Performance Assessment of Data Prefetchers in High Error Rate Technologies

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

John Kalamatianos
AMD Research

Vilas Sridharan
RAS Architecture

Advanced Micro Devices, Inc.
Boxborough, MA, USA
{john.kalamatianos, vilas.sridharan}@amd.com

***Abstract*—Modern microprocessors are equipped with an arsenal of speculative mechanisms, such as data prefetchers, to mitigate ever-growing memory access latency. Aggressive technology scaling along with near-threshold voltage operation exacerbates the likelihood and rate of hard faults not only in large arrays such as caches but in speculative components as well. While data prefetchers do not affect correct processor operation, they are critical for performance and faults in them can cause significant performance degradation and variability across otherwise identical cores. The impact of hard faults in data prefetchers has not been quantified accurately.**

In this paper, we quantify the performance and variability impact of hard faults in stride data prefetchers. Our study reveals fault scenarios in the prefetcher table that can degrade IPC more than 17%, while faults in the prefetch input and request queues can slow IPC up to 24% and 26%, respectively, compared to a fault-free component. Finally, a faulty data prefetcher can increase performance variability across identical cores because the standard deviation of IPC loss for different benchmarks can be more than 4.5%.

I. INTRODUCTION

High-performance multi-core microprocessor architectures [12] [13] [14] dominate different application domains. However, the inherent unreliability of deep nanometer-scale technologies [5] [6] [10] [18] and near-threshold voltage operation [1] [8] increase the vulnerability of on-chip memory array cells to hard (permanent) faults. If unaddressed, these faults will impose significant constraints on microprocessor design.

To hide the performance impact of high-latency memory accesses, computer architects integrate multi-layer cache memories along with sophisticated data prefetchers [17] [19]; both structures can occupy noticeable silicon estate. Data prefetch designs predict the flow of data and correspondingly increase CPU utilization by reducing the stalls due to cache misses. One widely used class of data prefetch mechanisms, the stride data prefetchers, have been shown to be highly effective for scientific, multi-media, desktop, and engineering applications [16].

Technology modeling in resilience roadmaps predicts that the failure probability of SRAM cells is expected to be extremely high (10^5 and 10^3 times larger than that of latches for the 16- and 12-nm nodes, respectively) [26]. Most reliability studies have focused on SRAM caches due to the area they occupy and their immediate impact on both functional correctness and performance [2] [4] [20] [23] [24] [25]. Previous work focused on non-cache SRAM arrays such as those in data flow speculative hardware [3] [9]. However, no previous work has performed a comprehensive and accurate assessment of the impact of hard faults on data prefetchers, despite these structures being critical for maintaining processor performance.

Unlike cache memories, data prefetchers do not affect program correctness because they are purely speculative in nature. 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). Recent work [9] 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, data prefetcher arrays can experience faults, which will trigger imbalances in the data stream sent to memory system, leading to inter-core performance variability. This is an undesirable property both for the data-center deployment [1] and for the mobile and desktop markets [15].

Figure 1 and Figure 2 visualize the motivation of this paper. Figure 1 shows the IPC of fault-free and faulty microprocessor models for GemsFDTD and bzip2 benchmarks. In Figure 1, the blue line presents the IPC with the data prefetcher disabled, the red line shows the IPC with the data prefetcher enabled, and the green line points show the maximum IPC loss among all faults with the same group cardinality. (Details for all SPEC CPU2006 benchmarks are provided in Section IV.) Clearly, faulty data prefetchers can severely affect performance: both GemsFDTD and bzip2 lost more than 17% of their performance in the worst case.

Figure 2 presents 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. The IPC difference between the worst and best cases is up to 17 percentage points, while standard deviation ranges from 1.9% to 4.5%. Thus, a faulty data prefetcher can increase significantly the variability across otherwise identical cores.

In this paper, we contribute the following:

- We measure, for the first time, the performance impact of multiple permanent faults on an L1 stride data prefetcher on an architectural simulator using the complete SPEC CPU2006 benchmark suite.
- We evaluate the degree of performance variability among cores caused by faulty stride data prefetchers.

