Faults in Data Prefetchers: Performance Degradation and Variability

Nikos Foutris Athanasios Chatzidimitriou Dimitris Gizopoulos
Department of Informatics & Telecommunications
University of Athens
Athens, Greece
{nfoutris, achatz, dgizop}@di.uoa.gr

John Kalamatianos Vilas Sridharan
AMD Research RAS Architecture
Advanced Micro Devices, Inc.
Boxborough, MA, USA
{john.kalamatianos, vilas.sridharan}@amd.com

Abstract—**High-performance microprocessors employ data prefetchers to mitigate the ever-growing gap between CPU computing rates and memory latency. Technology scaling along with low voltage operation exacerbates the likelihood and rate of hard (permanent) faults in technologies used by prefetchers such as SRAM and flip flop arrays. Faulty prefetch behavior does not affect correctness but can be detrimental to performance. Hard faults in data prefetchers (unlike their soft counterparts which are rare) can cause significant single-thread performance degradation and lead to large performance variability across otherwise identical cores. In this paper, we characterize in-depth both of these aspects in microprocessors suffering from multiple hard faults in their data prefetcher components. Our study reveals fault scenarios in the prefetcher table that can degrade IPC by more than 17%, while faults in the prefetch input and request queues can slow IPC up to 24% and 26%, respectively, compared to fault-free operation. Moreover, we find that a faulty data prefetcher can substantially increase the performance variability across identical cores: the standard deviation of IPC loss for different benchmarks can be more than 4.5%.**

Keywords—performance degradation; performance variability; permanent faults; dependable performance; microarchitectural simulators

I. INTRODUCTION

Multi-core microprocessor architectures dominate most application domains. The inherent unreliability of deep nanometer-scale technologies [4] [5] and near-threshold voltage (NTV) operation [1] [26] [27] increase the vulnerability of SRAM array cells and flip flops to hard faults; if unaddressed, these faults will impose significant constraints on microprocessor design.

To hide the latency of memory accesses, computer architects integrate multi-layer cache memories along with sophisticated data prefetchers [14] [15]; both structures can occupy noticeable silicon real estate. Data prefetch designs predict the flow of data and correspondingly boost instructions per cycle (IPC) by reducing the stalls due to cache misses. The most widely used class of data prefetch mechanisms, the stride data prefetchers, has been shown to be highly effective for scientific, multi-media, desktop and engineering applications [13].

Technology modeling in resilience roadmaps predicts that the failure probability of SRAM cells will be higher in 16nm and 12nm nodes [22]. NTV operation exacerbates permanent faults in SRAMs due to exposing variations across cells [26]. Thus, most reliability studies have focused on caches due to the area they occupy and their immediate impact on both functional correctness and performance [2] [3] [16] [19] [20]. Foutris et al. are the only ones who have measured the impact of single faults in data prefetchers [8].

Unlike cache memories, data prefetchers do not affect program correctness because they do not modify program state. However, hard faults in prefetcher arrays can degrade performance significantly, by (a) reducing training opportunities, and therefore decreasing the number of generated prefetch requests (reduce prefetch coverage); (b) issuing prefetch requests later or earlier than the fault-free case (degrade prefetch timeliness); and, (c) perturbing the prefetch address-generation logic (reduce prefetch accuracy). In [8], Foutris et al. reported that more than 48% of single hard faults in SRAM cells of a conservative data prefetcher model can degrade performance up to 3%. In many-core designs, faults in data prefetcher arrays will trigger imbalances in the data stream sent to memory system, leading to inter-core performance variability. This is an undesirable property for the data-center deployment [1], especially since NTV operation can be an attractive candidate for low power servers [26]. According to Total-Cost-of-Ownership (TCO) estimation frameworks such as the one in reported in [10], performance variability negatively affects system cost, power consumption and worsens the system's environmental impact. Finally, performance variability is undesirable in the mobile and desktop markets [12], because it reduces the ability to provide performance guarantees for real-time applications.

We visualize the motivation of this paper in Figure 1 where we present the performance variability in a multi-core design with faulty data prefetchers in different cores, suffering from the same number of faults and executing the same benchmark (GemsFDTD). The IPC difference between the worst and best cases is 17%, while standard deviation ranges from 1.9% to 4.5%. Thus, a faulty data prefetcher can significantly increase the variability across otherwise identical cores.

In this paper, we contribute the following:

- We measure the performance impact of *multiple* permanent faults on an L1 stride data prefetcher.

