GPU Voltage Noise: Characterization and Hierarchical Smoothing of Spatial and Temporal Voltage Noise Interference in GPU Architectures

Jingwen Leng Yazhou Zu Vijay Janapa Reddi
Department of Electrical and Computer Engineering
The University of Texas at Austin
jingwen@utexas.edu, yzu@utexas.edu, vj@ece.utexas.edu

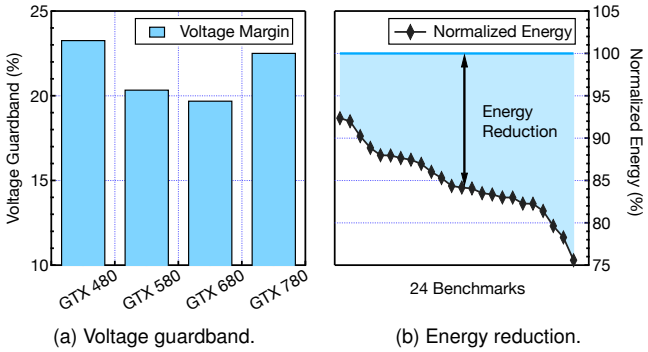
Abstract

Energy efficiency is undoubtedly important for GPU architectures. Besides the traditionally explored energy-efficiency optimization techniques, exploiting the supply voltage guardband remains a promising yet unexplored opportunity. Our hardware measurements show that up to 23% of the nominal supply voltage can be eliminated to improve GPU energy efficiency by as much as 25%. The key obstacle for exploiting this opportunity lies in understanding the characteristics and root causes of large voltage droops in GPU architectures and subsequently smoothing them away without severe performance penalties. The GPU's manycore nature complicates the voltage noise phenomenon, and its distinctive architecture features from the CPU necessitate a GPU-specific voltage noise analysis. In this paper, we make the following contributions. First, we provide a voltage noise categorization framework to identify, characterize, and understand voltage noise in the manycore GPU architecture. Second, we perform a microarchitecture-level voltage-droop root-cause analysis for the two major droop types we identify, namely the local first-order droop and the global second-order droop. Third, on the basis of our categorization and characterization, we propose a hierarchical voltage smoothing mechanism that mitigates each type of voltage droop. Our evaluation shows it can reduce up to 31% worst-case droop, which translates to 11.8% core-level and 7.8% processor-level energy reduction.

1. Introduction

The Green500 list shows an increasing trend of heterogeneous supercomputing systems at the top of the energy-efficiency list [1]. As general-purpose GPU architectures increasingly take the place of CPU computing in datacenters and supercomputers, each GPU unit's energy efficiency plays a critical role because of the overall energy costs and economies of scale involved. To that end, it has become important to reduce GPU power consumption without impacting its performance.

We focus on improving the GPU's energy efficiency by optimizing away inefficiency at the guardband level. GPU architectures, as do CPU architectures, typically rely on large voltage guardbands to tolerate real-world worst-case operating conditions that occur because of deviations in process, voltage, and thermal (PVT) variations. Prior work in CPU architectures shows that the voltage guardband can be as large as 20% [13], which can lead to significant energy inefficiency because worst-case conditions rarely occur during execution. Measurement results in Fig. 1a show that GPUs require similarly large voltage guardbands to tolerate worst-case conditions. The measurement approach is similar to prior work [18].



(a) Voltage guardband. (b) Energy reduction.

Fig. 1: Opportunity of exploiting the voltage guardband. (a) Measured voltage guardband on four commercial graphics cards. (b) Measured energy reduction benefits on GTX 480.

Typically, a large portion of the voltage guardband is allocated to tolerate supply voltage noise [8], i.e., the turbulence at the on-chip V_{dd} plane, due to interactions between the processor's current flow and its nonzero impedance power delivery network. Voltage noise can cause unreliability in the system due to sudden droops in the operating voltage. If the operating voltage transiently dips below assumed conditions, it may cause circuit-timing violations that result in processor failure or the generation of incorrect execution results.

Because large supply voltage noise is a rare event, it presents us with a unique opportunity to improve energy efficiency. Lowering the guardband to typical operating conditions and smoothing away large transient voltage droops improve energy efficiency. Various techniques have been proposed to smooth voltage noise in CPU architectures [6, 9, 10, 12, 15, 23, 24, 26, 27]. However, to the best of our knowledge, no such characterization and techniques exist for GPU architectures.

In this paper, we perform a thorough characterization and provide a deep understanding of voltage noise activity in GPU architectures. Our ultimate goal is to reduce the voltage guardband by smoothing voltage noise to improve energy efficiency. Fig. 1b shows measured results of energy efficiency improvements that can be achieved on the GTX 480 hardware while

running a variety of programs from the Rodinia, LoneStar, and CUDA SDK suites. On existing off-the-shelf hardware, we can achieve improvements between 8% and 25%. The improvements vary because different workloads have different program characteristics and microarchitectural activity that impact on-chip voltage noise by varying magnitudes. Thus, unlocking the potential for changing the guardband relies on a fundamental understanding of the voltage noise root causes.

Fig. 2 presents an overview of our work. Given the GPU's manycore architecture and single-program, multiple-data execution model, we first propose a generally applicable spatial and temporal voltage noise categorization framework to comprehensively analyze voltage noise in manycore GPU architectures. The temporal noise analysis focuses on the time domain response of voltage noise as it interacts with the PDN impedance profile. The spatial noise analysis focuses on how different cores, (i.e., single-instruction, multiple-data execution units) interfere with each other to aggravate or alleviate voltage noise. We demonstrate that studying both together is necessary to capture all interactions occurring in the GPU.

Our spatial and temporal analysis reveals two major voltage noise droop types in GPU architectures. First, the fast-occurring transient first-order droops impact only a small cluster of cores. Second, the more slowly occurring transient second-order droops have a chip-wide effect, impacting all cores across the entire die. In order to smooth voltage noise in GPUs, it is necessary to tackle both forms of voltage droops.

To understand the activity leading to the two voltage droop types, we conduct microarchitecture-level analysis. The local first-order droops are caused by microarchitectural stalls followed by bursty activity at the register file and dispatch units. The GPU has a large power-hungry (banked) register file to support the massive set of parallel SIMD pipelines. The dispatch unit controls warp execution. Suddenly dispatching a large number of warps to idling execution units causes a large surge in current that leads to a fast-occurring voltage transient.

The global second-order droops are caused by implicit synchronization points that are induced by I- and D-cache aligned miss activity, as well as thread block execution alignment. Unlike CPUs, GPUs lack an explicit chip-wide thread synchronization primitive at the software layer. Thus, global synchronization manifests only through low-level microarchitectural activity. Implicit synchronization happens sufficiently enough in some programs that it is the dominant droop type.