Our results show that the performance impact 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 queue, respectively). Meanwhile, the performance variability can be more than 26 percentage points compared to the fault-free case (standard deviation of IPC loss between benchmarks ranges from 0.01% to 4.5%).

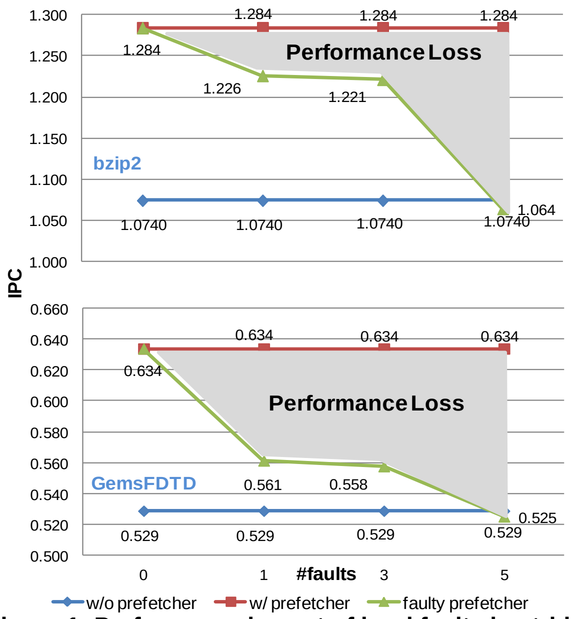

Figure 1: Performance impact of hard faults in stride data prefetchers (blue line is fault-free model with no data prefetching; red line is fault-free model with data prefetching; green line is faulty prefetcher model with one, three, and five hard faults).

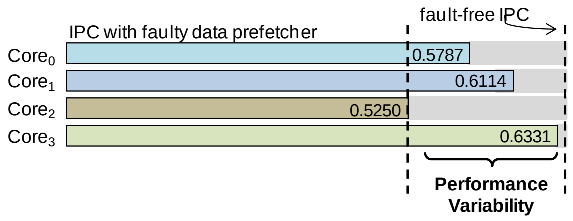

Figure 2: Performance variability in a 4-core chip for GemsFDTD with five hard faults in the data prefetcher of each core.

II. Background

Resilience roadmaps predict a large numbers of hard faults in SRAM arrays that operate in near-threshold voltages [1] [8] [28], as well as in forthcoming chips (16- and 12-nm processes) [26]. In both contexts, the single-bit failure probability (P_{fail}) of SRAM cells is expected to fall between 10^{-6} and 10^{-4} [8] [26] [28]. Given a binomial probability distribution, such failure rates would result in very high probabilities of multiple hard faults in SRAM arrays. For example, in a 5,000-bit SRAM array (a typical array size for a stride data prefetcher) at the 12-nm process node, the cumulative probability that up to five hard faults exist in the array is 7.25E-01. Circuit-level techniques such as wordline boosting [22] can be employed to reduce these probabilities; however, such techniques add complexity and area to the array design.

III. Experimental Set-up

We perform a comprehensive statistical fault-injection campaign on top of the PTLsim x86 architectural simulator [29]. We employ the microprocessor configuration of [9] enhanced with the modifications shown in Table 1. We use a statistical fault-injection framework [21] [22] (with a confidence level of 99% and an error margin of 3%) 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 [7] [11] [25] 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 injections for the 29 SPEC CPU2006 benchmarks are 26,100).

Parameter	***Setting***
Prefetch input queue	8 entries
Prefetch table	64 entries, direct-mapped, PC-indexed
Confidence size	3 bits (threshold: 2)
Stride size	5 bits
Prefetch distance	Address + Stride, Address +2*Stride
Prefetch request queue	8 entries

Table 1: L1 cache-stride data prefetcher configuration.

Each fault-injection run applies randomly 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 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 5,000-bit SRAM array, which is the size of the prefetcher array of this study). To examine performance variability, we calculate the maximum IPC loss and standard deviation of IPC losses per fault group size across all benchmarks.