- We measure the performance impact of single permanent faults in the prefetcher's supporting queues.
- We evaluate the degree of performance variability among cores caused by faulty L1 stride data prefetchers.

Our results show that the performance loss is up to 26% (on average, 0.5% when quintuple faults are injected into the prefetch table array, and 1.5% and 2.5% when a single fault is injected into the prefetch input and request queues, respectively). Meanwhile, the performance variability can be more than 26% compared to the fault-free case (standard deviation of IPC loss between benchmarks ranges from 0.01% to 4.5%).

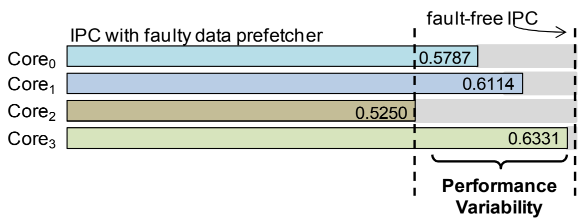

Figure 1: Performance variability across 4 identical cores for GemsFDTD with five hard faults in the data prefetcher of each core.

II. Background and Related Work

The impact of hard faults in data prefetchers was first measured by Foutris et al. [8]. Their work however is limited to single hard faults on prefetch tables (PT) and does not include multiple faults in PTs, faults in supporting logic (*Prefetch Input Queu*e - PIQ and *Prefetch Request Queue* - PRQ) and performance variability. Previous work on SRAM hard faults focused on caches but not on prefetchers. Abella *et al.* [1] proposed disabling/re-mapping to guarantee predictable performance at low voltages. Agarwal *et al.* [2] focused on yield improvements tolerating process variations. Ansari *et al.* [3] proposed the Zerehcache architecture to deal with massively defective caches. Chishti *et al.* [7] employed special types of error-correcting codes to improve lifetime reliability. Performance implications with disabled cache parts were discussed in [16], [19]. Roberts *et al.* [20] proposed cache-line merging techniques, and Wilkerson *et al.* [24] employed cache-line combining/disabling to survive voltage scaling.

Recent work either shows a large number of hard faults in SRAM arrays that operate in near-threshold voltages [26], [27], or projects high fault rates in 16-nm and 12-nm processes [22]. In both contexts, the single-bit failure probability (P_{fail}) of SRAM cells is expected to fall between 10^{-6} and 10^{-4} [22]. Under a binomial probability distribution, such failure rates would result in multiple hard faults per SRAM array. Circuit-level techniques such as wordline boosting [18] can be employed to reduce these probabilities; however, such techniques add complexity and area to the array design.

III. Experimental Setup

First, we calculate the cumulative probability of multiple faults in a typical 10Kbit SRAM array for a stride data prefetcher. Based on the single cell probabilities presented in [22] and a bimodal distribution we draw, in Figure 2, the cumulative probability for four technology nodes (32nm, 22nm, 16nm, 12nm). In 32nm, the multiple fault probability remains very low between 10^{-4} and 10^{-5}. Moving towards deeper nanometer-scale technologies the cumulative probability converges to 1.0 with 5 faults. Thus, we experiment with up to 5-tuple faults in a data prefetcher.

We then perform a comprehensive statistical fault-injection campaign on top of the PTLsim x86 architectural simulator [25]. We employ the same microprocessor configuration as in [8], enhanced with a L1 data stride prefetcher with 64 entries, tracking strides up to 5 bits wide (32 cache lines). An 8-entry PIQ and an 8-entry PRQ allow for queueing training and prefetch requests. The prefetch table is assumed to be direct mapped and PC-indexed.

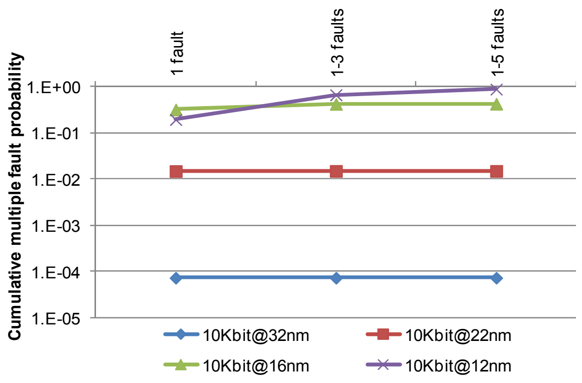

Figure 2: Cumulative probability of 1…k faults for 10K bit SRAM arrays in four technologies. Future technologies at 16m, and 12nm manufacturing process converge to probability 1.0 when 1…5 faults exist in the structure.

