Voltage Noise Mitigation With Barrier Approximation

Zamshed I. Chowdhury, S. Karen Khatamifard, Zhaoyong Zheng, Tali Moreshet, R. Iris Bahar, and Ulya R. Karpuzcu

Abstract—Barrier synchronization constructs are placed between phases of parallel programs to ensure correctness in the execution – by preventing threads from proceeding to the subsequent phases of the program before all threads have completed the preceding stage(s). Upon release, threads leaving the barrier at the same time cause sudden change in activity that can potentially lead to voltage emergencies in the form of timing errors, due to electrical properties of power delivery network. In this paper, we demonstrate how approximation through barrier relaxation – i.e., letting threads proceed past barriers without waiting for the others, and thereby preventing abrupt activity changes – can help prevent voltage emergencies.

Index Terms—Approximate barrier synchronization, voltage noise, timing errors, relaxed synchronization

1 Introduction

Power delivery network (PDN) directly tracks the activity of logic and memory blocks on chip through temporal changes in their current demand. In a system with fixed power budget, time dependent variation of workload during execution results in spatio-temporal changes in supply voltage, V. Voltage noise broadly refers to any deviation of V from the nominal supply voltage V_{NOM}, which constitutes the ideal operating voltage. The deviation can take the form of an overshoot over V_{NOM}, which challenges the physical reliability of the PDN; or an undershoot (droop) which increases the latency of basic logic operations.

Droops are particularly problematic: A significantly lower V can make operation at nominal frequency impossible, resulting in timing errors. Modern design practice adds a voltage (or frequency) guardband to mask such adverse impact. The more conservative the guardband to rule out droops is, the higher (alternatively, the lower) the operating voltage (frequency) would need to be, inevitably decreasing energy efficiency. The parasitic resistance R of the PDN causes an undershoot, proportional to $I \times R$, which does not depend on the rate of change in activity – where I (current demand) represents a proxy for microarchitectural activity. The parasitic inductance L of the PDN, as well, causes undershoots, proportional to $L \times dI/dt$, which directly depends on how abruptly the activity (i.e., I) changes over time, i.e., dI/dt. dI/dt based droops are harder to mask via guardbanding, since predicting potential activity changes during runtime is more challenging. Especially concerning are activity changes matching the resonance frequency of the PDN, where PDN parasitics, and hence voltage droops, reach their peak values.

- Zamshed I. Chowdhury, S. Karen Khatamifard, and Ulya R. Karpuzcu are with the University of Minnesota, Minneapolis, MN 55455 USA. E-mail: {chowh005, khatami, ukarpuzc}@umn.edu.
- Zhaoyong Zheng and R. Iris Bahar are with the Brown University, Providence, RI 02912 USA. E-mail: {zhaoyong_zheng, iris_bahar}@brown.edu.
- Tali Moreshet is with the Boston University, Boston, MA 02215 USA. E-mail: talim@bu.edu.

Manuscript received 2 Oct. 2020; revised 29 Oct. 2020; accepted 16 Nov. 2020. Date of publication 24 Nov. 2020; date of current version 11 Dec. 2020.
(Corresponding author: Zamshed I. Chowdhury.)
Recommended for acceptance by C. Wu.
Digital Object Identifier no. 10.1109/LCA.2020.3040088

Barrier synchronization ensures correctness of parallel programs by preventing threads from proceeding to the subsequent phases of the program before all threads have completed the preceding stage(s). Upon release, threads leaving the barrier at the same time, by construction, can cause sudden change in activity (i.e., large dI/dt) that can potentially lead to voltage emergencies in the form of timing errors. In this paper we explore how approximation through barrier relaxation can help prevent voltage emergencies. The idea is letting threads proceed past barriers without waiting for the others, and thereby preventing abrupt changes in activity, i.e., in the current drawn by logic and memory blocks. However, relaxing barriers by definition relaxes the correctness guarantee, and hence is only applicable if the impact on the end result of computation is limited by an acceptable loss in accuracy.