IV. Experiments

A. *Prefetch-friendly and -unfriendly Benchmarks*

We profile the full set of SPEC CPU2006 benchmarks to measure the IPC impact of a fault-free L1 cache-stride data prefetcher. Table 2 presents the IPC speed-up for each benchmark due to the stride data prefetcher. On average, the data prefetcher boosts IPC by 6.85%. However, performance improvement varies and depends on the stream of memory access patterns generated by each benchmark. For that reason, we classify benchmarks into two major categories: prefetch-friendly, in which the IPC change is greater than the average speed-up across all SPEC CPU2006 benchmarks, and prefetch-unfriendly, in which the change is less than the average speed-up. Eleven benchmarks are classified as prefetch-friendly and 18 are classified as prefetch-unfriendly. The impact of faults on performance significantly differs between the two groups of benchmarks.

Prefetch-friendly benchmarks	***IPC (%) Speed-up***	***Prefetch-unfriendly benchmarks***	***IPC (%) Speed-up***
bzip2	19.99	perlbench	2.58
bwaves	10.14	gcc	1.89
gamess	21.51	mcf	0.16
zeusmp	9.10	milc	–4.11*
leslie3d	7.55	gromacs	2.86
dealII	9.61	cactusADM	0.49
soplex	9.50	namd	0.34
GemsFDTD	19.66	gobmk	0.60
libquantum	17.20	povray	0.42
tonto	15.91	calculix	4.73
wrf	34.59	hmmer	0.62
average	**15.887**	sjeng	0.07
		h2564ref	0.96
		lbm	4.66
		omnetpp	0.07
		astar	3.01
		sphinx3	0.67
		xalancbmk	3.80
		average	**1.780**
Overall average (%)		**6.85**	

Table 2: Per-benchmark IPC speed-up provided by the L1 data prefetcher (*milc's IPC is slowed by 4.11%).

B. *Performance Impact of Faults*

In this section, we measure the performance impact of hard faults injected into the data prefetcher. Figure 3 shows the average and maximum IPC slow-down (due to faults) when one, three, and five faults are injected in the prefetch table along with the standard deviation; the upper diagram shows prefetch-friendly benchmarks and the lower diagram shows prefetch-unfriendly benchmarks. Figure 3 presents the average performance loss across all benchmarks (i.e., the prefetch-friendly benchmarks show a combined 3.049% IPC loss if we average the maximum IPC loss over all single fault runs per benchmark, 5.759% IPC loss over all triple faults, and 9.271% IPC loss over all quintuple faults). Thus, an L1 cache-stride data prefetcher can severely degrade microprocessor performance, up to 9.271% on average for the prefetch-friendly and up to 0.733% on average for the prefetch-unfriendly benchmarks, when the prefetcher table's SRAM cells suffer multiple hard faults.

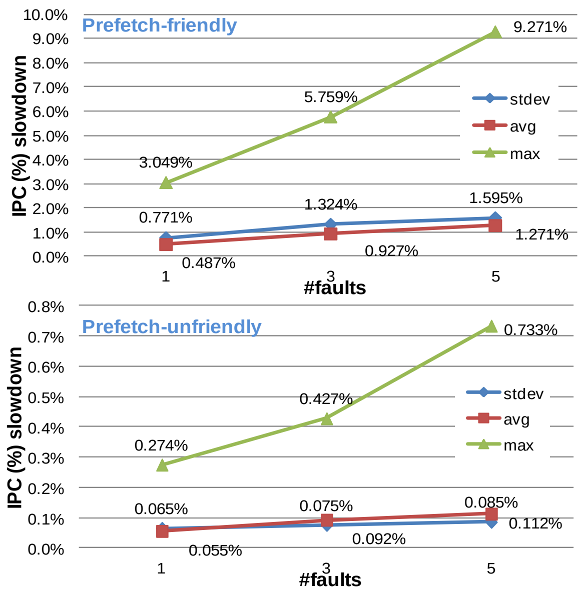

Figure 3: **IPC loss for the prefetch-friendly (upper graph) and -unfriendly (lower)**.