These two types of voltage droops have distinctive requirements for voltage smoothing. For example, first-order droop happens very quickly, which makes it difficult to detect and provides little response time for smoothing. On the other hand, the global second-order droop happens more slowly but involves the chip-wide activities. Naively throttling the chip-wide activities could incur a large performance overhead.

To smooth GPU voltage noise, we propose a novel hierarchical voltage smoothing mechanism, where each level specifically targets one type of voltage droop, as shown in Fig. 2. For

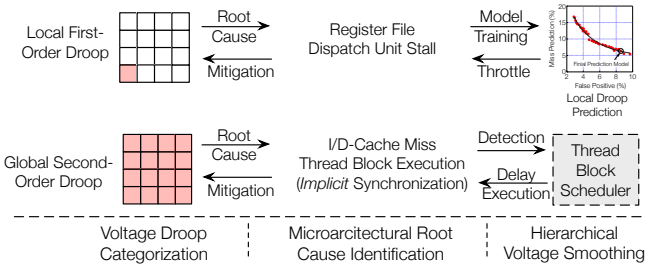


Fig. 2: Overview of our work.

the first-order droop, we train a prediction model to predict the local first-order droop using our root-cause analysis data based on register file and dispatch unit activity. The model is trained offline to avoid an expensive first-order droop detection mechanism and provide enough response time for smoothing to work. For the second-order droop, caused by the implicit synchronization due to aligned I- and D-cache misses and thread block execution, our smoothing mechanism leverages current hardware communication mechanisms and delays execution to disrupt the current and future synchronization pattern. As a result, it effectively smooths the voltage with negligible performance overhead.

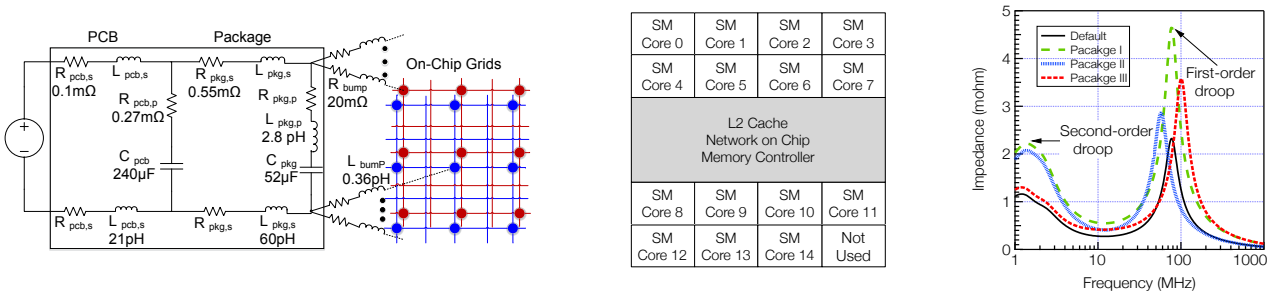
The hierarchical voltage smoothing mechanism reduces the worst-case droop by 31% and eliminates 99% of all voltage emergencies. The effectiveness of our proposed mechanism confirms the insights we made in our characterization. The reduced worst-case droop enables a smaller voltage guardband that enables energy-efficiency improvements. We observe an average 7.8% energy savings. The reduced voltage emergencies can also enlarge the energy benefits for future resilient GPU architectures with a hardware fail-safe mechanism that can tolerate voltage emergencies at an additional penalty.

The rest of the paper is organized as follows. Sec. 2 introduces the experimental setup we use for modeling and studying voltage noise in a simulated environment. We rely on simulation to gather highly specific insights. Sec. 3 explains the discovery of dominant droop types. Sec. 4 presents their microarchitecture-level root-cause analysis. Sec. 5 discusses how we mitigate the worst-case voltage droop on the basis of the hierarchical voltage smoothing mechanism. Sec. 6 evaluates our proposed mitigation method. Sec. 7 discusses related work and compares our GPU voltage noise findings with characterization and techniques that are well established for CPUs. We close with our concluding thoughts in Sec. 8.

2. Experimental Methodology

In this section, we describe our simulation infrastructure setup to study voltage noise in GPU architectures. Modeling the voltage noise requires the support of a GPU performance model, power model, and a power delivery network (PDN) model. We use cycle-based simulation for these components.

Performance Simulation We use GPGPU-Sim (version 3.2.0) [2, 3], a cycle-level performance simulator, running



(a) Details of power delivery network model (PCB, package, and on-chip PDN).

(b) GPU processor layout.

(c) Package impedance profile.

Fig. 3: Voltage modeling details. (a) Default PDN details. (b) Simulated GTX 480 layout. (c) Simulated PDNs' impedance profile.

CUDA 4.0 [22] and its PTX ISA. The key simulated microarchitectural parameters are summarized in Table 1. We use the default GTX 480 configuration provided by the simulator.

Power Simulation We use GPUWattch [16, 17] for the power simulation. GPUWattch is integrated with GPGPU-Sim for cycle-level power calculation. The authors of GPUWattch claim that the power model has an average accuracy that is within 9.9% error of the measured results for the GTX 480.

Voltage Simulation We use the GPUVolt [19], which is a GPU-specific voltage simulator. GPUVolt simulates the power delivery network (PDN) and its electrical characteristics to calculate voltage at every cycle. The input to the voltage simulator is the cycle-level current profile from GPUWattch.

Fig. 3a illustrates the power delivery network model that is assumed and simulated by GPUVolt. It consists of the printed circuit board (PCB), the package, and the on-chip PDN. For the on-chip PDN, we follow GPUVolt's modeling assumption that a GPU processor adopts a shared PDN among all the cores. Prior work has shown that a shared PDN in a manycore CPU is more robust to voltage noise than a split power grid where cores are connected to separate power grids [13]. In addition, GPUVolt uses a distributed shared grid model to capture intercore interference activity. The grids are configured to correspond to the GTX 480 layout that is shown in Fig. 3b.

The off-chip model used in GPUVolt is derived by scaling the parameters used in the original Pentium 4 model published by Gupta et al. [11]. The scaling factor is determined by the ratio of GPU's peak thermal design power (TDP) compared to the Pentium 4 processor, based on the assumption that the design PDN should match the target processor architecture's peak current draw [7,14]. GPUVolt uses a conservative scaling 2× (compared to the 4× TDP ratio between two processors).