Each training event can issue 2 prefetch requests at Address+Stride and Address+2*Stride. We use a statistical fault-injection framework (with a confidence level of 99% and an error margin of 3%, according to the sampling described in [17]) that includes a faults database populated with the fault descriptions (component, entry, bit, type) for the L1 cache-stride data prefetcher. We use the stuck-at fault model [21] in which a faulty cell permanently stores logic 0 or 1. A total of 900 different fault masks (100 single, 300 triple, 500 quintuple faults) are injected in the data prefetcher component (the total number of fault injection runs for the 29 SPEC CPU2006 benchmarks are 26,100).

Each fault-injection run applies randomly [1] selected multiple fault masks to the sub-arrays (fields) of the prefetch table (i.e., tag, load address, stride, confidence, LRU, and valid arrays) with the exception of the prefetch input queue (PIQ) and the prefetch request queue (PRQ), into which we inject single faults only due to these structures' small sizes.

We run all SPEC CPU2006 benchmarks, simulating the largest-weight 100-million-instruction SimPoint sample per benchmark with a 20-million-instruction warm-up. We

[1] We chose not to insert hard faults in a regular pattern because experimental data, indicating clustered fault patterns, are associated with soft errors only.

compare the results of each injection experiment to fault-free execution. To measure the average performance degradation, we look at the average IPC impact per fault group size (one, three, and five; based on Section II, one, three, and five faults have high probabilities of occurring on a 10K-bit SRAM array, which is the size of the prefetcher array of this study). To examine performance variability, we calculate the maximum value and standard deviation of IPC loss per fault group size across all benchmarks.

IV. Experiments

A. *Prefetch-friendly and prefetch-neutral Benchmarks*

We profile the full set of SPEC CPU2006 benchmarks to measure the IPC impact of a fault-free data prefetcher. On average, we find that the data prefetcher boosts IPC by 6.85%. However, performance improvement varies and depends on the memory access patterns generated by each benchmark. Thus, we classify benchmarks into two categories: (a) prefetch-friendly with IPC change greater than the average speed-up; and (b) prefetch-neutral with IPC change less than the average speed-up as shown in Table III.

B. *Performance Impact of Faults*

We measure the performance impact of hard faults in the data prefetcher. Figure 3 shows the average and maximum IPC slow-down (due to faults) when 1, 3 and 5 faults are injected in the prefetch table along with standard deviation; the upper diagram shows prefetch-friendly benchmarks and the lower shows prefetch-neutral ones. Prefetch-friendly benchmarks show a maximum 3.0% IPC loss due to single fault runs per benchmark, 5.7% IPC loss over triple faults, and 9.2% IPC loss over quintuple faults. Thus, a faulty L1 cache-stride data prefetcher can severely degrade microprocessor performance.

Figure 4 and Figure 5 show the average and standard deviation of the IPC slow-down for each SPEC CPU2006 benchmark when one, three, and five faults are injected. As expected, the prefetch-friendly benchmarks appear to have a greater IPC impact with the same number of faults compared to the prefetch-neutral ones. In particular, even though a fault-free prefetcher improves execution time in GemsFDTD by 20% and in sphinx3 by 0.6% (Table III), GemsFDTD suffers a maximum 17% IPC slow-down, while sphinx3 loses only 0.06% when quintuple-faults are injected.

To clarify the severity of the performance loss due to the faulty prefetcher, we also show in Table III the IPC of SPEC CPU2006 benchmarks for: (a) the CPU core *without* an L1 cache stride data prefetcher, and (b) the same core *with* a fault-free L1 cache stride data prefetcher enabled. If we compare the prefetcher IPC gain in Table III with the IPC loss in Figure 4 and 5, we see that 9 out of 29 benchmarks lost the IPC gain from data prefetching. For example, on bzip2, the IPC without the data prefetcher was 1.074. When quintuple faults were injected into the prefetch table array, IPC was reduced to 1.069 (we see similar behavior in: gamess, GemsFDTD, libquantum, tonto, cactusADM, povray, sjeng, omnetpp).

By further analyzing the prefetcher behavior, we found that the extent of the performance impact that faults have, depends on the distribution of the training input addresses across the prefetch table entries (apart from the prefetch-friendliness of the workload, as discussed on IV-A).