The runtime voltage signature of an application depends on both, the number of (dynamic) barriers and the time between successive barrier invocations. Increasing thread counts make the situation worse by causing a higher dI/dt. Scientific applications, which typically incorporate many threads along with many frequent barriers, are especially vulnerable to dI/dt induced emergencies. Luckily, many of these applications are approximate in nature and feature algorithmic noise tolerance. Barrier relaxation by removing static and/or dynamic barrier instances can particularly help in this case – by judicious selection of barrier instances that satisfy a predetermined accuracy constraint in execution outcome.

The main contributions of our study are twofold:

1) To the best of our knowledge, we are the first to propose and demonstrate that barrier relaxation can improve the voltage noise profile in barrier-heavy parallel applications.
2) We demonstrate, through case studies, how effective compile- and run-time implementations of barrier relaxation can help mitigate voltage noise without compromising application output accuracy or quality.

The rest of the paper is organized as follows: Section 2 covers the related work; Sections 3 and 4, our quantitative analysis; and Section 5, a discussion and summary of our findings.

2 Related Work

Previous studies identified synchronization as a major factor to induce voltage noise [1], [2]. The stressmark generation proposals focusing on capturing and characterizing resonance [1], [3], [4], [5] emphasize the significance of synchronization and alignment of microarchitectural events. EmerGPU [6] further analyzes resonance effects in GPUs due to lock-step (SIMD) execution. Orchestrator [7] demonstrates how activity-guided thread scheduling can help mitigate voltage noise. However, none of these studies, including the rich body of work on voltage noise mitigation at the architecture level (e.g., [8]) and utilizing different barrier techniques (e.g., [2], [9]), focus on barrier approximation to reduce voltage noise. Various proposals for relaxed synchronization, including barriers, also exist [10], [11], [12], [13]. The idea is to selectively omit synchronization to improve the performance and/or energy efficiency, as long as the potential loss in computation accuracy remains acceptable. Such studies typically characterize the trade-off space of accuracy versus energy efficiency or performance. To the best of our knowledge, ours is the first paper to analyze the impact of approximate barrier synchronization on voltage noise.

3 Experimental Setup

Simulation Framework: For voltage noise experiments, we deploy Sniper [14] integrated with a revised version of McPAT [15] to collect runtime power traces which capture microarchitectural activity

TABLE 1
Benchmark Applications

Application	Input	Accuracy Metric
INTEGER SORT (IS) [19]	class W	positional error
RADIX [20]	n=4194304, r=4096, m=524288	positional error
OCEAN [20]	n=514, e=1e-07, r=20000, t=28800	diff. between residual
BARNES [20]	16384, 123, NULL, 0.025, 0.05, 1.0, 2.0, 5.0, 0.075, 0.25, p	distance in body positions
WATER-NS [20]	1.5e-16, 512, 3, 6, -1, 3000, 3, 0, p, 6.212752	distance in body positions
STREAMCLUSTER (SC) [18]	10, 20, 32, 4096, 4096, 1000, none, output.txt, p	number of mismatches
BODYTRACK (BT) [18]	sequenceB_1 (simsmall)	diff. between position vectors

TABLE 2
Summary of Barrier Relaxation Experiments

Application	% Removed	% Accuracy loss	% Execution time reduction	% Energy reduction
INTEGER SORT (IS)	69.4	0.04	6.03	3.14
RADIX	100	0.53	1.87	1.04
OCEAN	22.22	0.022	10.5	4.84
BARNES	52.9	33.2	3.07	1.52
WATER-NS	75	8.02	0.3	0.12
STREAMCLUSTER (SC)	33.05	16.7	19.32	1.19
BODYTRACK (BT)	28.6	0.001	0.07	0.004

as a function of time. The simulated system closely mimics POWER7 [16] in accordance with the revised power model [15], where V_{NOM}=1.01V. We experiment with a core (thread) count of 8 and 32. Voltage noise analysis comes from Voltspot, a pre-RTL PDN model [17]. For accuracy experiments, Intel Xeon CPU based system is used. In all experiments, we pin the threads to individual cores in order to minimize simulation noise (and prevent potential serialization among threads during execution by OS intervention).