Configuration	Value	Configuration	Value
Number of SMs	15	SM clock frequency	700 MHz
Threads per SM	1536	Threads per warp	32
Registers per SM	128 KB	Memory controller	FR-FCFS
Shared memory	48 KB	Memory bandwidth	179.2 GB/s
Memory channels	6	Memory controller	FR-FCFS
Warp scheduling policy		Greedy-then-oldest [28]	
L1 cache (size/assoc/block size)		16 KB/4-way/128 B	
L2 cache (size/assoc/block size)		768 KB/16-way/128 B	

Table 1: GPGPU-Sim parameters.

We show the impedance profile of our simulated PDN in Fig. 3c. We use the default configuration (2× scaling) for GTX 480 to conduct our analysis. The PDN model together with GPGPU-Sim and GPUWattch are co-configured to resemble GTX 480. GPUVolt has a correlation factor of 0.9 against experimental measurement for this default setup [19].

We also sweep the parameters of both on-chip and off-chip components in order to provided sensitivity analysis for our work. Package I has about 2× both first-order and second-order droop impedance compared to the default package in Fig. 3c. Package II has the same first-order impedance but 2× second-order impedance. Package III has the same second-order impedance but 2× first-order impedance.

In the rest of this paper, we rely on breaking down and associating the magnitude of the voltage noise caused by activity on different cores (Sec. 3). We also need the breakdown to understand voltage noise's association with microarchitectural activity to identify the voltage noise root causes (Sec. 4). But because GPUVolt takes the power trace of all SM cores to compute the PDN's supply voltage profile as a whole, it cannot reveal core-level noise activity. To decompose the contribution to voltage droop from the different sources, i.e. cores, we leverage the PDN's linear property. The PDN model is simply an RLC network; thus, its voltage response from all the cores is equivalent to the case when each core's current trace is supplied separately and then added all together. We verify that this linear property holds true by simulating each individual core, aggregating the voltage output of all the cores, and comparing that sum with GPUVolt's simulated output. The resulting difference is less than 1%, i.e., negligible.

Benchmark Selection & Characteristics We use applications from a diverse set of benchmark suites to faithfully characterize voltage noise in GPUs. The simulated worst-case voltage noise of these chosen applications has been compared and validated against hardware measurements. Moreover, the applications demonstrate a large range of voltage droop behavior [19]. Both these reasons make them ideal for our study.

The application set includes five large programs from the CUDA SDK: BlackScholes (BLS), convolutionSeparable (CVLS),

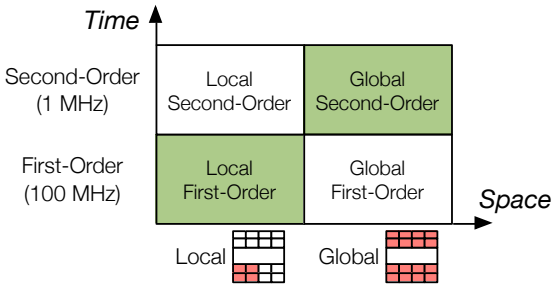


Fig. 4: GPU voltage noise categorization framework.

convolutionTexture (CVLS), dct8x8 (DCT), and binomialOptions (BO); seven from Rodinia [5]: BACKP, KMN, SSSP, NNC, CFD, MGST, and NDL; and the DMR program from LoneStarGPU [4].

The BLS program has the largest worst-case voltage droop (12%), whereas NNC has the smallest (4.6%). The large worst-case droop is what the allocated guardband must tolerate, so we conservatively assume the operating margin is determined by BLS. In reality, however, the worst-case guardband is much larger because it is required tolerate the other variations types.

3. Temporal and Spatial Voltage Noise Analysis

In this section, we present a conceptual framework for characterizing voltage noise in the manycore GPU architecture. We examine voltage noise in the temporal (i.e., time varying) and spatial (i.e., core versus chip-wide) dimensions. Using this framework, we show that there are two main types of GPU voltage noise: the fast-occurring first-order droops that are localized to a small cluster of neighboring cores, and the slow-occurring chip-wide second-order droops. We explain why and how these two types manifest in the GPU. We also explain why identifying and understanding these types is important.

3.1. Why Temporal and Spatial Analysis?

Understanding voltage noise activity in the GPU architecture is particularly challenging because of the nature of the GPU's manycore architecture. In GPUs, each core is an independent source of voltage noise, and a large transient voltage droop anywhere in the chip is the result of interactions between voltage noise generated from different cores. Therefore, to characterize GPU voltage noise thoroughly, we must quantify its spatial interference effects between the different cores. In addition, we also need to consider voltage noise's frequency-domain characteristics because the impedance profile of the GPU's power delivery network (Fig. 3c) has both first- and second-order peak values that can enlarge voltage noise occurring at the corresponding rates. Overlooking either of these factors can lead to incomplete characterization results.

To tackle this issue, we propose a characterization framework that contains two dimensions: temporal and spatial analysis. Temporal analysis is designed to study the voltage noise's frequency-domain characteristics. It filters out voltage noise at different rates and captures their features individually. Next, for each type of voltage noise categorized in temporal analysis,

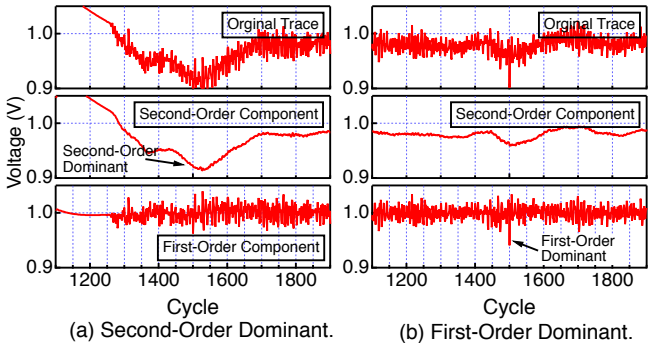


Fig. 5: Comparison between the first- and second-order droop effects: (a) second-order dominant. (b) first-order dominant.

the spatial analysis decomposes its noise contribution from different sets of cores to expose spatial interference. These two steps, along with two orthogonal dimensions, allow us to comprehensively characterize voltage noise in GPUs.

The characterization framework has four quadrants, as shown in Fig. 4. The *y*-axis focuses on temporal analysis. The frequency ranges we focus on are limited to the first- (100 MHz) and second- (1 MHz) order frequencies (Fig. 3c) because they have higher impedance values and cover most of the frequency spectrum. The *x*-axis focuses on spatial analysis, which is divided into local and global regions. Local analysis studies a cluster of four neighboring cores (e.g., cores 1, 2, 5, and 6 in Fig. 3b). Global analysis focuses on all the cores.