Table I presents the activity per entry in the prefetcher table. Each column shows the number of prefetch table entries that is trained with less than or equal to X% of all addresses training the prefetcher. X is 50%, 75% and 100% for the 2nd, 3rd and 4th columns respectively. For example, in libquantum, a single entry is trained by 50% of the traffic, 2 entries by 75%

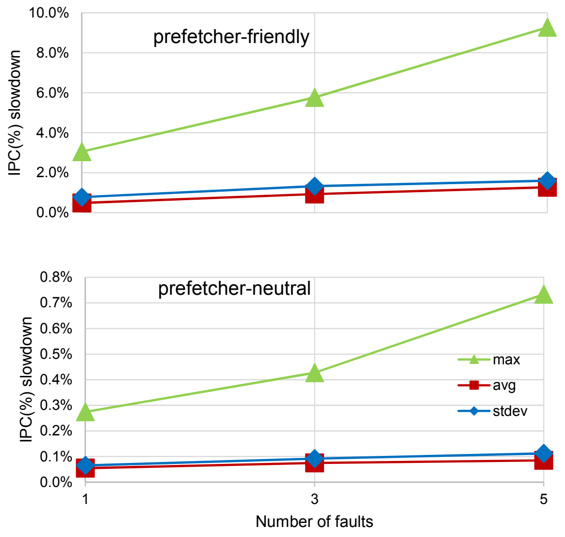

Figure 3: IPC loss: prefetch-friendly (upper) and prefetch-neutral (lower).

and 5 entries by 100% of the memory traffic (libquantum speed-up is 17.20%, while slow-down is up to 13.80% when quintuple faults are injected). In contrast, the prefetch training address stream generated by gcc is distributed across the entries of the prefetch table array (gcc speed-up is 1.89%, while slow-down is up to 0.06%). Therefore, in gcc, the probability of polluting the majority of the training addresses by a given number of injected faults is low. In libquantum, if a fault occurs in one of the heavily used entries, the majority of training will be affected, and so the IPC loss will be much greater.

The prefetch-neutral benchmarks show a much more uniform usage of the table entries, compared to the prefetch-friendly ones, where most of the training is concentrated in a few entries (33 vs. 51 entries on average, respectively). This is because even though their memory traffic does not exhibit as many strides, they still allocate table entries and attempt to train. Furthermore, the prefetch-neutral benchmarks generate few prefetch requests per prefetch table entry because there aren't many strides detected. Therefore, when few entries are trained by the majority of memory traffic (for example, in cactusADM the entire memory traffic is directed to only 11 entries), the impact on IPC is negligible, since the actual generated prefetch requests are few.

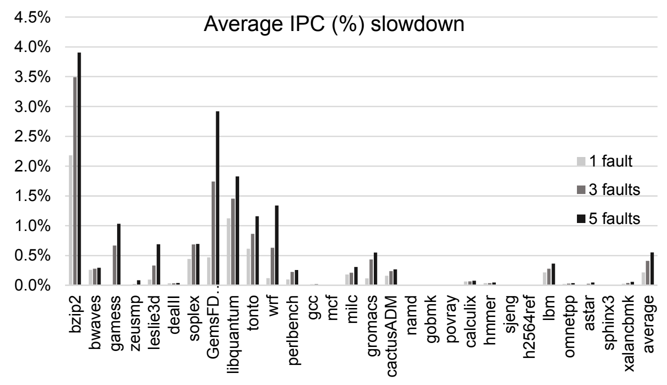

Figure 4: Average IPC slow-downs for 1, 3 and 5 hard faults on Prefetch Table.

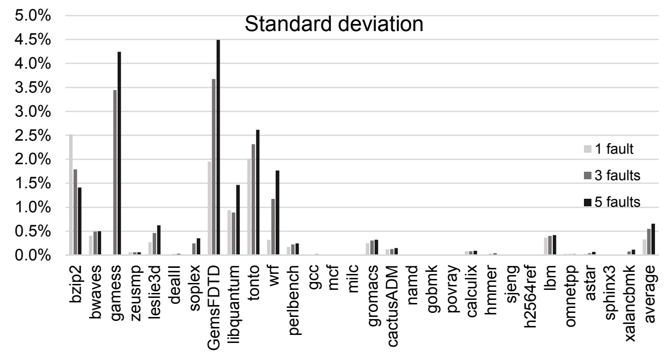

Figure 5: IPC slowdown standard deviations for 1, 3 and 5 hard faults on Prefetch Table.

As we can observe in Figure 6 the faulty prefetcher is throttled because the faults reduce the number of training events. As a result, the number of issued prefetch requests drops for all benchmarks (on average, the number of issued prefetch requests drops from 22 to 20 per 1,000 committed instructions); therefore, performance gains due to prefetching are lower (the average L1 cache miss rate roughly increases from 26 to 27 MPKI in the quintuple injected fault scenario). The data in Figure 6 and Figure 4 also illustrate the greater performance sensitivity of the prefetch-friendly benchmarks to faults. Faults in the prefetch table corrupt the prefetch addresses sent to memory, which in turn increases the L1 cache miss rate and hurts IPC.

