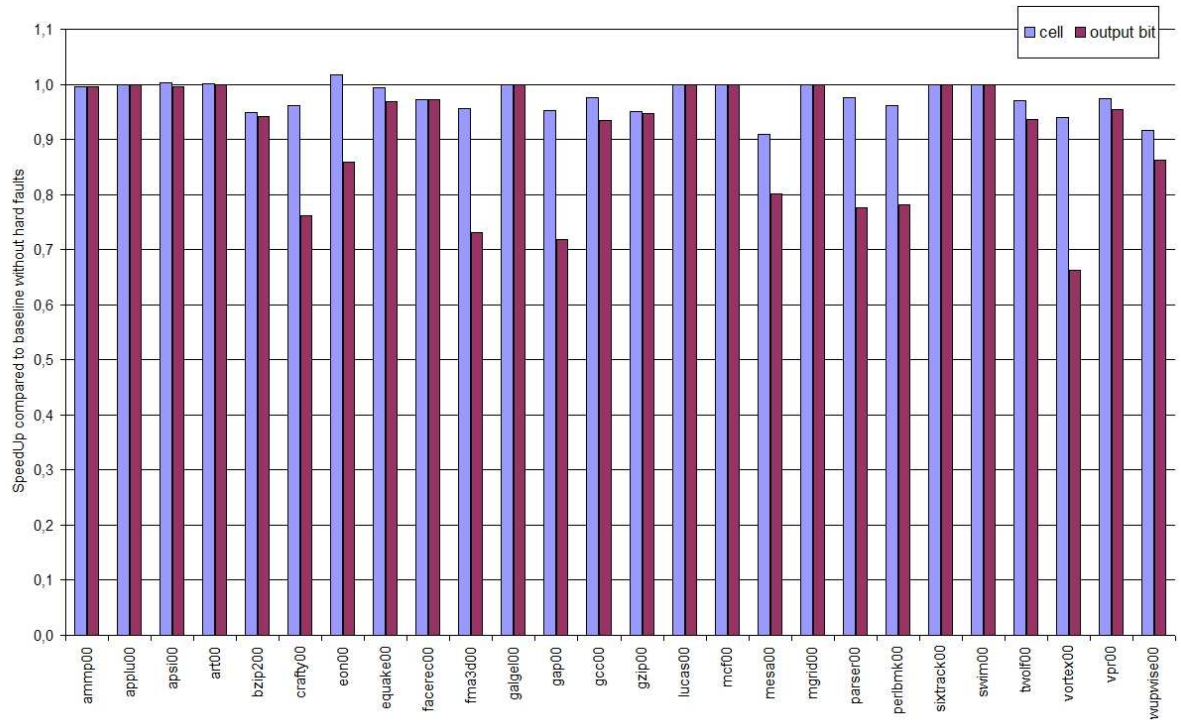


Figure 6. Speedup compared to baseline without hard faults in return-address-stack.