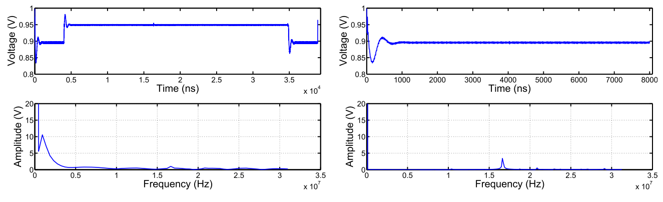
Benchmarks: We use a voltage noise virus along with 7 barrier-rich applications from 3 different benchmark suites (Table 1). *Integer Sort (IS)* and *RADIX* are (integer) sorting applications. *OCEAN* evaluates the large scale movement in oceans through eddy and boundary currents simulation. Two of the benchmark applications are n-body simulators, which capture the interactions in a system of bodies, i.e., particles for *BARNES*; and water molecules, for *WATER-NSQUARED*. *STREAMCLUSTER* (SC) is an RMS [18] workload that solves an online clustering problem given a set of input points. Another RMS application, *BODYTRACK (BT)*, tracks a human body through a sequence of images with particle annealing.

Relaxed Barrier Synchronization: We first profile the benchmark applications by removing one barrier at a time. We exclude implicit barrier constructs of the synchronization API from relaxation. We repeat each experiment at least 100 times to ensure statistical significance. We label any barrier as critical if it causes catastrophic termination or convergence problems when relaxed. Such profiling is performed once and used for both static and dynamic relaxation techniques. We only consider the non-critical subset of the barriers for approximation and report the outcome under combined relaxation of such non-critical barriers. Table 2 lists the % static barrier instances removed from the applications. We use application-specific metrics to quantify the associated accuracy loss, i.e., whether program outputs are deemed valid upon relaxation (Table 1). To determine validity, we use the applications' implicit error check and acceptance tests.

4 Barrier Relaxation

4.1 Impact on Voltage Noise

Opportunistically letting threads proceed without waiting at the barrier (for the arrival of the slowest thread) can reduce performance overhead as well as any activity differential (hence, voltage noise) due to barrier execution, in exchange of reduced computational accuracy.



(a) Before Relaxation (b) After Relaxation

Fig. 1. V-signature under barrier relaxation (8 threads).

As a representative example, using 8 threads, we show the voltage signatures of IS without (Fig. 1a) and with (Fig. 1b) removal of the first barrier (in order of appearance) in the source code.[1] The accuracy loss in this case is negligible (Table 2). Furthermore, the non-relaxed execution takes longer than the relaxed version and exhibits a sharp droop of 86.3 mV, which is non-existent in the relaxed case.[2] Figs. 2a and 2b depict the corresponding traces for 32 threads. Our observations for 8 threads hold and get even more pronounced in this case.

Note that for the non-relaxed executions shown in Figs. 1 and 2, in the frequency spectrum we notice peaks at around 1 MHz, 20 MHz, and 25 MHz, which indicates the intensity of the voltage noise at particular frequencies. The only peaks common to both non-relaxed and relaxed cases are at around 16.62 MHz, which corresponds to the voltage trace outside the barrier region. *The key observation is that, under barrier relaxation, high frequency components of voltage noise subside. As the frequency domain analysis reveals, the relaxed barrier trace does not feature any dominating high frequency components, which indicates a lower rate of change in activity during execution.*