Figure 4 shows the average (upper) and maximum (lower) normalized IPC slow-down for each SPEC CPU2006 benchmark when one, three, and five faults are injected. The benchmarks are grouped into prefetch-friendly (left half) and -unfriendly (right half), to identify any correlation between workloads and performance loss. The colors on each column depict the additional IPC loss relative to the fault-free model from the injection of one, three, and five faults. For example, on bzip2, the maximum IPC loss is 4.5% for a single injected fault, 4.8% for triple faults, and 17.1% for quintuple faults (i.e., the aggregation of single, triple, and quintuple IPC losses). As expected, the prefetch-friendly benchmarks show a greater IPC impact with the same number of faults compared to the prefetch-unfriendly. In particular, a fault-free prefetcher improves execution time of GemsFDTD by 20% and sphinx3 by 0.6% (Table 2). GemsFDTD suffers a maximum 17% IPC slow-down, while sphinx3 loses only 0.06% when quintuple faults are injected.

By further analyzing the internal behavior of the prefetcher, we found that the extent of the performance impact depends on the distribution of the training input addresses across the prefetch table entries. For example, Figure 5 presents the activity of each prefetcher table entry for two benchmarks, bzip2 and gcc, and shows very different sensitivities to data prefetching (17% and 0.06%, respectively). gcc shows a much more uniform usage of the entries of the table, while bzip2 trains only seven entries (95% of training occurs on only three entries and the remaining four are trained only marginally). Thus, in gcc, the majority of the training addresses remain unaffected by the injected faults; even if they do access a faulty entry, the IPC impact is relatively small because of the lower average dynamic usage frequency. In bzip2, if the fault occurs in one of the heavily used entries, the majority of training is affected, and so the IPC loss due to faults is much greater.