Each of the four quadrants in Fig. 4 indicates a specific type of voltage droop: local first- and second-order droop, and global first- and second-order droop. Prior work [14, 23] focused on single-core voltage noise pointed out that the chip-wide current fluctuation around the first-order droop frequency is the most important and noticeable form of voltage noise. However, our analysis shows that 1) voltage noise in the many-core architecture is significantly different from that in the single-core architecture; and 2) all four types of droop do not occur in the GPU architecture. The only two possible types are the local first-order droops and global second-order droops.

3.2. Temporal Analysis: First- vs. Second-Order Droop

The results of our analysis emphasize that both first- and second-order droop are important in GPU architectures. Unlike prior work [14, 23] that only shows the severity of first-order droop in the single-core CPU architecture, we show that the second-order droop also matters in the manycore architecture. In some cases, it can be the leading cause of droops.

The first step is to separate a voltage droop into its first- and second-order droop components. The voltage model calculates the time domain response of the power delivery network, which mixes both the first- and second-order droops together. To separate the two types of droops, we apply a low-pass filter to the original voltage trace. Because the first-order droop happens at a much higher frequency (100 MHz as compared to 1 MHz), the resulting trace contains only the second-order

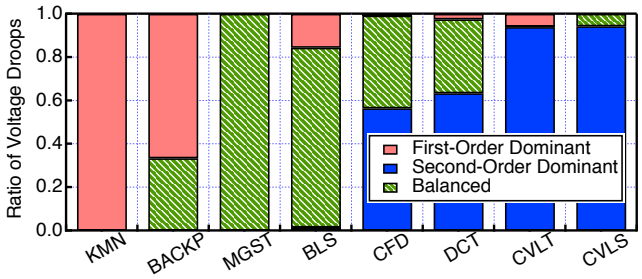


Fig. 6: The ratio of first- and second-order droops.

droop. The subtraction of the original trace from the second-order droop trace results in the first-order droop trace.

Fig. 5 illustrates the separation process. The top plot shows the original simulated voltage trace, which is the superposition of the first- and second-order droop components. The middle plot shows the low-pass filtered voltage trace, i.e. second-order droop component. The bottom plot shows the trace of first-order droop component, derived by subtracting the second-order component from the original trace.

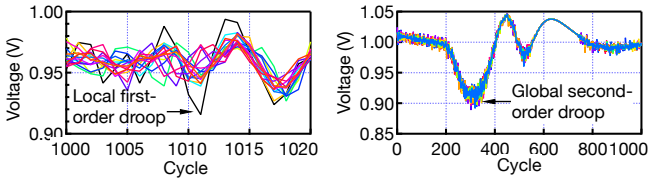
After applying the filter to separate the first- and second-order droop components, it is easy to identify the droop types. Fig. 5a shows a voltage droop that is caused by the second-order component. The duration of the droop matches the expected second-order droop characteristics, approximately 700 cycles, because the cores are operating at 700 MHz. Similarly, Fig. 5b shows an example where the voltage droop is caused by the first-order component. Its droop duration also matches the expected first-order droop characteristics for the simulated architecture's clock frequency and PDN behavior.

On the basis of this proposed approach, we are able to quantify the extent to which a program is affected by the first-order and second-order droops. We define a droop as *first-order (or second-order) dominant* if the first-order (or second-order droop) contributes more than 60% to the total droop. We call the remaining cases balanced because neither droop is dominant. We first focus on the voltage droops larger than 8%, and expand the analysis to other droops in the end.

Fig. 6 shows the percentage of first- and second-order dominant droops for the simulated workloads. The figure shows that both first- and second-order dominant droops exist in GPUs. It also shows that the benchmarks have different behaviors. For example, KMN and BACKP are first-order dominant, whereas CFD, DCT, CVLT and CVLS are dominated by second-order droops. Some other workloads, such as MGST and BLS, consist mainly of balanced droops. From here on forward, we use shorthand and refer to first-order and second-order dominant droop types succinctly as first-order and second-order droops.

3.3. Spatial Analysis: Local vs. Global Droop

In this section, we demonstrate how the physical spatial locality of the cores interacts with the temporal components (i.e. first- and second-order) of the voltage droop. We show that although first- and second-order voltage droop are important, they manifest under two different circumstances. Our analysis



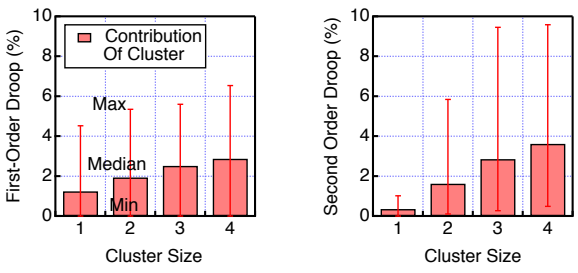
(a) First-order droop. (b) Second-order droop.

Fig. 7: Example of local first- and global second-order droop.

demonstrates that the first-order droop is caused only when a small cluster of neighboring cores interact. For example, only SM 0 experiences a large first-order droop in Fig. 7a (each line represents the voltage trace of an SM core). As such, we define this type of voltage droop as *local first-order droop*. However, the second-order droop is caused by global chip-wide activity. As a result, all SM cores experience the second-order droop in Fig. 7b, which we define as *global second-order droop*.

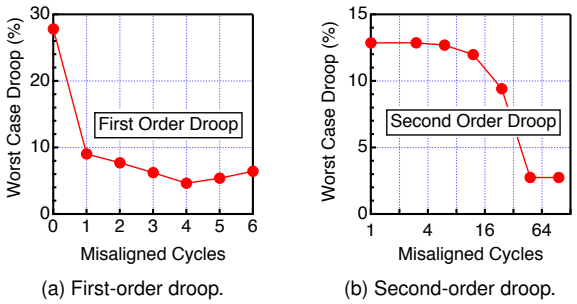
We conduct the spatial analysis on all voltage droops that are greater than 8%. We pick 8% because when we analyze the distribution of the droop magnitude, we find that 8% distinguishes the worst-case droop from the typical-case droop [19]. We postprocess the data for spatial analysis. We decompose the ratio of voltage droop contributed by each core (as described previously in Sec. 2), and then decouple its first- and second-order components (as described previously in Sec. 3.2). We aggregate the droop contribution into different numbers of SM cores, depending on the "cluster size." A cluster size of two means that we consider four adjacent cores (e.g., cores 1, 2, 5, and 6 in Fig. 3b). A cluster size of four implies all cores and a cluster size of one means only one SM core.

Our characterization data in Fig. 8 shows that the first- and second-order droops have distinctive spatial properties. Although both the first- and second-order voltage droops increase with a higher number of active cores, the manner in which they increase is markedly different. The first-order droop shown in Fig. 8a increases noticeably from a cluster size of one to two, but beyond that point it shows marginal increase, meaning that a large number of active cores does not help much in building first-order droops. Second-order droop, however, shown in Fig. 8b, increases much more as the cluster sizes increase. Therefore, we determine that a small cluster of local cores is the main contributor of first-order droops, whereas the second-order droop is mostly caused by chip-wide activities.