We also looked at a variety of events that can be used to identify at run-time *when* faults in the data prefetcher lead to significant IPC loss. We found that an off-range stride is such an event and is triggered when the difference between the trained address and a new memory address is out of the legal stride limits. As a result, the incoming memory address is dropped, since it fails to train the stride data prefetcher.

Table II presents the number of off-range stride events per 1K committed instructions for the fault-free and the faulty microprocessor models (i.e., 1, 3 and 5 faults injected into the prefetch table array). In particular, multiple permanent faults increase the amount of off-range stride events up to 25% for the prefetch-friendly benchmarks (off-range stride rate increased from 118.7 to 148.3 per 1K commits) and up to 8% for the prefetch-neutral benchmarks (ranging from 231.3 to 249.3 per 1K commits).

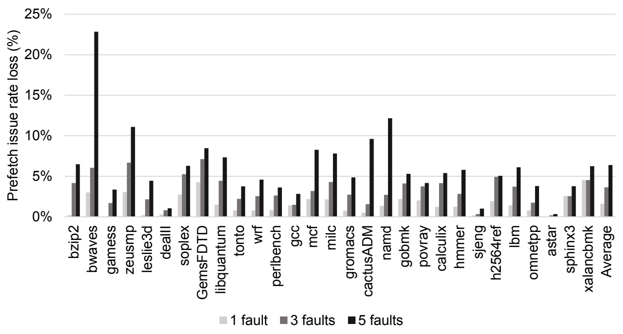

Figure 6: Percentage of prefetch issue rate losses with 1,3 and 5 hard faults compared to the fault free prefetch issue rate

The severe impact of faults on the performance of SPEC CPU2006 benchmarks indicates the need to develop and integrate fault detection schemes. By monitoring off-range stride occurrences per prefetch table entry, one could steer the design of fault detection mechanisms for data prefetchers, since the increase (up to 25%) in the number of off-range stride events correlates well with the number of faults in the prefetcher.

We also performed a fault injection campaign in the PIQ and PRQ. Due to the small size of PIQ and PRQ (8 entries each), we injected only single faults in them. This was sufficient to show the severe impact on performance that hard faults on these queues can have on microprocessor performance. Even though these queues are typically implemented with flip flops, the probability of single faults, especially in NTV mode remains significant. Table III shows the IPC change for single faults injected into the PIQ and PRQ.

Across all 29 benchmarks, the average IPC loss (1.5% and 2.5% for PIQ and PRQ, resp.) and maximum IPC loss (24.3% and 26.3% for PIQ and PRQ, resp.) are significantly higher than that of the prefetch table because a large number of training addresses (buffered in PIQ) and prefetch requests (queued in PRQ) are likely to be polluted by a single hard fault. It is evident that having a faulty PRQ or PIQ severely slows-down performance (13 benchmarks out of the 29 lost the speed-up gained by the data prefetcher due to a faulty PRQ entry and 11 due to a faulty PIQ entry).

The fault location determines the extent of the performance impact. Figure 7 shows the average utilization of each entry of the PRQ and PIQ (percentage of times a given entry of the queue is utilized). In particular, the PIQ entries are utilized uniformly across all benchmarks with the exception of the top 2 entries. The top three entries in PRQ are utilized 95% of the time across all benchmarks. Therefore, faults that reside in the rear entries of both queues have minimal impact on performance.

Our analysis clearly confirms that microprocessor performance can be severely degraded by a faulty L1 cache-stride data prefetcher. The impact can be more than 24% IPC

Table I: TRAINING ACTIVITY OF THE PREFETCH TABLE ENTRIES. NUMBER OF PREFETCH TABLE ENTRIES THAT HANDLE LESS THAN 50%, LESS THAN 75% AND 100% OF THE MEMORY TRAFFIC TRAINING THE PREFETCHER.

