Performance Implications of Faults in Prediction Arrays

Nikolas Ladas[1], Yiannakis Sazeides[1], and Veerle Desmet[2]

[1] University of Cyprus
[2] Ghent University, Belgium

Abstract. 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 overall processor performance significantly and, therefore, may render them as important to deal with as architectural faults.
In this work we quantify the performance impact of faults in several non-architectural array structures found in a high-end processor. The analysis reveals that performance can degrade significantly, up to 53%, even with a small fraction—0.5%—of faulty bits. Also, it is shown that not all structures are equally vulnerable to faults. In particular, for the processor configuration used in this study, the return-address-stack and the conditional branch predictor are found to be the most sensitive whereas the replacement arrays of the instruction and data cache are the most resilient.

1 Introduction

As technology scaling trends are leading us toward smaller feature sizes and larger transistor budgets per chip, faults are becoming more frequent [3]. These developments present to the processor designer both opportunities and challenges. One of these challenges is to provide reliable operation with little or no performance degradation in the presence of faults. Feature miniaturization and narrower design margins are increasing the vulnerability of chips to faults. For instance, process variation is known to result in as much as 30% variation in maximum frequency and 20x variation in leakage power [3]. Process variation also increases performance unpredictability and heterogeneous performance in multicores.

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 [1,12,15,18]. With future technology projections pointing to increased process variability and faults, a more general need for 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.

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 [5,8,16]. 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, due to more mispredictions and misses, 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.

Let us also stress that if architectural resources are well-protected and the frequency of faults keeps increasing eventually non-architectural resources, if left unprotected, will become the performance bottleneck.

In this paper we investigate the significance of protecting non-architectural arrays from faults. We quantify the performance implications of faults in six different non-architectural array-based units of a high-end out-of-order processor: LRU array for the level-1 instruction cache, LRU array for the level-1 data cache, line-predictor, memory dependence predictor, conditional branch predictor and the return-address-stack. The results show that performance can degrade significantly even with a small fraction of faulty bits. Also, it is shown that not all units are equally vulnerable to faults.

The remainder of this paper is organized as follows. Section 2 discusses fault modeling of non-architectural array structures and further motivates the need for protection of non-architectural structures from faults. Section 3 overviews the used methodology, Section 4 discusses the experimental results, and Section 5 points at related work before Section 6 concludes the paper.

2 Faults in Non-Architectural Structures

Non-architectural structures can be divided into arrays, such as prediction arrays, and random logic, such as an adder used for computing the address of prefetched data. The focus in this paper is on faults in *non-architectural arrays*. A fault in an array may occur in several places like bit-line, word-line, cell, driver, decoder etc. This work considers only faulty cells.

A large number of non-architectural array units can be found in modern processors: arrays used for prediction, such as a branch direction predictor, LRU arrays useful for guiding replacement in various set-associative tables, and hysteresis arrays used to control updates of various predictors.

The study focuses on the impact of permanent faults, such as those due to manufacturing imperfections and wear-out, and faults caused by process-variability. Transient faults, with short duration, are not an issue for non-architectural resources, because they occur very infrequently and when they occur they can be corrected as soon as the structure gets updated.

The impact of faults in non-architectural structures can vary widely depending on the fault-model, the microarchitecture, and the architectural properties of programs.

2.1 Fault Model

The choice of the model used to study the effects of faults in non-architectural arrays can have a significant bearing on our observations. There are several

parameters that need to be considered: (i) the number of faults, (ii) the location of the faults in the array, and (iii) the fault model of each fault. The more faults the more potential for degradation. Also, a fault in a frequently used entry is likely to have a bigger performance impact than a fault in an infrequently accessed entry. Similarly, a cell stuck-at-1 may have more impact if the bit stored in the cell is more biased toward zero.

The physical principles that will determine the value of the above parameters in the future are poorly understood and difficult to predict accurately. So we make the following assumptions about them.

Regarding the number of faults it is important to consider scenarios covering a range of faults from small to large. In this work we will consider scenarios where 0.125% and 0.5% of the cells of non-architectural arrays are faulty. 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 entries [22].

As far as the physical fault distribution, it is generally known that faults, at least due to manufacturing [12] and process variation [20], are distributed according to a *random* component—expressing the non-determinism of fault locations—and a *spatial* component—expressing the clustering of faults. In this work we consider faults at random locations where the fault cluster size is one cell. The effects of multi-cell fault clusters will be investigated in subsequent work.