(a) First-order droop. (b) Second-order droop.

Fig. 8: Voltage droop contribution from core clusters.



(a) First-order droop. (b) Second-order droop.

Fig. 9: Sensitivity analysis of the first-order and second-order droop toward misaligned core execution activity.

3.4. Only Local First- and Global Second-Order Matter

In this section, we explain why only local first-order and global second-order droops are important in GPUs. The other two temporal and spatial combinations illustrated in Fig. 4 are not critical because they do not contribute to large voltage droops.

Prior work has shown that the degree of activity alignment across cores is important because in-phase current surge build up large voltage droops [27]. The key determinant here, however, is the likelihood that multiple small GPU cores simultaneously align at either the first-order or second-order frequency to cause a corresponding large voltage droop. We use Fig. 9 to explain the sensitivity analysis of the first-order and second-order droops toward activity (mis)alignment. The experiment is conducted by feeding synthesized sine-waveform current traces matching first- or second-order droop frequency to each core, and varying the alignment cycles between them.

Local First-Order Droop The first-order droop is a fast-occurring transient effect. As such, the likelihood that activity across a large die consisting of multiple small cores is largely aligned is minimal, especially because voltage noise has a decaying effect that affects how a voltage droop propagates [13].

Fig. 9a shows that the first-order droop is extremely sensitive to misalignment. Even a one-cycle misalignment can reduce the worst-case droop from 28% to 9% because the first-order frequency is high (100 MHz) and gives the cores little time to align their activities. As a result, only a small number of cores can run into aligned activities. Note that this 28% is not the absolute worst-case droop for the simulated applications, but rather a theoretical upper limit of the worst-case droop magnitude assuming the identical current fluctuation of all cores matching the first-order droop frequency.

Furthermore, the first-order droop effects can only be felt by neighboring cores, not far apart cores, owing to the aforementioned decaying propagation effect. We find that the voltage droop caused by two cores can vary from 3% to 6% depending on the core's spatial distance to the two active cores.

Global Second-Order Droop The second-order droop, on the other hand, has a much higher tolerance for activity misalignment. Fig. 9b shows the second-order droop's sensitivity alignment. It can tolerate up to 20 cycles of misalignment because of its relatively lower frequency (around 1 MHz).

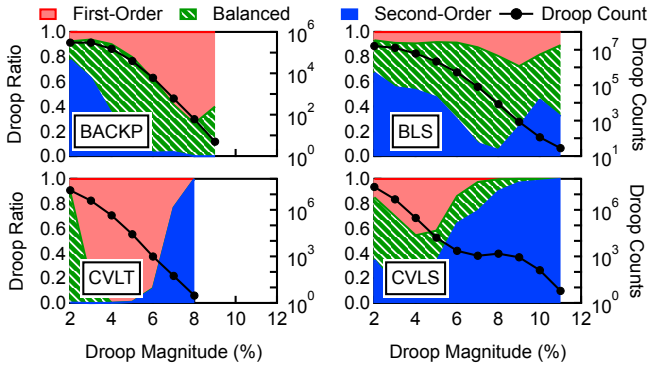


Fig. 10: Ratio of first- and second-order droops, and total number of droops (in log scale) with different droop thresholds.

Chip-wide activities can synchronize at the rate, and lead to the global second-order voltage droops that affect all the cores.

Furthermore, the second-order droop manifests as a global, rather than a local, chip-wide droop because of the PDN's impedance characteristics. The second-order droop's impedance is lower than the first-order droop's impedance (Fig. 3c), and therefore it requires much larger current fluctuations to cause a large droop as compared to the first-order droop. Large and sudden current variations do not occur at the first-order droop because of the first-order droop's misalignment sensitivity, as discussed previously, but such large current variations can occur at the more slowly accumulating second-order droop frequency. Thus, only the global second-order droop occurs, and the local does not manifest.

Maximum Droop We conclude this subsection with a comparison between the local first- and global second-order droop for their maximum magnitude. Fig. 8 shows that the maximum magnitude of first- and second-order droop is 6.4% and 9.5%, respectively. This means that the second-order droop caused by chip-wide activities has a larger maximum droop magnitude than the first-order droop caused by local activities.

3.5. Voltage Droop Threshold Sensitivity Analysis

The magnitude of worst-case droop determines the supply-voltage operating margin. We demonstrate that the results of our spatial and temporal analysis are similar for droops >4%, and as such the insights from our previous analysis on droops >8% are applicable while operating with a different threshold.

We perform the temporal and spatial analysis for voltage droops with magnitudes greater than 2%, all the way to their maximum droop (e.g. CVLT only goes to 8%). The results are shown in Fig. 10 for four representative benchmarks. The value on the *x*-axis indicates that we analyze droops greater than that value. The *y*-axis on the left plots the ratio between the different types (first-order, balanced, and second-order dominant). The *y*-axis on the right plots the total droop count.

By inspecting the total droop count in Fig. 10, we observe that the 5% droop threshold (compared to the 8% threshold we assume for our analysis) is also capable of distinguishing the

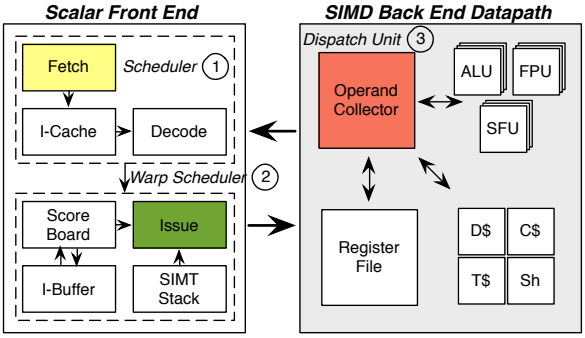


Fig. 11: SM core microarchitecture and its schedulers.

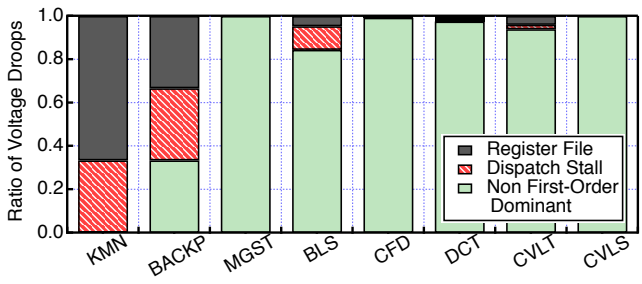


Fig. 12: Breakdown of local first-order droop root causes.

typical-case from the non-typical-case droop. In other words, droops >5% account for only 1-10% of the total droop counts.