Prefetch Table Entries Training Activity							
Prefetch-friendly	*≤ 50%*	*≤ 75%*	*=100%*	***Prefetch-neutral***	*≤ 50%*	*≤ 75%*	*=100%*
bzip2	1	2	7	perlbench	3	4	63
bwaves	4	7	12	gcc	7	20	64
gamess	17	17	17	mcf	3	5	47
zeusmp	9	25	62	milc	1	2	23
leslie3d	9	17	47	gromacs	12	22	58
dealII	1	2	24	cactusADM	4	6	11
soplex	8	18	63	namd	2	8	57
Gems FDTD	4	7	32	gobmk	3	9	64
quantum	1	2	5	povray	6	14	64
tonto	3	6	40	calculix	4	8	63
wrf	12	22	59	hmmer	6	10	59
				sjeng	3	6	64
				h2564ref	8	20	64
				lbm	1	2	32
				omnetpp	1	2	31
				astar	4	6	47
				sphinx3	1	2	57
				xalancbmk	2	3	62
average	**6**	**11**	**33**	**average**	**4**	**8**	**51**

slow down due to a single fault in the PIQ, up to 26% IPC slow down due to a single fault in the PRQ, and up to 18% IPC slow down due to five faults in the prefetch table, mainly due to prefetch throttling and/or cache pollution.

C. *Performance Variability Summary*

In this section, we examine the performance variability across identical CPU cores due to multiple faults in their data prefetchers. Performance variability can affect the cost and the power budget of a data center. Our findings show the following

The maximum IPC slow-down can be up to 7% for single faults in the prefetch table, 24% for the PIQ, and 26% for the PRQ. For most benchmarks, maximum IPC loss is higher than the IPC gain of the fault-free data prefetcher. This large variation in IPC finding holds when all cores are affected by the same number of faults.

The difference in IPC slow-down for different numbers of faults (1 to 5) across the cores ranges between 0.005% and 17% compared to the fault-free IPC, when considering only faults in the prefetch table. The same range is 0.01 to 24% for PIQ faults and 0.02% to 26% for PRQ faults.

The difference between the best- and worst-case performance for single and multiple faults is not due to outlier behavior. The standard deviation of the IPC loss on the prefetch-friendly benchmarks due to faults in the prefetcher table is 0.7%, 1.3%, and 1.6% for single, triple, and quintuple faults, respectively.

As Figure 5 shows, certain benchmarks have even larger stdev values: for example the stdev of bzip2 benchmark with single fault injected is 2.5%, while the gamess benchmark stdev is 3.4% and 4.2% for triple and quintuple faults, respectively.

Table II: NUMBER OF OFF-RANGE STRIDE EVENTS FOR THE PREFETCH-FRIENDLY AND NEUTRAL BENCHMARKS WITH A FAULT-FREE AND WITH 1, 3, 5 FAULTS INJECTED IN THE PREFETCH TABLE.

Off-range stride (per 1K commits)									
Prefetch-friendly	***fault-free***	***1 fault***	***3 faults***	***5 faults***	***Prefetch-neutral***	***fault-free***	***1 fault***	***3 faults***	***5 faults***
bzip2	81	84	91	98	perlbench	176	179	185	191
bwaves	200	215	218	232	gcc	451	454	463	474
gamess	1	9	15	21	mcf	30	34	47	61
zeusmp	58	62	70	79	milc	126	131	139	149
leslie3d	163	164	174	184	gromacs	305	306	311	317
dealII	162	167	185	197	cactus	154	156	162	171
soplex	185	197	254	257	namd	345	346	349	352
Gems FDTD	232	239	249	258	gobmk	323	326	332	337
quantum	200	210	227	240	povray	344	347	350	358
tonto	7	13	22	34	calculix	56	58	63	69
wrf	19	22	26	32	hmmer	220	226	239	251
					sjeng	114	128	142	151
					h2564ref	181	181	189	196
					lbm	414	414	418	422
					omnetpp	314	316	321	328
					astar	106	121	126	133
					sphinx3	309	311	316	321
					xalanc bmk	195	202	207	208
average	**119**	**126**	**139**	**148**	**average**	**231**	**235**	**242**	**249**
Overall average						**175**	**181**	**191**	**199**

The standard deviation of IPC loss for the prefetch-neutral benchmarks is 0.06%, 0.07%, and 0.08% (Figure 5). The standard deviation of the IPC drop for all single faults injected into PIQ and PRQ is 2.0% and 2.4%, respectively.

The key message regarding performance variability is that hard faults in data prefetchers in a many-core system significantly increase inter-core performance variability.