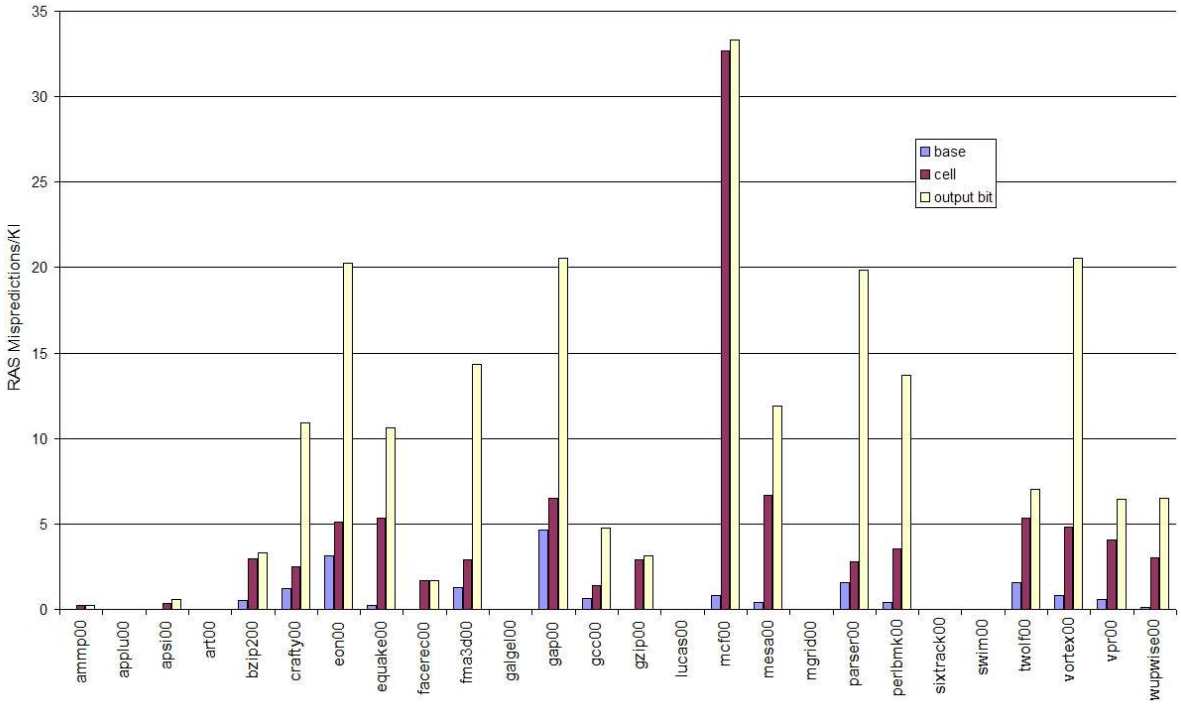


Figure 7. Return-address-stack misprediction per kilo instructions.

for target calculation. Both worst and average case analysis will be performed in our future work. In addition, we will explore the use of low-overhead detection and correction techniques for non-architectural hard-faults that are found to be more performance critical, to ensure future processors can operate with minimal degradation at the presence of non-architectural hard-faults. We will leverage existing reconfiguration techniques that have been proposed for error detection/correction of architectural structures, but, non-architectural resources may provide a distinct opportunity for simpler detection and correction techniques since they do not require a full repair.

Acknowledgments

This work is partially supported by Ghent University, University of Cyprus, HiPEAC, the European SARC project No. 27648 and Intel. Yiannakis Sazeides would like to credit Elli Demetriou and Constantinos Kourouyiannis for the preliminary studies leading to this work.

References

[1] J. Abella, X. Vera, O. Unsal, O. Ergin, and A. González. Fuse: A technique to anticipate failures due to degradation in ALUs. In *13th IEEE International On-Line Testing Symposium*, pages 15–22, July 2007.

[2] T. Austin. DIVA: A dynamic approach to microprocessor verification. *Journal of Instruction-Level Parallelism*, 2, May 2000.

[3] T. M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *Proceedings of the 32nd Annual International Symposium on Microarchitecture*, pages 196–207, Nov. 1999.

[4] 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*, pages 51–60, June 2004.

[5] F. A. Bower, D. J. Sorin, and S. Ozev. A mechanism for online diagnosis of hard faults in microprocessors. In *Proceedings of the 38th Annual International Symposium on Microarchitecture*, pages 197–208, Nov. 2005.

[6] F. A. Bower, D. J. Sorin, and S. Ozev. Online diagnosis of hard faults in microprocessors. *ACM Transactions on Architectural Code Optimization*, 4(2), June 2007.

[7] B. Calder and D. Grunwald. Fast and accurate instruction fetch and branch prediction. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 2–11, June 1994.

[8] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. BulletProof: A defect-tolerant cmp switch architecture. In *Proceedings of the 12th International Symposium on High Performance Computer Architecture*, pages 3–14, Feb. 2006.

[9] R. Desikan, D. Burger, S. Keckler, and T. Austin. Sim-alpha: a validated execution driven alpha 21264 simulator. Technical report, Department of Computer Sciences, University of Texas at Austin, 2001.

[10] G. Hamerly, E. Perelman, and B. Calder. How to use simpoint to pick simulation points. In *ACM SIGMETRIC Performance Evaluation Review*, 2004.

[11] Failure mechanisms and models for semiconductor devices. *JEDEC Solid State Technology Association*, Mar. 2006.

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

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

[14] X. Li and D. Yeung. Application-level correctness and its impact on fault tolerance. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, pages 181–192, Feb. 2007.

[15] 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*, 5(3):419–427, Sept. 2005.

[16] A. F. Pour and M. D. Hill. Performance implications of tolerating cache faults. *IEEE Transactions on Computers*, 42(3):257–267, Mar. 1993.

[17] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, and T. Austin. Ultra low-cost defect protection for microprocessor pipelines. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2006.

[18] D. P. Siewiorek, R. S. Swarz, and A. K. Peters. *Reliable computer systems (3rd ed.): design and evaluation*. Ltd, 1998.

[19] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 2–13, June 2003.

[20] G. S. Sohi. Cache memory organization to enhance the yield of high performance VLSI processors. *IEEE Transactions on Computers*, 38(4):484–492, Apr. 1989.

[21] L. Spainhower and T. Gregg. IBM S/390 parallel enterprise server G5 fault tolerance: A historical perspective. *IBM Journal Research and Development*, 43(5/6):863–874, 1999.

[22] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The impact of technology scaling on lifetime reliability. In *Proceedings of the 34th Annual International Conference on Dependable Systems and Networks*, pages 177–186, June 2004.

[23] 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*, pages 520–531, June 2005.

[24] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Lifetime reliability: Toward an architectural solution. *IEEE Micro*, 25(3):70–80, May 2005.

[25] S. Zafar, B. Lee, J.Stathis, A. Callegari, and T. Ning. A model for negative bias temperature instability (NBTI) in oxide and high-K pFETs. In *2004 Symposium on VLSI Technology*, pages 208–209, June 2003.