Fig. 10 also shows that all three droop types are prevalent for droops >4%. Moreover, we observe workload-dependent behavior. For example, BACKP has mostly first-order droops, whereas CVLS has mostly second-order droops. However, the workload characteristics can be represented by our analysis with the 8% threshold. Thus, our conclusions from the previous sections also applies to the non-typical-case droops.

4. Microarchitecture-Level Root Cause Analysis

In this section, we analyze the microarchitecture-level root causes of the local first-order and global second-order droops. We examine all first- and second-order droops larger than 8%.

For the local first-order droop (Sec. 4.1), we divide the chip into four clusters. Each cluster contains four cores. We identify the cores responsible for the droop by examining each core's current surge at the first-order droop frequency. Next, to capture the root cause we inspect microarchitectural stalls. Prior work in CPUs has shown that stalls can lead to large droops, owing to the sudden activity burst following the stall [11]. Given that an SM has three schedulers in the scalar front end and SIMD backend (Fig. 11) and that the relationship between these schedulers and voltage droop is unknown, we consider stalls of the three schedulers (e.g., warp scheduler) and also the major microarchitectural components in an SM.

For the global second-order droop (Sec. 4.2), almost every large second-order droop involves synchronized current surge, contributed by at least nine cores. Prior to the current surge of each core is a stall period that spans over a few hundred cycles (i.e., second-order droop frequency). We consider microarchitecture-level root causes for the long stall period. The microarchitectural activity we consider includes all the possible events that prevent the threads from being issued, such as barrier synchronization, as well as various cache misses.

Our key findings are as follows. The dispatch unit stalls and register file activity are the major causes of local first-order droop, and *implicit synchronization* of the activities across cores is the predominant cause of second-order voltage droops. These root causes comprehensively identify 99% of all droops.

4.1. Local First-Order Voltage-Droop Root Causes

Dispatch Unit Stall A GPU core has three schedulers to maximize throughput, as shown in Fig. 11. The front-end fetch scheduler takes instructions from the I-cache in a round-robin manner to create an instruction pool of different warps. The issue scheduler, more commonly known as the warp scheduler, selects and sends warps to the backend in accordance with operand data availability and dependencies. The last scheduler located in the backend, also called the dispatch unit, accesses the register file, taking bank conflicts and write reservations into consideration, and sends instructions to the SIMD units.

Of the three schedulers, the issue scheduler is mostly studied due to its large impact on performance [28]. However, we find that the last scheduler (i.e., the dispatch unit) rather than the issue scheduler is the direct cause of large first-order droops.

Execution activity stalls in the dispatch unit can induce local first-order droops for two reasons. First, the SIMD backend is the most power-consuming component in GPUs [17]. The dispatch unit is closely and tightly coupled with the power-hungry backend components that can cause current surges, such as the register file. Second, the time the dispatch unit takes to ramp up the backend aligns with the first-order droop's frequency. By comparison, when the two other schedulers stall, the cycles it takes to ramp up the power of the backend datapath exceeds the first-order droop cycles, because it takes time for the activity to propagate through the schedulers.

Register File Modern GPUs require a large register file to hold the architectural states of thousands of threads in each core. It is 128 KB in our simulated GTX 480 architecture. Compared to all the other power-hungry components in the backend, the register file experiences the largest and fastest current changes, in a manner that aligns with the first-order droop frequency, thus making the register file a voltage noise hotspot inside the core. We extended the analysis of our local first-order droop to include the register file, and find that the sudden current surges cause a large number of voltage droops.

Cause Distribution Fig. 12 shows the ratio of the first-order droops caused by the register file and dispatch unit. The ratio only applies to the first-order droop here, and the "non-first-order dominant" droop (second-order or balanced) will be analyzed later. We find the register file and dispatch unit contribute roughly the same in BACKP, BLS and CVLT. KMN

```
...
st.global.f32 [%rd7+0], %f2;
ld.const.f32 %f3, [ff_variable+8];
mul.lo.s32 %r6, %r4, 2;
add.s32 %r7, %r3, %r6;
cvt.s64.s32 %rd8, %r7;
mul.wide.s32 %rd9, %r7, 4;
add.u64 %rd10, %rd1, %rd9;
st.global.f32 [%rd10+0], %f3;
ld.const.f32 %f4, [ff_variable+12];
mul.lo.s32 %r8, %r4, 3;
...
```

Fig. 13: PTX code of kernel *cuda_initialize_variables()* in CFD.

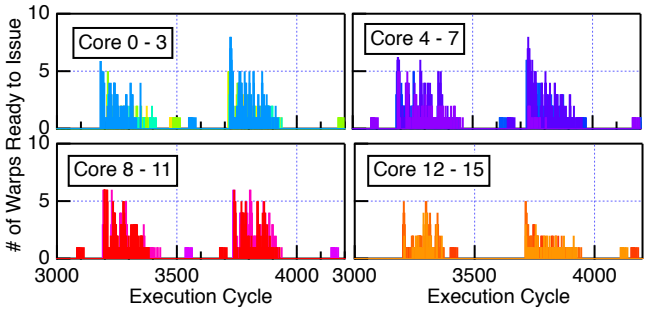


Fig. 14: Snapshot of showing D-cache misses alignment.

has mostly register-file-induced droops. We examine its PTX source code to verify this observation and find that it has a large sequence of back-to-back PTX instructions that manipulate and move operands around in the register file. Because the register file consumes a large portion of the total power [17], its dominant cause to voltage droops in KMN makes sense.

4.2. Global Second-Order Voltage-Droop Root Causes

The global second-order droop is the result of temporally aligned multicore activity. Prior work (VRSync [20]) concludes that explicit synchronization points in CPU programs can cause large voltage droops. However, our analysis of the global second-order voltage droop in GPUs reveals that implicit synchronization is the major source of temporally aligned activity across multiple GPU cores.

The implicit synchronization across cores is the artifact of the GPU architecture's single-program, multiple-data (SPMD) execution model: all the cores execute the same code. The consequence of the SPMD model is that each core can experience very similar microarchitectural events, such as instruction or data cache misses. Another important observation of the implicit synchronization is that it has a recurring pattern because the microarchitectural event causing the synchronization is also recurring. We identify three dominant types of microarchitectural root causes for implicit synchronization in the GPU architecture. These are the data-cache miss stalls, instruction-cache miss stalls, and thread block alignment.