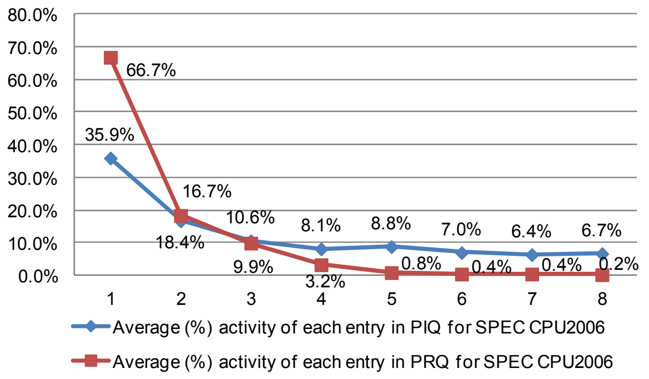

Figure 7: Utilization of the PIQ and PRQ entries across all SPEC CPU2006 benchmarks.

Table III: AVERAGE IPC FOR PREFETCH-FRIENDLY AND –NEUTRAL BENCHMARKS, WITHOUT THE DATA PREFETCHER, WITH A FAULT-FREE DATA PREFETCHER AND WITH SINGLE FAULTS INJECTED INTO THE PREFETCH INPUT AND REQUEST QUEUE.

Average IPC									
Prefetch-friendly	*w/o pref*	*w pref*	*PIQ fault*	*PRQ fault*	***Prefetch-neutral***	*w/o pref*	*w pref*	*PIQ fault*	*PRQ fault*
bzip2	1.07	1.29	1.04	1.00	perlbench	1.66	1.71	1.68	1.67
bwaves	0.65	0.71	0.71	0.71	gcc	0.69	0.71	0.70	0.70
gamess	1.74	2.12	1.82	1.87	mcf	0.21	0.21	0.21	0.21
zeusmp	0.97	1.06	0.96	0.97	milc	0.76	0.73	0.70	0.66
leslie3d	0.78	0.84	0.80	0.82	gromacs	0.94	0.97	0.95	0.94
dealII	0.99	1.08	1.07	1.05	cactus	1.46	1.46	1.45	1.45
soplex	0.55	0.60	0.57	0.55	namd	1.55	1.56	1.55	1.55
Gems	0.53	0.63	0.48	0.51	gobmk	1.23	1.24	1.24	1.24
quantum	0.40	0.46	0.38	0.34	povray	1.13	1.14	1.13	1.13
tonto	1.60	1.86	1.57	1.57	calculix	1.13	1.18	1.18	1.17
wrf	0.79	1.07	0.87	0.82	hmmer	1.17	1.18	1.17	1.16
					sjeng	1.18	1.18	1.18	1.18
					h2564ref	1.55	1.56	1.54	1.56
					lbm	0.69	0.72	0.71	0.70
					omnetpp	0.51	0.51	0.50	0.50
					astar	0.91	0.94	0.93	0.93
					sphinx3	1.37	1.37	1.37	1.37
					xalanc	1.10	1.14	1.10	1.11

V. CONCLUSIONS

The existence of hard faults in a stride data prefetcher can affect microprocessor performance significantly and increase inter-core performance variability. Our analysis shows that the performance loss due to hard faults in the prefetch table can be up to 17%, and up to 24% and 26% for the PIQ and PRQ respectively. Also, performance variability across cores is increased: the standard deviation of IPC loss between benchmarks can be more than 4.5%. Our findings imply that prefetchers should be supported by fault protection mechanisms in both current and forthcoming technologies. Events, such as the rate of off-range strides, can steer the implementation of fault detection mechanisms.

VI. ACKNOWLEDGMENTS

This paper has been supported by the FP7 Program of the European Union through the CLERECO Project (Grant Agreement FP7-611404) and by an AMD Research Grant. AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other names used herein are for identification purposes only and may be trademarks of their respective companies. Early version of this work has been presented at the 10th workshop on Silicon Errors in Logic – System Effects (SELSE 2014).

REFERENCES

[1] J. Abella, J. Carretero, P. Chaparro, X. Vera, A. Gonzalez, “Low Vccmin Fault-Tolerant Cache with Highly Predictable Performance,” MICRO, 2009.

[2] A. Agarwal, B.C. Paul, H. Mahmoodi, A. Datta, K. Roy, “A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies,” IEEE Trans. on VLSI Systems,” vol. 13, no. 1, pp. 27-38, January 2005.

[3] A. Ansari, S. Gupta, S. Feng, S. Mahlke, “Zerehcache: Armoring Cache Architectures in High Defect Density Technologies,” MICRO, 2009.

[4] R. Blish, T. Dellin, S. Huber, M. Johnson, J. Maiz, B. Likins, N. Lycoudes, J. McPherson, Y. Peng, C. Peridier, A. Preussger, G. Prokop, L. Tullos, “Critical Reliability Challenges for The Int’l Technology Roadmap for Semiconductors (ITRS),” March 2003 [Online] http://www.itrs.net/ Links/2003ITRS/LinkedFiles/PIDS/4377atr.pdf.

