Barrier Synchronization vs. Voltage Noise: A Quantitative Analysis*

Zamshed I. Chowdhury
University of Minnesota
chowh005@umn.edu

S. Karen Khatamifard
University of Minnesota
khatami@umn.edu

Zhaoyong Zheng
Brown University
zhaoyong_zheng@brown.edu

Tali Moreshet
Boston University
talim@bu.edu

R. Iris Bahar
Brown University
ruth_bahar@brown.edu

Ulya R. Karpuzcu
University of Minnesota
ukarpuzc@umn.edu

***Abstract*—Synchronization is the key to guaranteeing correct execution of parallel programs at any scale. Barriers represent heavily used synchronization primitives which prevent parallel tasks from proceeding to subsequent stages of computation before all tasks are done with previous stages. Accordingly, all tasks wait at a barrier until the slowest tasks finish, at which point all tasks can proceed to the next stage of computation. This usually translates into an abrupt change in the activity, i.e., current demand from the power delivery network, and if not orchestrated properly, can easily lead to voltage emergencies. In this study we characterize the impact of different barrier structures on voltage noise.**

***Index Terms*—barrier synchronization, voltage noise**

I. Motivation

By construction, the current demand from the power delivery network (PDN) closely tracks activity of logic and memory blocks. Therefore, over the course of execution, changes in workload activity inevitably result in spatio-temporal changes in the current demand I. Under a fixed power budget P, due to $P = V \times I$, this in turn renders fluctuations in the supply voltage V. As a result, V may deviate from its nominal value, V_{NOM}(the ideal operating voltage for which the system is designed). While overshoots over V_{NOM} challenge the physical integrity of the PDN, undershoots, which form the focus of this study, directly affect the operating speed due to the operating frequency f being proportional to V. A lower effective operating V than V_{NOM} can result in significant slow-down such that operation at the nominal frequency f_{NOM} becomes impossible and timing errors emerge.

The typical design practice for dealing with *voltage noise* (i.e., worst-case fluctuations around V_{NOM}) is to add a voltage guardband to mask its impact. Voltage noise comes in two flavors: 1) The parasitic resistance R of the PDN causes a voltage drop (with respect to V_{NOM}), proportional to $I \times R$, irrespective of the rate of change in activity, and 2) the parasitic inductance L of the PDN causes voltage undershoots or *droops* (again, with respect to V_{NOM}), proportional to $L \times dI/dt$. For case 2), the magnitude of voltage droops evolves as a function of how abruptly the activity (i.e., current demand) changes over time (dI/dt). Tailoring the voltage guardband to mask dI/dt induced droops is more challenging, simply because predicting potential changes in runtime activity is harder, where an overly conservative guardband can only degrade energy efficiency. At the same time, even a conservative guardband may miss worst-case droops and can still lead to *voltage emergencies*. This is especially concerning for activity patterns matching the resonance frequency of the PDN, where PDN parasitics, and hence voltage noise, assume their peak values.

As a fundamental construct to guaranteeing correct execution of parallel programs, barrier synchronization, by definition, can induce particularly problematic activity patterns. All parallel tasks or threads wait at a barrier until the slowest thread finishes computation. Only at that time, all threads can proceed to the next stage of computation. This usually translates into an abrupt change in the activity, i.e., dI/dt, and if not orchestrated properly, can induce voltage emergencies. Figure 1 demonstrates this effect, which gets exacerbated at higher thread (core) count.

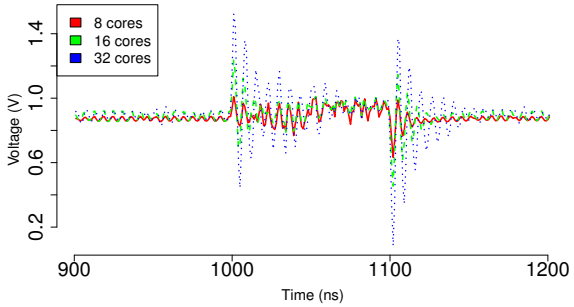

Fig. 1: Barrier induced voltage noise. Each thread runs on a separate core. The x-axis captures the activity in the vicinity of a representative barrier.