4.2 Static Relaxation

Subject to the accuracy loss incurred, we can relax barriers *statically* (at compile time) or *dynamically* (at runtime). Static relaxation refers to removal of barrier instances from the code, which implies removal of all respective dynamic instances during runtime. Practically, we can only relax a barrier statically if we can guarantee no or acceptable accuracy loss as a result. This inevitably requires profiling analysis to capture correctness and accuracy implications under different execution scenarios. Fig. 3 illustrates, on per

1. The y-axis captures the effective operating voltage, i.e., V_{NOM} - Voltage droop, which reduces as the core activity increases (due to a higher droop). At 0.5 $(\times 10^4)$ ns mark in Fig. 1a, threads enter the barrier. At 3.5$(\times 10^4)$ ns, threads leave the barrier simultaneously which is demarcated by a voltage droop event due to sudden change in thread activity. Conservatively, we assume that the threads are in a low power state when inside the barrier. As indicated in Fig. 1b, with removal of this barrier, the corresponding region between 0.5$(\times 10^4)$ ns and 3.5 $(\times 10^4)$ ns in Fig. 1a disappears, flattening the signature and removing the corresponding voltage droop event in the process.

2. There is another droop at the beginning of the execution that persists in Figs. 1a and 1b, which is due to thread spawning and initialization events, where relaxing barriers has no impact.

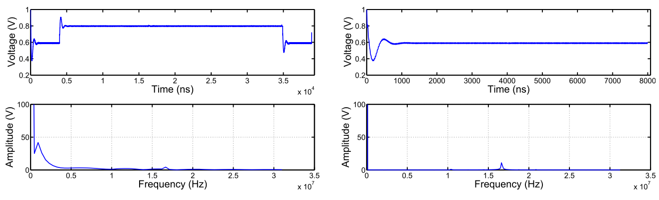

(a) Before relaxation (b) After relaxation

Fig. 2. V-signature under barrier relaxation (32 threads).

TABLE 3
Input Datasets Used for Sensitivity Analysis

Application	Parameter	Input-1	Input-2	Input-3
IS	Keys	2^{20}	2^{23}	2^{25}
RADIX	Keys	4M	16M	64M
OCEAN	Grid dimension	514	1,026	2,050
BARNES	Particles	16,384	32,768	65,535
WATER-NS	Particles	512	3,375	8,000
SC	Particles	4,096	8,192	16,384
BT	Particles	1,000	2,000	4,000

application basis, the maximum accuracy loss observed under static relaxation, when we remove one barrier at a time. As explained in Section 3, we exclude any barrier relaxation which leads to catastrophic execution (such as no termination due to deadlock or livelock conditions) from this analysis. In these experiments, we aggressively relax the non-critical barriers as long as the program can still generate a valid output with low accuracy loss, to extract the pareto-frontier of the voltage noise versus accuracy trade-off space. Note otherwise that, depending on the application, even a less than 1 percent accuracy loss may not be acceptable.

For IS, RADIX, OCEAN, WATER-NS, SC and BT, a high number of static barrier relaxations, 83.33, 85.7, 66.67, 55.5, 60 and 100 percent, respectively, exhibit practically negligible accuracy loss. BARNES represents an exception, where all of the static barrier relaxations result in high accuracy loss with a mean of 35.8 percent. Table 2 summarizes the execution outcome under combined static relaxation, where we statically relax a subset of non-critical barriers in an application at the same time. *% removed* captures the relative number of (runtime) barrier invocations removed from the execution as a result, and hence serves as a proxy for the relative number of the (barrier-induced) droop events eliminated. For the majority of the applications, more than 30 percent relaxation is possible (without catastrophic termination). The accuracy loss varies from less than 1 percent (IS) to less than 17 percent–except for BARNES (33.2 percent), where the high accuracy loss is due to a high number of write accesses to shared data [21].