Data-Cache Miss Stall Although GPUs are designed to hide memory access latency, a cache miss can still stall the pipeline. The pipeline stalls when all executing threads miss in the D-cache and no threads can be issued. Data hazards are intuitive because the GPU cannot perform out-of-order execution. Therefore, following a data-cache miss return, there can be a surge of activity that causes a voltage droop.

However, we find that there is one more reason for data-cache miss stalls that can lead to large voltage droops. When the memory request rate of the threads exceeds a core's ability to hide the access latency, all threads may stall for the outstanding/pending memory requests to complete due to a structural hazard. Because the memory requests are serviced in a round-robin fashion, after the stall each core starts up at almost the same time, which can cause a globally aligned sudden power increase that can cause a voltage droop.

Fig. 13 shows a code example. In Fig. 13, there are multiple *st.global* instructions that store the result back to the global memory address space. Each thread executing the first store instruction experiences a D-cache write miss. The cores continue executing the following instructions because there is no true dependency on this *st.global*. However, as more threads experience the write misses, the memory request buffer in the load-store unit becomes full, and the threads cannot *issue* the second store instruction, as well as any other instructions that follow. As a result, all cores ramp down to a halt, as seen in the corresponding execution graph (Fig. 14). As the structural hazard eases, warp activity starts ramping up across all the cores, which creates the chip-wide aligned current burst that ultimately results in a global second-order droop.

Instruction-Cache Miss Stall Another cause of global second-order droop is core-wide instruction-cache miss activity. The simulated GTX 480 architecture has a 2 KB I-cache [29]. When the program's instruction footprint exceeds the I-cache, each core experiences an I-cache miss stall when a branch or an instruction outside the I-cache is executed.

Fig. 15 shows an example of an I-cache miss stall that occurs across many cores. We show an execution snapshot of a kernel in benchmark CFD. The kernel's binary size is about 4 KB, which exceeds the I-cache size. To demonstrate the correlation between the I-cache miss and a voltage droop, each subplot in the figure shows the power on the left y-axis and number of warps experiencing I-cache misses per cycle on the right y-axis. In the highlighted interval, almost all the cores except 4, 11, and 13 experience I-cache misses and incur a power surge afterward. The last subplot shows the total power from all cores increases from 40 watts to 100 watts in the highlighted interval, which leads to a large global second-order droop.

Thread Block Alignment Thread blocks can cause second-order voltage droops in one of two forms. First, thread block launches can cause global second-order droops when multiple thread blocks align with each other at the start of execution. The resulting power variation behavior is equivalent in activity to a bursty pipeline following a microarchitectural stall. When a kernel launches, multiple thread blocks sent to all cores can cause aligned activity that leads to a current surge.

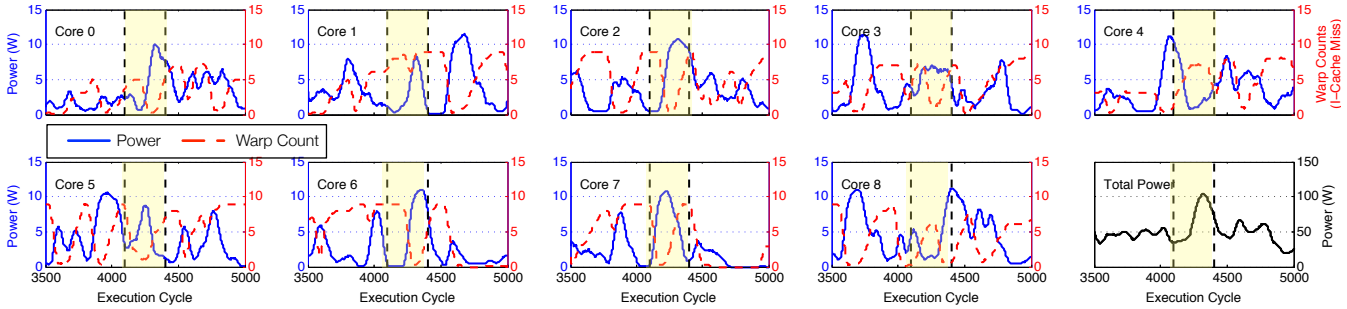


Fig. 15: Snapshot of a kernel in CFD. The aligned power surge is due to chip-wide I-cache misses in all cores except 4.

Second, aligned thread block execution activity can also happen in the middle of kernel execution. In regular programs, thread blocks typically have similar execution times. It implies that a batch of thread block launches may likely end at the same time, immediately followed by the simultaneous issue of pending thread blocks, thus leading to sudden power variation. Fig. 16 shows this effect happening repeatedly midway through execution by plotting the number of active warps per cycles for a kernel in DCT. The launch of a new batch of thread block results in repeated power spikes at around 14,000, 15,500, and $\sim$ 17,000 cycles. The aligned thread block activity across all cores causes recurring global second-order droop.

Cause Distribution To understand the importance of each global second-order droop root cause, we show the ratio of droops caused by the three sources in Fig. 17. The data-cache and instruction-cache misses are the two most important causes for second-order dominant droops. However, this does not necessarily mean that the thread block launch is less important, because it can cause extremely large droops (10% in BLS and CVLS). Each program has *only* one dominant cause. For example, voltage droops in CFD and DCT are mostly caused by the data-cache misses, which conforms with prior work that describes their memory-bound characteristics [17]. The dominant cause in CVLT and CVLS is instruction-cache misses, because their kernel size exceeds the instruction-cache size.

5. Voltage Smoothing

In this section, we describe the details of our voltage smoothing mechanism for the two dominant types of voltage droops we have identified: local first-order and global second-order droops. We first show how the space-time voltage-droop categorization framework helps us to reason about the voltage smoothing requirement for each type of droop. Recognizing that the two dominant types of droops have distinctive smoothing requirements, we propose a *hierarchical* smoothing mechanism. In particular, we propose a two-level hierarchical smoothing mechanism where each level smooths one droop type. Our hierarchical method is succinct and adds little overhead to the design process when implemented in hardware.

5.1. The Need for a Hierarchical Smoothing Mechanism

We explain the need for a hierarchical voltage smoothing mechanism in GPUs by understanding the design requirement of smoothing mechanisms imposed from voltage-droop spatial and temporal characteristics. Because in the GPU architecture each type of voltage droop has its unique temporal and spatial characteristics, hardware-smoothing mechanisms must be designed accordingly to match their needs. The rest of this subsection explains our design considerations in detail.

Voltage Smoothing Mechanism Characteristics Typically, a voltage smoothing mechanism has two parts – a front-end detector and an actuator. The front end monitors the trend of processor voltage variation, either by inferring it through certain processor microarchitectural events [20, 23, 26], or by measuring it directly with voltage sensors [14, 15]. Once the front end decides the supply voltage would likely drop below a certain threshold, it triggers the actuator, which is responsible for mitigating the droop (e.g., via processor throttling).