As critical constructs for the correctness and speed of parallel execution, the vast majority of barrier proposals are performance-optimized [1], [2]. Barrier synchronization has three phases: *(i)* arrival at the barrier; *(ii)* waiting in the barrier (for arrival of the slowest thread); *(iii)* release from the barrier. Performance optimization aims to minimize the time spent across all phases, *irrespective of the impact on voltage noise*. Program specifics and execution dynamics dictate phases (i) and (ii), which leaves very limited room for performance optimization. This picture, however, changes for phase (iii), which features a rich optimization space for thread release schedules. At the other end of the spectrum reside power-optimized barriers [3], [4], where the focus shifts to phase (ii), usually by bringing threads to deep-sleep states to minimize

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

power waste during waiting, *irrespective of the impact on voltage noise*. In fact, power-optimized barriers thereby increase the activity differential and exacerbate voltage noise.

As intuition suggests, a voltage-noise-optimized barrier must prevent such abrupt activity changes, i.e., avoid transitioning between highest and lowest activity states. The goal in this case primarily becomes smoothening the activity differential between phases (ii) and (iii), respectively, which usually translates into a slower thread release schedule than enabled by performance-optimized barriers. While voltage-noise-aware barrier proposals exists (e.g., [5]), we argue in this paper that performance-optimized barriers often can achieve a similar voltage signature, at the same time, without compromising performance. This is because performance-optimized barriers cannot afford deep-sleep states in phase (ii) which incur very long wake-up times. Therefore, the activity differential between phases (ii) and (iii) cannot be substantial, by construction.

Both, the (dynamic) barrier count and the (time) distance between subsequent barrier invocations determine the voltage signature. Typical scientific applications feature many barriers that usually are close enough (in the dynamic execution trace) to exhaust the voltage guardband. Such barriers primarily demarcate parallel loops to prevent race conditions. Voltage droops can only become more pronounced if the PDN does not have enough time to recover between subsequent barrier invocations. At the same time, most of these applications conform to weak scaling [6], i.e., the problem size naturally expands as higher number of cores (i.e., threads) become available, which challenges barrier-induced voltage noise mitigation further (see Figure 1).

We demonstrate, in this paper, that performance-optimized barriers often provide a better performance versus voltage-noise trade-off than naïve voltage-noise-aware barriers. The rest of the paper is organized as follows: Section II covers the related work; Section III, the basics; Sections IV and V, the evaluation; and Section VI, a discussion and summary of our findings.

II. Related Work

As we will detail in Section III, different implementations of performance-optimized barriers [1], [2] and voltage-noise-aware barriers [5] exist. On the other hand, power-optimized barriers [3], [4], focus on minimizing power dissipation during the wait time in the barrier by having the waiting threads transition to lower power states and staying there until the slowest thread arrives. For example, the Thrifty Barrier [3] determines specific low power states of each waiting thread as a function of anticipated wait time at the barrier, which in turn gets estimated from past execution history. Unfortunately, these policies typically exacerbate voltage noise. Previous stressmark generation proposals focusing on capturing and characterizing resonance [7]–[10] also emphasize the significance of synchronization and alignment of microarchitectural events. EmerGPU [11] further analyzes resonance effects in GPUs due to lock-step (SIMD) execution. Orchestrator [12] 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., [13]), focus on barrier-induced voltage noise.

III. Basics

We will next take a closer look into representative performance-optimized and voltage-noise-aware barriers, and will compare and contrast them, considering the three phases of barrier execution.

A. Voltage-Noise-Aware Barriers

Voltage-noise-optimized barriers must avoid abrupt activity changes, specifically when it comes to transitioning from phase (ii) to phase (iii). Threads may go to deep sleep or simply wait idle in phase (ii). Even if deep sleep is not the case, there is a significant difference in activity between idle wait in phase (ii) and (active) release in phase (iii). Therefore, releasing threads in a staggered fashion can help prevent abrupt changes in activity.

Linear Exit (LE) and **Bulk Exit (BE)** barriers [5] represent two voltage-noise-aware barrier implementations based on this idea. The LE barrier releases one thread at a time, in the reverse order of arrival. The time between the release of each individual thread, Δ, is a critical design parameter dictating the performance overhead. The performance overhead grows as $N \times \Delta$, where N reflects the number of threads. The BE barrier, on the other hand, releases threads in batches of size B, where $1 < B < N$ applies. Similar to LE, before release, each batch waits Δ time after the release of the previous batch. Accordingly, the performance overhead grows asymptotically by $N/B \times \Delta$. While the performance overhead reduces with increasing batch size B, the activity differential between phases (ii) and (iii) tends to increase. Therefore, compared to LE, BE results in higher voltage noise in exchange for lower performance overhead.