Relaxed barrier synchronization is a stochastic process and can only provide statistical guarantees when it comes to correctness of execution or acceptability of the accuracy loss. Combined relaxation of non-critical barriers can always result in higher accuracy loss when compared to individual relaxations, even if individual relaxations by themselves render no accuracy loss for practical purposes. For instance, the maximum accuracy loss in IS increases by 4X when four non-critical barriers are relaxed at the same time, when compared to the maximum under individual barrier relaxation (Fig. 3). A similar trend applies to the rest of the applications. *The key observation here is that neither accuracy loss nor voltage noise under relaxed barrier synchronization is cumulative as more barriers get relaxed*. We should also note that this profiling based analysis only provides us with a list of barriers that can be relaxed. This list is not exclusive by any means, and due to the stochastic nature of the process, inclusion of non-critical, yet very high accuracy loss barriers might still result in relatively lower loss of accuracy in combined relaxations.

When it comes to secondary effects (beyond voltage noise), we observe a big variation in the percentage of execution time reduction, which can get as high as almost 20 percent (SC). Applications such as RADIX, WATER-NS and BT, where the time spent in barriers is already relatively low, experience practically no reduction. The percent reduction in energy consumption is not significant across the board, as we conservatively let threads go to a low-power state during their wait time in the barrier when not relaxed. Note that such low savings in energy is with respect to barriers where threads go to deep sleep (inside the barrier), at the expense of higher voltage noise. This result hints at a better energy versus voltage noise trade-off under relaxed barrier synchronization.

Execution outcome under relaxed synchronization is highly dependent on the size and value distribution of input data, as well. To capture this effect, we repeat the accuracy experiments using two more input data-sets of varying size. Specifically, to guarantee balanced scaling of the problem size, we use the input data-sets suggested by the benchmark suites: simsmall (default), simmedium and simlarge for SPLASH2 [22] and PARSEC [18]; and classes W (default), A and B for NAS [23]. Table 3 lists the key input parameters for each case in increasing order of the problem size.

Fig. 4 illustrates the sensitivity of static relaxation to the problem size. Overall we observe that, in the worst case, as the input size increases, so does the loss in accuracy, but only slightly. For most cases (except OCEAN), the loss in accuracy reduces with increasing problem size. For IS, the accuracy loss remains less than 1 percent across the board. For SC, the mean accuracy loss (across 100 experiments) is 16.75, 9.29, and 9.79 percent for simsmall (the default input data-set), simmedium, and simlarge, respectively, with a standard deviation of 3.56, 2.90, and 1.82 percent. For BARNES, the difference in accuracy loss for different inputs is negligible, but the accuracy loss itself is very high. At the other end of the spectrum lies OCEAN, which incurs an increasing loss in accuracy with increasing problem size, although very low in amplitude ($\leq 1\%$) for all inputs. This analysis further highlights the stochastic nature of relaxed synchronization, since there is no definite monotonic trend in the accuracy loss for a given application or across applications, as observed in previous work [24].

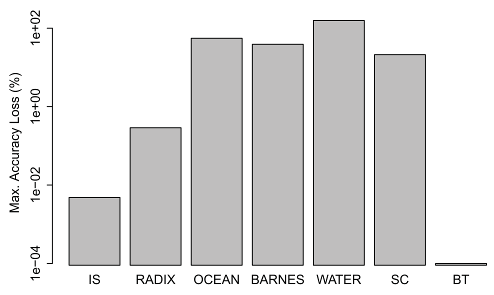


Fig. 3. Maximum accuracy loss under static relaxation.

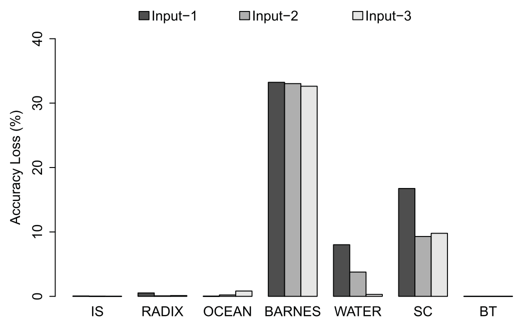


Fig. 4. Sensitivity of accuracy loss to input data.