It is understood that various faulty 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 [7]. We assume stuck-at faults with each faulty cell assigned randomly either 0 or 1.

2.2 Non-architectural Arrays and their Fault Semantics

Today's high end processor 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), dependence between load and store instructions, cache hit/miss, cache way and cache bank etc. In addition, non-architectural arrays are used to guide replacement in set associative caches (LRU bits) or to guide the updating of predictors (hysteresis bits).

Although faults can occur in any of these arrays, the semantics of a faulty cell varies 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.

Faults in LRU replacement arrays may cause the LRU block not to be replaced. For example, for a 2-way cache the LRU can be implemented using a single bit. The sets with faulty LRU bit are effectively converted from 2-way to a direct-mapped sets. This can lead to an increase in the cache misses. However, cache miss latency, especially for level-one, can be hidden with out-of-order execution.

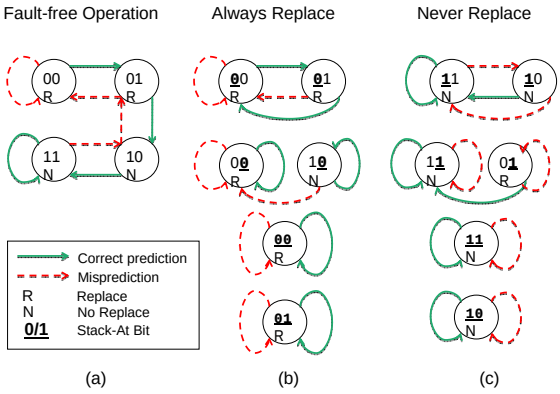


Fig. 1. *Fault Semantics of a 2-bit saturating counter used for update hysteresis*

A fault can have different implications even within a given array. A faulty cell in a memory dependence predictor, such as the one used in [11], may force an independent load to wait until all previous stores commit. This is a significant penalty but not as serious as a pipeline flush since independent instruction in an out-of-order processor can proceed to execution and may hide some of this delay. On the other hand, a fault that allows a load with a dependence to a store in the instruction window to proceed to execution causes a pipeline flush.

In the same vein, a fault in a 2-bit hysteresis array mainly results in two behaviors, always replace on a misprediction or never replace. Both behaviors can degrade performance but the second can be more grave. This is illustrated in Fig. 1. The two-bit saturating counter is shown Fig. 1(a) while Fig. 1(b)-(c) show how the state machine is transformed with different combinations of stuck-at values. For example, if the most significant bit is stuck-at-0 then the machine can only transition between states 00 and 01. Assuming that the most significant bit signifies whether to replace or not, then for the stuck-at fault cases shown in Figure. 1(b) a fault causes on a misprediction to (almost)always replace. As a result, the hysteresis counter is not protecting its corresponding entry from spurious update. Whereas the reduced states in Fig. 1(c) do not permit the entry to get updated.

The above discussion reveals that performance is not equally vulnerable to faults across non-architectural units and even within a unit. Asymmetry, also exists due to the inherent higher accuracy of some units, like the return address stack. Furthermore, there is variation due to differences in the dynamic behavior between programs, such as the instruction mix and predictability. For example, a program with no return instructions or with low branch predictability will not suffer significantly from faults in the return-address-stack or the conditional branch predictor. Consequently, to assess the performance impact of faults in non-architectural arrays and determine which arrays are more vulnerable to faults we perform simulation studies that we will present in Section 6.

Parameter description	Setting
Pipeline depth	15 stages
Superscalarity	4
Line Predictor	6.5 KB, 4096 entries, 11 bit prediction + 2 bit hysteresis/entry
RAS	16 entries
Branch Predictor	8 KB gshare (15 bits history)
Fetch/Decode/Issue/Commit	up to 4/4/6/4 instr. per cycle
Branch Resolution	In-order
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
L1 instr. cache	64 KB, 2-way, 64 B blocks, 1-cycle, LRU
L1 data cache	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 1. Baseline Processor

3 Methodology

We extended the validated cycle accurate simulator *sim-alpha* [9] to measure the performance implications of faults in non-architectural arrays of a high performance out-of-order superscalar processor for which key parameters are summarized in Table 1. We simulated all the SPEC CPU 2000 benchmarks using reference inputs and report each time results for 100M committed instructions. An in-house SimPoint-like tool is used to select the regions to simulate.