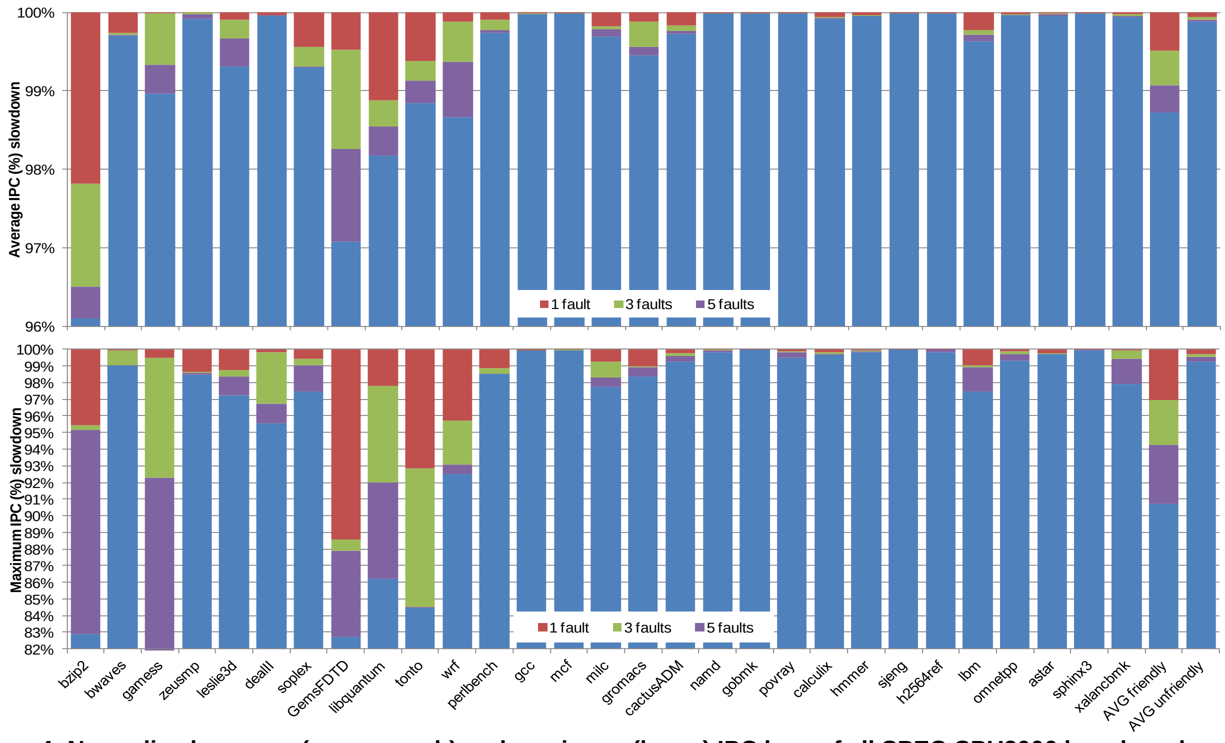

Figure 4: Normalized average (upper graph) and maximum (lower) IPC loss of all SPEC CPU2006 benchmarks when one, three, and five faults are injected into the prefetcher table. The 100% point of the vertical axis is the fault-free IPC; we show the IPC loss due to single faults (red bar), triple faults (green bar), and quintuple faults (purple bar). The 11 left-most bars show the prefetch-friendly benchmarks, the next 18 show the prefetch-unfriendly benchmarks, and two right-most bars of each diagram show the averages for the two categories.

To understand the results of Figure 4 better, we collected additional data, shown in in Figure 6, that shows the issued prefetch requests rate (per 1,000 committed instructions) and the average L1 cache miss rate (misses per 1,000 committed instructions, or MPKI) for the fault-free and faulty cases for each group of injected faults.

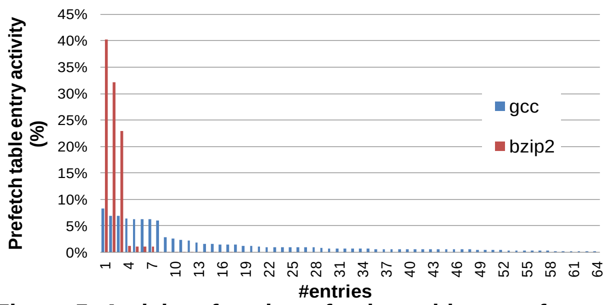

Figure 5: Activity of each prefetcher table entry for gcc and bzip2.

According to Figure 6, the faulty prefetcher is throttled because the faults block a 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 increases from 26 to 27 MPKI in the quintuple injected fault scenario). The data in Figure 6 illustrate the greater performance sensitivity of the prefetch-friendly benchmarks to faults. Also, faults in the prefetch table sometimes change the prefetch addresses sent to memory, which affects the L1 cache miss rate and IPC.

Due to the small size of PIQ and PRQ (8 entries in total), we injected only single faults. This was sufficient to demonstrate the severe impact on performance that hard faults have on these queues. Figure 7 shows the maximum and average IPC slow-downs and the standard deviation for single faults injected into the PIQ and PRQ per benchmark.

Across all benchmarks, the average IPC loss (1.5% and 2.5% for PIQ and PRQ, respectively) and maximum IPC loss (24.3% and 26.3% for PIQ and PRQ, respectively) are significantly higher than that of the prefetch table because the training addresses (buffered in PIQ) and the prefetch requests (queued in PRQ) are more likely to be polluted by a fault due to their small size. The fault location determines the extent of the performance impact. More specifically, the PIQ entries are utilized uniformly across all benchmarks (utilization ranges from 35% to 7%, moving from the top to the bottom entry). By contrast, the top three entries in PRQ handle 95% of activity across all benchmarks. Therefore, faults that reside in the bottom entries of the queue have minimal impact on performance.