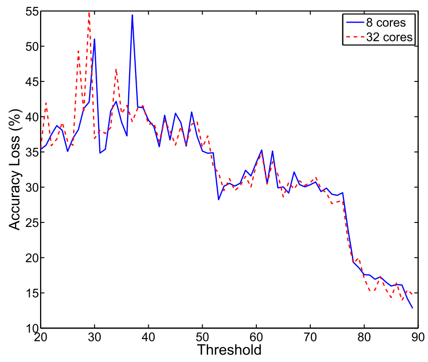


Fig. 5. Accuracy loss under dynamic relaxation (WATER-NS).

4.3 Dynamic Relaxation

We next look into the potential of dynamic (runtime) barrier relaxation, which may achieve a better trade-off than conservative static relaxation by exploiting runtime information on the fly. In the following, we will report the outcome under random relaxation, without loss of generality, to eliminate any potential bias due to a particular implementation. Specifically, each thread reaching a barrier generates a random number based on which it waits at (or leaves) the barrier if the random number is less (greater) than a predefined threshold. Thereby, we can relax a subset of the dynamic instances of a given barrier (as opposed to all under static relaxation). For dynamic relaxation experiments we consider all non-critical barriers (as opposed to a subset of non-critical barriers that don't incur excessive accuracy loss, in case of static relaxation), regardless of the individual impact on the overall accuracy. This is particularly useful for cases where not all dynamic instances of a barrier are crucial for high accuracy, but only a subset. Capturing such cases can improve not only the accuracy under relaxation, but also the voltage noise profile of the execution by relaxing more dynamic barrier instances. As the threshold decreases, it is more likely that a thread goes through a barrier without waiting for other threads to arrive at that barrier, which tends to lead to a higher accuracy loss, but lower voltage noise due to the lower number of threads released from the barrier. On the other hand, if the threshold increases, more threads would wait at the barrier which tends to cause a lower accuracy loss, but higher voltage noise due to the higher number of threads released from the barrier simultaneously.

As a representative example, Fig. 5 shows the dynamic change in the accuracy loss of *WATER-NS* for different values of the threshold (as captured by the x-axis). We experiment with 8 and 32 cores, and use the default input data-set from Table 1. We sweep the threshold between 20–90 and let each thread generate a random number between 0–100 at each barrier entry.[3] We observe a relatively high accuracy loss for both core counts, when threads are less likely to wait at the barrier (corresponding to lower values of the threshold). When the threshold increases, more threads are likely to wait at the barrier, and the accuracy improves accordingly. The total execution time is not significantly sensitive to the threshold as the slowest thread dictates the execution time, rather than the percentage of dynamic barrier instances relaxed. Therefore, the runtime overhead is insensitive to the % of threads released from the barrier simultaneously – except for when no thread waits at the barrier similar to static relaxation. While dynamic relaxation unlocks a much wider range of accuracy loss in the trade-off space as an opportunity for more aggressive optimization, it also challenges practical dynamic control due to its highly stochastic nature.

3. A threshold of 0 corresponds to (a more aggressive variant of) static relaxation (where all non-critical barriers get relaxed), where no thread waits at the barrier; of 100, to fully synchronized execution, where no barrier gets relaxed.

5 Conclusion

Barrier synchronization constructs are required to ensure correctness of parallel execution, at the expense of abrupt change in activity at the point of thread release. We demonstrate the potential of barrier relaxation, i.e., how opportunistically letting threads proceed without waiting at the barrier can reduce voltage noise in exchange of accuracy loss.

Acknowledgments

This work was supported in part by NSF Grant no. CCF-1438286.