B. Performance-Optimized Barriers

Performance-optimized barriers cannot afford deep-sleep states in phase (ii), since they usually incur very long wake-up times. The distinctive feature of performance-optimized barriers is some form of busy-wait in phase (ii), which usually also helps optimize the thread release schedule in phase (iii). In phase (iii), threads can leave the barrier in batches or individually. The key is communicating to each thread that it is time to proceed. Accordingly, the busy-wait in phase (ii) mainly entails communicating the other threads' arrival among all the threads until reaching a consensus that all threads have arrived at the barrier. We will cover three representative examples next.

Dissemination Barrier (DB): Dissemination Barriers [2] rely on multiple rounds of pair-wise communication among threads based on a predefined gossip protocol in order to disseminate arrival information in phase (ii). After the last round of communication in phase (ii), staggered thread release in phase (iii) takes a very similar form to LE, but assuming a much smaller Δ.

Tree Barrier (MCS): Tree Barriers [1] maintain a tree to record and communicate the thread arrival times in phase (ii).

Every node in the arrival tree has a preset number of children nodes corresponding to threads, which notify their parents as soon as they arrive at the barrier. Once the root node and all of its children have arrived at the barrier, thread release can start in phase (iii). The thread release schedule follows the traversal of this tree starting from the root, where each node triggers the release of its children once done. Therefore, MCS releases threads in batches as well, but the batch size increases as we get closer to the leaves.

Tournament Barrier (TOURN): Tournament Barriers [2] are similar to DB, except that at each round two threads compete with each other, where only the winner proceeds to the next round. A champion emerges at the end of the last round, which indicates that all threads have arrived at the barrier. At this point, the champion triggers thread release in phase (iii), and the trigger signal propagates to previous rounds recursively. TOURN, as well, releases threads in batches. As each round features a different number of threads, the batch size varies during release.

IV. Evaluation Setup

Simulation Framework: We deploy Sniper [14] integrated with a revised version of McPAT [15] to collect runtime power traces which capture microarchitectural activity 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]. In all experiments, we pin threads to cores to minimize simulation noise.

Barriers: We implement different performance optimized (DB, MCS, TOURN) and voltage noise aware (LE, BE) barriers using both OpenMP and POSIX threads. We experiment with two representative values of Δ for LE, 10 cycles and 20 cycles, respectively. We exclude $\Delta > 20$ cycles, which results in smoother voltage signatures, yet incurs an excessive performance overhead with increasing thread count. For a fair comparison, we evaluate the voltage signatures under each barrier implementation at *iso-performance*, i.e., we set timing related degrees of freedom for each barrier implementation in a way such that the performance overhead incurred by each implementation is equal. Thereby, we equate the time spent at each barrier, from the arrival of the first thread until the release of the last. For each node in the arrival tree of the MCS barrier there is a maximum of 4 children, and in the thread release tree there is a maximum of 2. For thread release, we experiment with batch sizes B of 2, 4, 8, and 16 for BE. The batch size for MCS and TOURN varies between 2–16 (depending on the round of communication).

Thread Release Schedules (BE, MCS, TOURN): Since BE, MCS, and TOURN, release threads in multiple batches, we have the option of releasing the intermediate batches at different points in the time window tw, between the end of release of the first batch and the start of release of the last batch. This decision does not have an impact on performance overhead (where the release time of the last batch matters), but can affect the voltage noise as different release schedules give rise to different activity patterns over tw. To capture this effect, we align such release times to the start (S), middle (M), and end (E) of the time window tw in the experiments.

Baseline for Comparison: As a comparison baseline throughout the evaluation we use an ideal, performance-optimized barrier which does not incur any performance overhead during release, (i.e., all threads are released simultaneously upon the arrival of the last thread at the barrier). By construction, this barrier features the worst voltage noise.

V. Evaluation

A. Maximum Droop

Using Δ = 10 cycles, Figure 2 shows the percent improvement in maximum voltage droop (due to barrier synchronization), compared to the maximum droop under the ideal, performance-optimized barrier from Section IV (which by construction incurs the worst voltage noise). We experiment with a thread count of 8 and 32 (labeled M8 and M32 for LE and DB, respectively). As explained in Section IV, we have an additional degree of freedom for batch scheduling during thread release for BE, MCS and TOURN: For intermediate batches we can align the release of any other batch to the start (S), middle (M), or end (E) of the time window between the release of the first and last batch. Bars labeled by SX, MX, and EX, respectively, capture this effect for different thread counts of X=8 and 32.