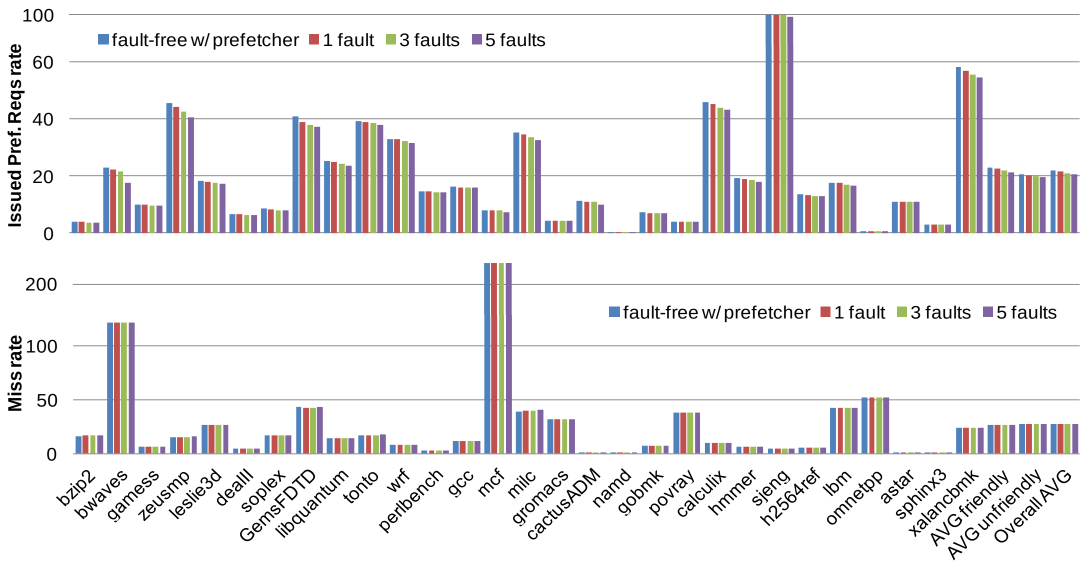

Figure 6: Average L1 cache miss rate and average issued prefetch requests (per 1,000 committed instructions) for SPEC CPU2006 benchmarks with one, three, and five faults injected.

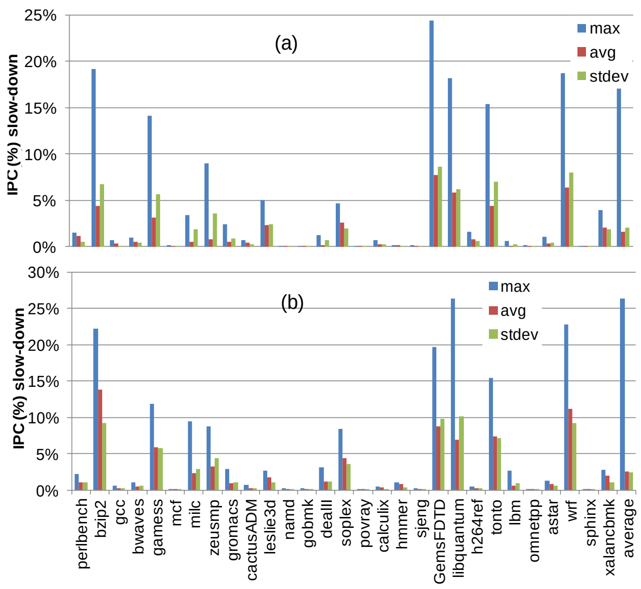

Figure 7: Average and maximum IPC slow-downs and standard deviations for the fault-free and faulty (a) PIQ and (b) PRQ when single faults are injected in the SPEC CPU2006 suite.

The impact of faults in the prefetcher can affect microprocessor performance even more when data sharing occurs in multi-threaded applications. A faulty prefetcher can prefetch shared data erroneously into the cache, causing additional remote references from other threads, or not prefetch shared data into the cache, causing additional remote references from the current thread.

Based on this analysis, we conclude that microprocessor performance can be degraded severely by a faulty L1 cache-stride data prefetcher. The impact can be more than 24% IPC slow-down due to PIQ faults, up to 26% IPC slow-down due to PRQ faults, and up to 18% IPC slow-down due to faults in the prefetch table due to prefetch throttling or cache pollution. Furthermore, the benchmarks with highly repeatable address patterns are the most susceptible to IPC loss due to faults.

