Mitigating the Performance Degradation due to Faults in Non-Architectural Structures

Constantinos Kourouyiannis[*] Yiannakis Sazeides[*] Veerle Desmet[+]

[*]Dept. of Computer Science
University of Cyprus, Nicosia

[+]Dept. of Electronics and Information Systems
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 microarchitectural studies on processor reliability and yield improvement aim to solve the problem for architectural resources.

Faults in non-architectural resources received little attention because they do not affect correctness. However, faults in non-architectural structures can degrade processor performance and may need to be addressed to ensure acceptable performance levels, in particular for applications where performance is of paramount importance, e.g. in real time systems that can not afford missing deadlines.

This work first quantifies the performance implications of faults in two non-architectural structures: a line-predictor and a return-address-stack. A simulation based analysis of a high-end processor that experiences faults in 25% of the cells in the line-predictor and the return-address-stack revealed an average performance degradation of 5%. Next, we engineered a hardware protection scheme that combines a low-cost fault detection and repair through address remapping. This scheme can recover most of the performance loss when faults are present, while it rarely degrades performance when no faults exist.

Introduction

Current computer technology scaling trends are leading us toward smaller feature size and larger transistor budgets per chip. These developments force the designer to address the issue of dependable operation with little or no performance degradation in the presence of faults.

Our work quantifies the performance implications of faults in two non-architectural structures to determine whether such structures merit protection against faults. In particular, the structures we examine are a line-predictor and a return-address-stack. Additionally, we propose and evaluate the effectiveness of a simple fault detection and repair scheme. The proposed scheme detects during execution that there are significant accesses to defective entries and remaps the accesses to different locations to reduce the number of accesses to defective entries.

Low-cost fault detection

We propose a low-cost approximate detection scheme for predictors where a counter is associated per data bit position with separate counters used for the input and output data. The input counters are updated when predictions are written to the array and the output counters are updated when predictions are read from the array. At regular intervals the counters are checked to detect possible faults in the array. This is done by searching for any large difference between the input and output counter at each bit position. As such the counters are used to track the balance of the ones being written and read in an array per bit position. We refer to the threshold used to decide that an absolute difference value is large as the *delta-threshold*. The counters are reset after each interval.

The size of the counters and the value of the delta-threshold can have an impact as they may lead to false positives and false negatives being detecting a problem when there is none and not detecting a problem when there is one respectively. For lower cost, we can use one instead of two counters per bit position and increment at the input and decrement at the output on the same bit value. To further reduce cost, we can share counters across bit positions.

The above detection scheme can detect frequent accesses to faulty entries but does not provide information as to which are the faulty entries. This compromise is made in an effort to keep the overhead of detection as low as possible.

Address Remapping

The idea of remapping logical to physical locations to avoid faults have been proposed before but, as far as we know, all previous work required knowledge of which entries are defective and the remapping function mapped a previously defective entry to either a non-defective entry or to a spare [1, 2]. The remapping function we employ simply redistributes the accesses, i.e. there are no spares to remap to, and no two entries that mapped to different locations before remapping are mapped together after the remapping.

An analysis of the access distributions of prediction structures shows that very few entries are responsible for the majority of correct predictions for the line predictor and the return-address-stack. Therefore, when accesses to faulty entries are detected by the fault detection unit, it may be possible for a remapping function to remap frequently accessed faulty entries to rarely accessed entries without a fault. In this ideal case, the performance will be as if there were no faults and no further remapping is needed. Of course, it is possible for remapping to make things worse, i.e. increase the number of faulty accessed entries. In that case, we rely on the detection mechanism to detect that and perform another remapping. This process can be allowed to repeat forever. We found, however, that it can lead to worse performance than no-remapping. Consequently, to prevent this unending detection-and-remapping cycle a throttling mechanism can be employed to turn-off remapping if too many remappings are performed over a short time.

Results

We extended the validated cycle accurate simulator *sim-alpha* [3] to measure the performance of a high performance out-of-order superscalar processor with and without faults.We quantify the performance implications with increasing number of faults in the line-predictor and the return-address-stack for three scenarios: worst-case, best-case and average-case. Assuming n faults, the worst-case scenario injects n faults in the top n entries that gave the most correct predictions. For the best-case the n faults are injected in the entries that gave the least correct predictions. To determine the average-case performance we use an analytical approach.

The average-case results for the line predictor show that with 512 defective entries, without repair, no benchmark suffers more than 3% slowdown. However, when faulty entries increase to 1024, the average-case performance degradation can be up to 6%. The results for the return-address-stack follow the same trend.

Finally, we did experiments to evaluate the effectiveness of our proposed detection-repair scheme where the faults are injected in random locations. The results shows that for the majority of the runs the proposed scheme works, i.e. when there is degradation due to faults it is usually recovered and when there is no degradation the performance does not get worse.

References

[1] 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.

[2] A. Das, S. Ozdemir, G. Memik, J. Zambreno, and A. Choudhary. Mitigating the effects of process variations: Architectural approaches for improving batch performance. In *Workshop on Architectural Support for Gigascale Integration*, 2007.

[3] 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.