Not surprisingly, we observe that LE and DB provide a better percentage improvement when compared to other barriers that release threads in batches with a batch size $B > 1$. LE and DB both release threads one after another; therefore, the activity differential between the waiting period in the barrier and the release from the barrier – between phases (ii) and (iii) – increases in the smallest possible steps over time, which does not lead to a significant droop overall. Moreover, DB outperforms LE by 1.53X and 1.13X for 8 and 32 threads, respectively, since DB keeps threads highly active whereas LE allows threads to go to sleep while waiting in the barrier. DB thereby reduces the activity differential between phases (ii) and (iii).

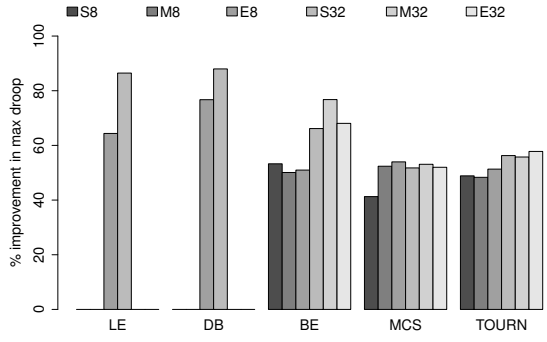

Fig. 2: % improvement in maximum droop for Δ = 10 cycles.

MCS and TOURN barriers, which rely on batch release of threads, exhibit very similar behavior for 8 threads since their thread release schedule takes a very similar form under a thread count of 8. However, as the thread count increases to 32, TOURN shows a higher improvement due to a more

pronounced difference in batch release patterns at higher thread count. More specifically, over the course of release, the batch size tends to reduce for TOURN, but monotonically increase for MCS. As results show, S8, M8 and E8 differ by up to 11.6% for MCS and only by 2.5%, for TOURN. This indicates a higher sensitivity of MCS to the alignment of batch release schedules. As the thread count increases to 32, the difference in S32, M32 and E32 gets even less pronounced for both.

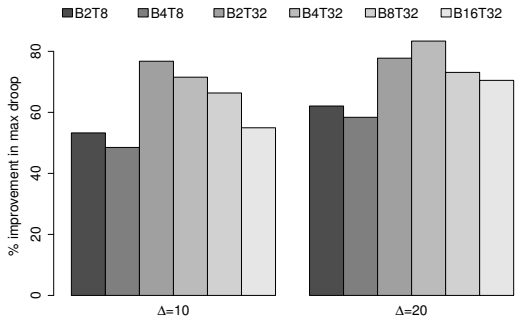

Fig. 3: BE's sensitivity to batch size. $BXTY$: B=batch size, T=thread count (X, Y = corresponding value of B, T).

In comparison to BE, which represents a voltage-noise-aware barrier, both MCS and TOURN exhibit a lower improvement in maximum voltage droop, mainly because BE features the smallest batch size (and thereby smallest activity differential during release) among the three. To look into this effect closer, Figure 3 captures the impact of the batch size on the maximum droop under BE, for both Δ = 10 cycles and 20 cycles. Overall, we observe that the percent improvement in maximum droop under BE reduces with increasing batch size B, for both 8 and 32 threads. The only exception is $B4T32$, which shows a marginal increase over $B2T32$. This is likely due to the combined effect of B being too small relative to the total thread count 32, and the relative alignment between the release of batches. When the maximum batch size in MCS and TOURN gets closer to the batch size in BE, the percent improvement in maximum droop for each differs by less than 2% compared to BE. This indicates that, for equal (or similar) batch size, MCS and TOURN can closely match BE in terms of maximum droop.

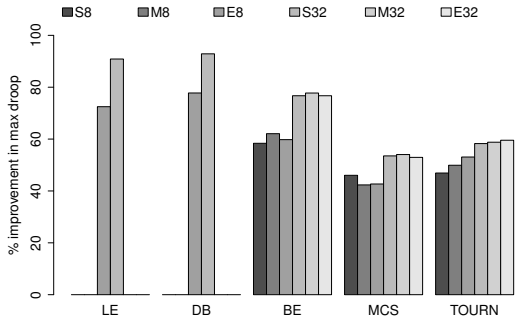

Fig. 4: % improvement in maximum droop for Δ = 20 cycles.