C. Performance Variability

In this section, we discuss performance variability across cores in the presence of multiple faults in the prefetcher. Performance variability can affect the cost and the power budget of a data center. Our findings are the following:

- The maximum IPC slow-down for even single faults can be up to 7% for 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. Therefore, there is a large variation in IPC under the presence of different single faults in the data prefetcher. This finding holds when all cores are affected by the same number of faults.
- The difference in IPC slow-down for different numbers of faults across the cores ranges between 0.005% and 17% compared to the fault-free IPC, on average and 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 performances 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. The standard deviation of IPC loss for the prefetch-unfriendly benchmarks is 0.06%, 0.07%, and 0.08%. 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 the inter-core performance variability.

V. Related Work

Previous work on array faults focused on caches. Abella et al. [1] proposed disabling and re-mapping to guarantee predictable performance at low voltages. Agarwal et al. [2] focused on yield improvements tolerating process variations. Ansari et al. [4] proposed Zerehcache architecture to deal with massively defective caches. Chishti et al. [8] employed special types of error-correcting codes to improve lifetime reliability. Performance implications with disabled cache parts were discussed in [1] [4] [20] [23]. Roberts et al. [24] proposed cache-line merging techniques, and Wilkerson et al. [28] also employed cache-line combining and disabling to survive voltage scaling. Hardy et al. [11] presented an analytical model that covers microprocessor arrays, while previous related effort on caches only was presented by Sanchez et al. [27]. Almukhaizim et al. [3] presented an approach for deterministic, software-based testing of a multi-entry stream buffer. Foutris et al. [9] measured the impact of single hard faults using a conservative data prefetcher configuration. Our work differs significantly from [9] because we measure the impact of multiple faults in the prefetcher and its supporting logic (PIQ and PRQ).

VI. Conclusions

The existence of hard faults in a stride data prefetcher can affect microprocessor performance significantly and increase inter-core performance variability. Our detailed experimental analysis demonstrates that IPC loss due to hard faults in the prefetch table can be up to 17%, and up to 24% and 26% for the prefetch input queue and prefetch request queue, respectively. Also, performance variability across cores is increased: the standard deviation of IPC loss between benchmarks can be more than 4.5%. Thus, prefetcher structures should be protected in both current and forthcoming microprocessor technologies.

Acknowledgment

This work has been supported by an AMD Research grant to the Computer Architecture Lab of the University of Athens, by Thales-UoA HOLISTIC Project and the 7th Framework Program of the European Union through the CLERECO Project, under Grant Agreement 611404.

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] S. Almukhaizim, P. Petrov, A. Orailoglu, “Faults in processor control subsystems: testing correctness and performance faults in data prefetching unit,” ATS 2001.

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

[5] 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),” Technical Report 03024377A-TR, Int’l SEMATECH, March 2003 [Online] http://www.itrs.net/Links/2003ITRS/LinkedFiles/PIDS/4377atr.pdf.

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

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

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

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

[10] 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.

[11] D. Hardy, I. Sideris, N. Ladas, Y. Sazeides, “The performance vulnerability of architectural and non-architectural arrays to performance faults,” MICRO, 2012.

[12] http://newsroom.intel.com/community/intel_newsroom/blog/2013/06/17/intel-powers-the-worlds-fastest-supercomputer-reveals-new-and-future-high-performance-computing-technologies.

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

[14] http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf.

[15] 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.

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

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

[18] JEDEC Solid State Technology Association. “Failure Mechanisms and Models for Semiconductor Devices.” JEDEC Publication JEP122-G, October 2011 [Online] http://www.jedec.org/standards-documents/docs/jep-122e.

[19] 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.

[20] H. Lee, S. Cho, B.R. Childers, “Performance of Graceful Degradation for Cache Faults,” IEEE Comp. Society Symp. on VLSI, 2007.

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

[22] 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.

[23] 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.

[24] 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.

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

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

[27] 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.

[28] 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.

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