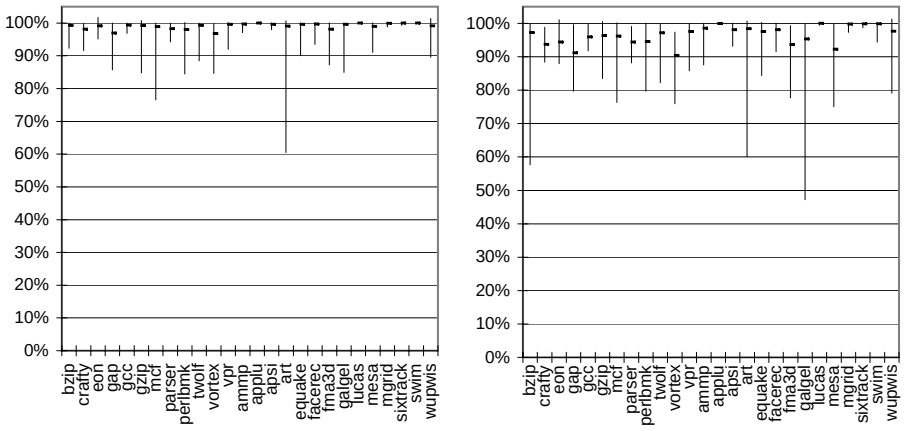
All performance numbers are always normalized to the baseline performance obtained with the same configuration but without faults and no remapping. The *normalized performance* is a higher-is-better metric with results below 100% meaning performance degradation.

The experiments study the impact of randomly injected faults in six non-architectural structures, namely conditional branch predictor, line-predictor, return-address-stack, store-wait predictor, and replacement arrays for the L1 instruction and L1 data cache. We consider scenarios where the fraction of faulty cells is 0.125% and 0.5%.

For each non-architectural array and for each of these fractions, we randomly generate 50 different fault maps which are used throughout this study. The number of faulty bits for each fraction is dependent on the size of the array and shown in Table 2.

	Entr.xBits/Entry	TotBits	0.125%	0.5%
Gshare Direction Predictor [14]	32768x2	65,536 bits	82	328
Line Predictor Array [6]	4096x11	45,056 bits	56	225
Line Predictor Hysteresis Array [6]	4096x2	8,192 bits	10	41
Memory dependence predictor [11]	1024x1	1,024 bits	1	5
Return address stack [21]	16x31	496 bits	1	3
LRU array for 2-way I$	512x1	512 bits	1	3
LRU array for 2-way D$	512x1	512 bits	1	3

Table 2. Faults injected per structure (in number of bits).



(a) 0.125% faults (b) 0.5% faults

Fig. 2. *Performance degradation per benchmark for various percentage of faults.*

4 Results

In this section we present our results starting with the performance degradation due to faults in non-architectural arrays, then ranking the non-architectural structures according to their vulnerability to faults.

4.1 Performance Degradation with All Non-Architectural Units Faulty

Fig. 2 shows the average, as well as the minimum and maximum performance per benchmark when **all** arrays are faulty. The fraction of faults in the non-architectural arrays ranges from 0.125% faulty bits in Fig. 2(a) to 0.5% in Fig. 2(b). The data for each benchmark and fraction of faults is obtained over runs with 50 different random fault-maps.

Results reveal that most of the benchmarks, both INT and FP, can suffer from substantial degradation. Even with a small fraction of faulty bits, some programs experience up to 39% (*art*) degradation with 0.125% faults and 53% (*galgel*) with 0.5% of faults. The average degradation over all benchmarks is 1% and 3.5% when the fraction of faults is 0.125% and 0.5%, respectively. Few benchmarks remain insensitive even with 0.5% faults. This is due to their instruction mix that does not include many instructions, like returns and indirects, that access faulty non-architectural units or have very slow execution dominated by memory accesses.

One might argue that 3.5% average performance degradation is not significant and does not motivate protecting non architectural structures. However, Fig. 2 shows that it is possible, albeit more rare, to have much more severe performance degradation. At 0.125%, for ten benchmarks, performance drops below 10%. At 0.5% ten benchmarks see their performance drop below 20%. Also, for both 0.125% and 0.5% there are are cases where performance drops more than 30%. This behavior can be explained by examining the access patterns of the

predictor arrays. Due to locality effects, some benchmarks only use only a small fraction of a given predictor array's entries. For example, in the case of *galgel*, 14 entries are responsible for 90% of all gshare accesses. For a program with this behavior, a single fault may cause severe performance degradation. We believe that this will be a likely senario since it happened 10 times in our 50 runs for 0.5% faults.