References

[1] V. J. Reddi *et al.*, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *Proc. 43rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2010, pp. 77–88.
[2] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in *Proc. 39th Annu. Int. Symp. Comput. Architecture*, 2012, pp. 249–260.
[3] R. Bertran *et al.*, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2014, pp. 368–380.
[4] Y. Kim *et al.*, "AUDIT: Stress testing the automatic way," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2012, pp. 212–223.
[5] J. Leng, A. Buyuktosunoglu, R. Bertran, P. Bose, and V. J. Reddi, "Safe limits on voltage reduction efficiency in GPUs: A direct measurement approach," in *Proc. 48th Annu. IEEE/ACM Int. Symp. Microarchit.*, 2015, pp. 294–307.
[6] R. Thomas, N. Sedaghati, and R. Teodorescu, "EmerGPU: Understanding and mitigating resonance-induced voltage noise in GPU architectures," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2016, pp. 79–89.
[7] X. Hu, G. Yan, Y. Hu, and X. Li, "Orchestrator: A low-cost solution to reduce voltage emergencies for multi-threaded applications," in *Proc. Des. Autom. Test Europe Conf. Exhib.*, 2013, pp. 208–213.
[8] V. J. Reddi, M. S. Gupta, G. Holloway, G.-Y. Wei, M. D. Smith, and D. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in *Proc. IEEE 15th Int. Symp. High Perform. Comput. Archit.*, 2009, pp. 18–29.
[9] Z. I. Chowdhury, S. K. Khatamifard, Z. Zheng, T. Moreshet, R. I. Bahar, and U. R. Karpuzcu, "Barrier synchronization vs. voltage noise: A quantitative analysis*," in *Proc. IEEE Int. Symp. Workload Characterization*, 2019, pp. 263–267.
[10] L. Renganarayana *et al.*, "Programming with relaxed synchronization," in *Proc. ACM Workshop Relaxing Synchronization Multicore Manycore Scalability*, 2012, pp. 41–50.
[11] M. Rinnard, "Parallel synchronization-free approximate data structure construction," in *Proc. 5th USENIX Workshop Hot Topics Parallelism*, 2013, Art. no. 6.
[12] S. Misailovic *et al.*, "Dancing with uncertainty," in *Proc. ACM Workshop Relaxing Synchronization Multicore Manycore Scalability*, 2012, pp. 51–60.
[13] F. Niu *et al.*, "HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 693–701.
[14] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2011, pp. 1–12.
[15] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks, "Quantifying sources of error in McPAT and potential impacts on architectural studies," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit.*, 2015, pp. 577–589.
[16] B. Sinharoy *et al.*, "IBM power7 multicore server processor," *IBM J. Res. Develop.*, vol. 55, no. 3, pp. 1:1–1:29, May/Jun. 2011.
[17] R. Zhang *et al.*"Architecture implications of pads as a scarce resource," *SIGARCH Comput. Archit. News*, vol. 42, pp. 373–384, 2014.
[18] C. Bienia, "Benchmarking modern multiprocessors," PhD thesis, Comput. Sci. Dept., Princeton Univ., Princeton, NJ, 2011.
[19] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," NASA Advanced Supercomputing Division, USA, Tech. Rep. NAS-99-011, Oct., 1999.
[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. 22nd Annu. Int. Symp. Comput. Archit.*, 1995, pp. 24–36.
[21] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of SPLASH-2 and parsec," in *Proc. IEEE Int. Symp. Workload Characterization*, 2009, pp. 86–97.
[22] PARSEC Group, "A memo on exploration of SPLASH-2 input sets," 2011. [Online]. Available: http://parsec.cs.princeton.edu/parsec3-doc.htm
[23] D. H. Bailey *et al.*, "The NAS parallel benchmarks summary and preliminary results," in *Proc. ACM/IEEE Conf. Supercomput.*, 1991, pp. 158–165.
[24] I. Akturk, K. Khatamifard, and U. R. Karpuzcu, "On quantification of accuracy loss in approximate computing," in *Proc. 12th Annu. Workshop Duplicating Deconstructing Debunking*, 2015, pp. 1–9.