Figure 4 shows how the picture changes for Δ = 20 cycles, as the duration between two successive thread releases doubles. Overall, when compared to Figure 2, we observe a slightly higher improvement for all barrier implementations, as a result of releases being further apart from each other in time, which in turn provides more time for the PDN to settle down between successive releases. DB outperforms LE in this case, as well, for both 8 and 32 threads. Similar to Figure 2, MCS and TOURN exhibit smaller improvement in comparison to BE, mainly due to the larger effective batch size.

We have two takeaways from these experiments: First, gradual thread release results in less voltage noise than batch release. This makes sense intuitively since a larger batch size implies more threads being released at the same time, which in turn leads to a higher activity differential. Second, at higher thread counts, performance-optimized barriers can closely match or even exceed the maximum droop improvement achieved by voltage-noise-aware barriers, while delivering a better voltage noise vs. performance trade-off.

VI. Conclusion & Discussion

As a critical construct for the correctness and the speed of parallel execution, barrier synchronization can result in abrupt changes in activity, and thereby induce significant voltage noise. While the vast majority of classic proposals cover only performance optimization to reduce the timing overhead, recent voltage-noise-aware variants trade voltage noise with performance.

Contrary to intuition, we argue and demonstrate in this paper that performance-optimized barriers can achieve a similar (or even less-noisy) voltage signature, when compared to voltage-noise-aware variants, without compromising performance. This is simply because performance-optimized barriers cannot afford putting threads in deep-sleep states while waiting in the barrier, which naturally reduces the activity differential between waiting in the barrier and release from the barrier, the main contender for voltage noise.

Specifically, we confirm that gradual thread release (one at a time) renders less voltage noise than releasing threads in batches, where performance-optimized barriers such as DB (already relying on gradual thread release), can deliver a better voltage noise vs. performance trade-off than voltage-noise-aware barriers such as LE. Similarly, performance-optimized barriers such as MCS and TOURN, which rely on batch release, can result in similar voltage noise when compared to the voltage-noise-aware counterpart BE, without compromising performance.

We did not assume any specific voltage guardband throughout the evaluation and analyzed raw voltage traces instead. In fact, any improvement in voltage noise (i.e., maximum droop) can translate into a reduction in the required voltage guardband, which in turn leads to lower power dissipation at the same performance point. We also explicitly did not include power-optimized barrier implementations in the evaluation, as these barriers by construction feature the highest voltage noise. This is because minimizing power dissipation during the wait time in the barrier almost exclusively results in the maximum activity differential. That said, the voltage-noise-aware barriers that we analyzed in this study all put threads to sleep (while waiting at the barrier), hence represent voltage-noise-aware power-optimized barriers.

REFERENCES

[1] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," *ACM TOCS*, 1991.

[2] D. Hensgen, R. Finkel, and U. Manber, "Two algorithms for barrier synchronization," *Int. J. Parallel Prog.*, 1988.

[3] J. Li, J. F. Martinez, and M. C. Huang, "The thrifty barrier: energy-aware synchronization in shared-memory multiprocessors," in *HPCA*, 2004.

[4] C. Liu, A. Sivasubramaniam, M. Kandemir, and M. J. Irwin, "Exploiting barriers to optimize power consumption of cmps," in *IPDPS*, 2005.

[5] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "Vrsync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in *ISCA*, 2012.

[6] J. L. Gustafson, "Reevaluating Amdahl's Law," *Commun. ACM*, 1988.

[7] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. Carey, R. F. Rizzolo, and T. Strach, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *MICRO*, 2014.

[8] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan, "Audit: Stress testing the automatic way," in *MICRO*, 2012.

[9] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, (Washington, DC, USA), pp. 77–88, IEEE Computer Society, 2010.

[10] 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 *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 294–307, Dec 2015.

[11] R. Thomas, N. Sedaghati, and R. Teodorescu, "Emergpu: Understanding and mitigating resonance-induced voltage noise in gpu architectures," in *ISPASS*, 2016.

[12] X. Hu, G. Yan, Y. Hu, and X. Li, "Orchestrator: A low-cost solution to reduce voltage emergencies for multi-threaded applications," in *DATE*, 2013.

[13] V. J. Reddi, M. S. Gupta, G. Holloway, G. Wei, M. D. Smith, and D. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 18–29, Feb 2009.

[14] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations," in *SC*, 2011.

[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 *HPCA*, 2015.

[16] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, *et al.*, "IBM power7 multicore server processor," *IBM J. Res. Dev.*, 2011.

[17] R. Zhang, K. Wang, B. H. Meyer, M. R. Stan, and K. Skadron, "Architecture implications of pads as a scarce resource," *SIGARCH Comput. Archit. News*, 2014.