These observations lead us to believe that dealing with faults in non architectural structures might become a necessity in future generation processor design.

Next we will try to determine which of the non-architectural units are more sensitive to faults.

4.2 Criticality of non-architectural Arrays

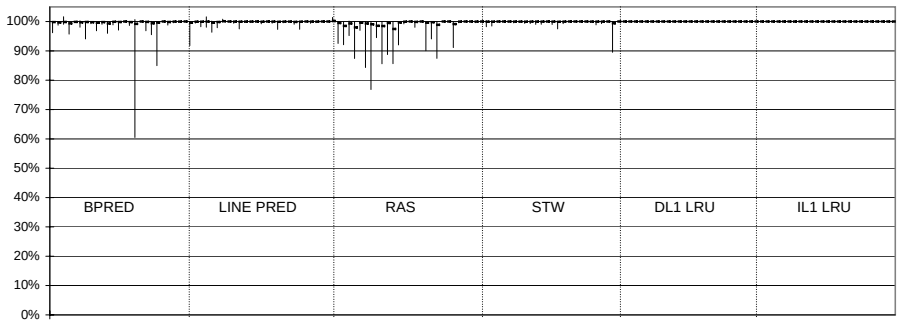
In Fig. 3 we report the performance degradation when only one non-architectural array is faulty at a time. This can help isolate the vulnerability of each non-architectural array to faults. Like in Fig. 2 we report results for 0.125% and 0.5% faults. Each graph is divided into 6 regions, one for each structure we examine. The x-axis in the graph omits the benchmark names, which for each region are in the same order as in Fig. 2.

The results clearly show that, for the processor configuration used in this study, the various non-architectural units are not equally vulnerable to the same fraction of faults.

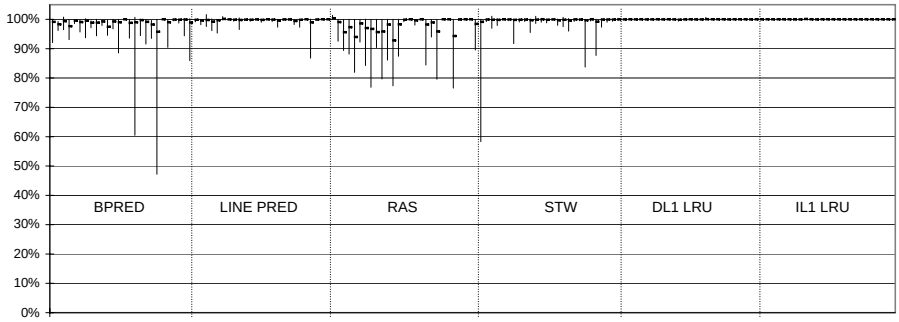
The most critical non-architectural array is the return-address-stack which degrades performance even with a single faulty bit (0.125%). The branch direction predictor becomes critical as soon as 0.5% faults are present. However, even at 0.125% it is possible for performance to drop more than 35%. Line-predictor and memory-dependence predictor may require protection when there are more faults, whereas the LRU arrays from the 2-way associative data and instruction caches seems not sensitive even with more faults. The latter suggests that having a fraction of the sets for the two L1 caches being direct mapped it does not affect the performance. Section 2 offers several causes that can lead to the observed diverse vulnerability among units.

5 Related work

As stated in the introduction most of previous fault research has concentrated on the impact of correctness. However, a few papers touched on performance implications before. Sohi [19] 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 [17] to study the performance impact of manufacturing faults in caches. The work by [17] 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 in non-architectural structures, and propose a generic technique to minimize the detrimental effect of faults.



(a) 0.125%



(b) 0.5%

Fig. 3. *Performance Vulnerability of the various Non-Architectural Structures.*

Bower *et al.* [5] investigate the performance effects of up to 8 faulty entries in a branch history table. Their conclusion is that it is not worthwhile protecting this table against hard faults as performance degradation is negligible. Also, Makris *et al.* [2] evaluated the effect of a single fault in the most frequently accessed entry of a conditional branch predictor. Our paper considers the performance impact for several structures, and ranks them according to the level of protection they may require in the near future where more faulty bits are expected.