[5] S. Borkar, “Designing reliable systems from unreliable components: The challenges of transistor variability and degradation,” IEEE Micro, 25(6):10–16, 2005.

[6] F.A. Bower, P.G. Shealy, S. Ozev, D.J. Sorin, “Tolerating Hard Faults in Microprocessor Array Structures,” DSN, 2004.

[7] Z. Chishti, A.R Alameldeen, C. Wilkerson, W. Wu, S.-L. Lu, “Improving Cache Lifetime Reliability at Ultra-low Voltages,” MICRO, 2009.

[8] N. Foutris, D. Gizopoulos, J. Kalamatianos, V. Sridharan, “Assessing the impact of hard faults in performance components of modern microprocessors,” ICCD, 2013.

[9] N. Hardavellas, I. Pandis, R. Johnson, N.G. Mancheril, A. Aillamaki, B. Falsafi, “Database servers on chip multiprocessors: Limitations and opportunities,” Proceedings of the 3rd CIDR, January 2007.

[10] D. Hardy, M. Kleanthous, I. Sideris, A. Saidi, E. Ozer, Y. Sazeides, “An Analytical Framework for Estimating TCO and Exploring Data Center Design Space”, ISPASS, 2013.

[11] http://www.amd.com/us/products/server/processors/2100seriesplatform/ Pages/x2150seriesprocessors.aspx

[12] C. Hughes, P. Kaul, S.V. Adve, R. Jain, C. Park, J. Srinivasan, “Variability in the Execution of Multimedia Applications and Implications for Architecture,” ISCA 2001.

[13] S. Iacobovici, L. Spracklen, S. Kadambi, Y. Chou, S.G. Abraham, “Effective stream-based and execution-based data prefetching,” ICS 2004.

[14] B. Jacob, S. Ng, D. Wang, *Memory Systems: Cache, DRAM, Disk*, Morgan Kaufmann, 2008.

[15] V. Jimenez, R. Gioiosa, F.J. Gazorla, A. Buyuktosunoglu, P. Bose, F.P. O’Connell, “Making data prefetch smarter: adaptive prefetching on POWER7,” PACT 2007.

[16] H. Lee, S. Cho, B.R. Childers, “Performance of Graceful Degradation for Cache Faults,” ISVLSI, 2007.

[17] R. Leveugle, A. Calvez, P. Maistri, P. Vanhauwaert, “Statistical Fault Injection: Quantified Error and Confidence,” DATE, 2009.

[18] Y. Pan, J. Kong, S. Ozdemir, G. Memik, S.W. Chung, “Selecting Wordline Voltage Boosting for Caches to Manage Yield under Process Variations,” DAC, 2009.

[19] A.F. Pour, M.D. Hill, “Performance implications of tolerating cache faults,” IEEE Trans. on Computers, vol. 42, no. 3, pp. 257-267, March 1993.

[20] D. Roberts, S.K. Nam, T. Mudge, “On-chip cache device scaling limits and effective fault repair techniques in future nanoscale technology,” Microprocessors and Microsystems, vol. 32, no. 5-6, pp. 244-253, August 2008.

[21] B.F. Romanescu, D.J. Sorin, “Core Cannibalization: Improving Lifetime Chip Performance for Multicore Processors in the Presence of Hard Faults,” PACT, 2008.

[22] S.R. Nassif, N. Mehta, Y. Cao, “A Resilience Roadmap,” DATE, 2010.

[23] D. Sánchez, Y. Sazeides, J.L. Aragon, J.M. Garcia, “An Analytical Model for the Calculation of the Expected Miss Ratio in Faulty Caches,” IOLTS, 2011.

[24] C. Wilkerson, G. Hongliang, A.R. Alameldeen, Z. Chishti, M. Khellah, L. Shih-Lien, “Trading off Cache Capacity for Reliability to Enable Low Voltage Operation,” ISCA, 2008.

[25] M. Yourst, “PTLsim: A cycle Accurate Full System x86-64 Microarchitectural Simulator,” ISPASS, 2007.

[26] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, T. Mudge: “Near-Threshold Computing: Reclaiming Moore’s Law through Energy Efficient Integrated Circuits”, Proceedings of the IEEE, February 2010.

[27] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar, “Near-threshold voltage (NTV) design: Opportunities and challenges,” DAC 2012.