In a recent paper, Wilkerson *et al.* [22] present a way to trade off power for reliability by reducing voltage while giving up 25-50% of the cache capacity.

Hamdioui *et al.* overview in detail the various models for fault types that can occur in memory cells [10]; we have presented a technique that can protect a non-architectural structure against any of these fault types.

A direction of relavant work is concerned with the testing and validation of mechanisms aiming to enhance performance [4]. This underlines the importance of mitigating faults in prediction arrays. However, these ealrier works did not evaluate the performance implications of faults.

6 Conclusions

This paper argues that it may be important to consider the protection of non-architectural structures against faults in the future because they can significantly degrade performance. While there is a plethora of research on protecting the architectural units to guarantee correctness, non-architectural units received little attention. We quantify the performance implications of faults for six different non-architectural units and showed trends in the presence of a small to moderate fraction of faulty bits. The analysis of the data shows that performance can degrade considerably even with a small fraction of faults. The results also indicate that the least vulnerable structures, are the LRU arrays in 2-way associative L1-caches, and the most sensitive to faults are the return-address-stack and branch direction predictor.

Acknowledgments

This work is partially supported by the University of Cyprus, Ghent University, HiPEAC, and Intel.

References

1. F. J. Aichelman. Fault-tolerant design techniques for semiconductor memory applications. *IBM Journal of Research and Development*, 28(2):177–183, Mar. 1984.
2. S. Almukhaizim, T. Verdel, and Y. Makris. Cost-effective graceful degradation in speculative processor subsystems: The branch prediction case. *Computer Design, International Conference on*, 0:194, 2003.
3. 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*, pages 338–342, June 2003.
4. P. Bose. Testing for function and performance: Towards anintegrated processor validation methodology. *J. Electron. Test.*, 16(1-2):29–48, 2000.
5. F. A. Bower, P. G. Shealy, S. Ozev, and D. J. Sorin. Tolerating hard faults in microprocessor array structures. In *Proceedings of the 34th International Conference on Dependable Systems and Networks*, pages 51–60, June 2004.
6. B. Calder and D. Grunwald. Next cache line and set prediction. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 287–296, May 1995.
7. Q. Chen, H. Mahmoodi, S. Bhunia, and K. Roy. Modeling and testing of SRAM for new failure mechanisms due to process variations in nanoscale CMOS. In *23rd IEEE VLSI Test Symposium*, pages 292–297, May 2005.
8. A. Das, S. Ozdemir, G. Memik, J. Zambreno, and A. Choudhary. Microarchitectures for managing chip revenues under process variations. *Computer Architecture Letters*, 6, June 2007.
9. R. Desikan, D. Burger, S. Keckler, and T. Austin. Sim-alpha: a validated execution driven Alpha 21264 simulator. Technical Report TR-01-23, CS Dept., University of Texas at Austin, 2001.

10. S. Hamdioui, Z. Al-Ars, A. J. van de Goor, and M. Rodgers. Linked faults in random access memories: concept, fault models, test algorithms, and industrial results. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(5):737–757, 2004.
11. 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.
12. I. Koren and C. M. Krishna. *Fault Tolerant Systems*. Morgan Kaufmann Publishers Inc., 2007.
13. H. Lee, S. Cho, and B. R. Childers. Performance of graceful degradation for cache faults. In *IEEE Computer Society Symposium on VLSI*, pages 409–415, Mar. 2007.
14. S. McFarling. Combining branch predictors. Technical Report TN-36, Digital Western Research Laboratory, June 1993.
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. M. Mutyam and V. Narayanan. Working with process variation aware caches. In *Proceedings of the 2007 Design, Automation and Test in Europe Conference*, pages 1152–1157, Apr. 2007.
17. A. F. Pour and M. D. Hill. Performance implications of tolerating cache faults. *IEEE Transactions on Computers*, 42(3):257–267, Mar. 1993.
18. D. P. Siewiorek, R. S. Swarz, and A. K. Peters. *Reliable computer systems (3rd ed.): design and evaluation*. Ltd, 1998.
19. 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.
20. B. Vinnakota and J. Andrews. Repair of rams with clustered faults. In *ICCD1992*, pages 582–585, Oct. 1992.
21. C. F. Webb. Subroutine call/return stack. *IBM Technical Disclosure Bulletin*, 30(11):221–225, Apr. 1988.
22. 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 International Symposium on Computer Architecture*, pages 203–214, June 2008.