The smoothing mechanism has three characteristics: response time, actuation scope, and duration. The first one pertains to the front end and describes the time slack it has to detect the droop, whereas the other two describe the actuator.

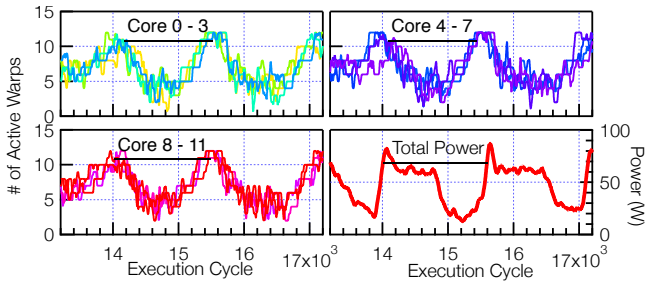


Fig. 16: Snapshot of showing thread block alignment.

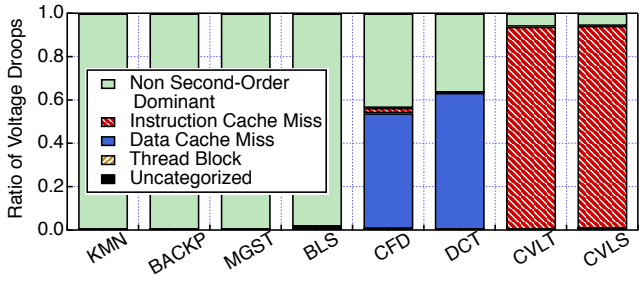
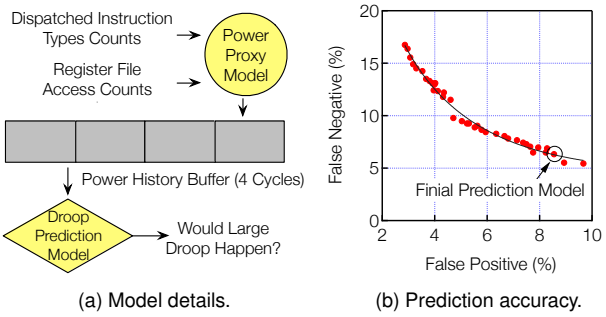


Fig. 17: Breakdown of global second-order droop root causes.



(a) Model details. (b) Prediction accuracy.

Fig. 18: Local first-order droop predictor. (a) Overview. (b) The prediction accuracy: false negative versus false positive.

Droop Type and Smoothing Characteristics Our proposed temporal and spatial analysis framework in Sec. 3.1 helps us reason about the aforementioned voltage smoothing mechanism characteristics. For high-frequency first-order droop, the front end has less than 10 cycles' response time. Failing to respond to it in time will miss predicting the droop. Meanwhile, the short duration of first-order droops also means that it requires a very short mitigation duration. In contrast, a second-order droop spans over 500 cycles. Therefore, it has a more relaxed response-time requirement and needs a longer duration to mitigate the droop. In terms of spatial characteristics, local droops require the actuation scope to focus on a small cluster of local cores, whereas global droops require full chip actuation because they are caused by chip-wide activities.

Effectively smoothing voltage noise with minimal performance overhead requires the smoothing mechanism to cater to the droop's spatial and temporal characteristics. A naive smoothing mechanism throttling the whole chip for a long duration would incur large performance and power overhead.

5.2. Local Voltage Smoothing

The first level in our smoothing mechanism is designed for the local first-order voltage droops. In Sec. 4.1 we pointed out that local first-order droops are mostly caused by dispatch unit stall and register file accesses. Based on this finding, we construct a model that uses the dispatch unit and register file activities to predict if large first-order droops will occur. Then the actuator will throttle core activities accordingly.

Front End We train a local voltage-droop prediction model to detect the local first-order droop. Because the first-order droop has a very short transition time, a prediction model provides fast enough response time for the actuator.

Fig. 18a shows the details of our prediction model. The local first-order droop is predicted by inspecting each core's power variation rate. A four-entry FIFO is adopted to record the power history in the past four cycles. In each cycle, a new power estimation gets enqueued, and a new prediction is made based on the history information. We find a four-cycle duration long enough to predict first-order droops.

For instantaneous power estimation, the model uses the number of instructions issued from the dispatch unit (including integer, floating-point, special functional, and load/store instructions), and register file access counts, as shown in Fig. 18a. We adopt these metrics because they are the dominant causes of first-order droops. We extract their energy information from GPUWattch to estimate the power consumption.

The predictor extracts the maximum power increment across two, three, and four cycles from the FIFO and compares them against a set of thresholds. If any of the power increment exceeds the corresponding threshold, a large droop is predicted. The threshold values are derived offline and deployed for online prediction. We trained this prediction model using a number (about 200) of large local droops (first-order droop larger than 8%) and a set of randomly chosen small droops.

We evaluate the prediction accuracy Pareto frontier of our model with a different set of thresholds in Fig. 18b. We select a final model with 7% false negative and 9% false positive rate. We can improve the local prediction model's accuracy by considering neighbor cores' activities. But we find our single-SM-based prediction works effectively for smoothing.

Actuator The actuator for local smoothing throttles the dispatch unit or the register file if a large droop is predicted. Each cycle, the local predictor examines the power history buffer. The warp dispatch or register file access will be throttled if the predictor deems that doing so would cause a large droop.

5.3. Global Voltage Smoothing

The second level in our hierarchical smoothing mechanism is for reducing the global second-order voltage droops. Because of its global spatial property, detecting the second-order droop requires knowledge of chip-wide activities. Our mechanism reuses the existing hardware components for global droop detection, and thus avoids the overhead of building an extra, expensive global chip-wide communication network or channels. Also, due to the long duration of the second-order droop, throttling the whole chip's activities would likely incur a large performance overhead. Thus, we have come up with a technique that effectively mitigates the second-order droop with minimal performance overhead. To achieve this, we must leverage the architecture's throughput-optimization features.

Front End Our root-cause analysis shows that implicit synchronization due to I-/D- cache miss and thread-block execution is the root cause for the global second-order droop. An important implication of this is that we can the detect global second-order droop by monitoring each SM's active warp counts. A warp becomes inactive if it experiences I- or D-cache misses, or finishes all of its instructions. From Fig. 14 to Fig. 16, we observe that the global droop occurs when most SMs transit from a small number of active warps to a large number of active warps. Our front end decides a global droop would happen if it detects at least nine cores transiting from zero active warps to at least one active warp in a 30-cycle window. The reason we choose this value is that if the alignment cycle among cores exceeds that value, the voltage droop

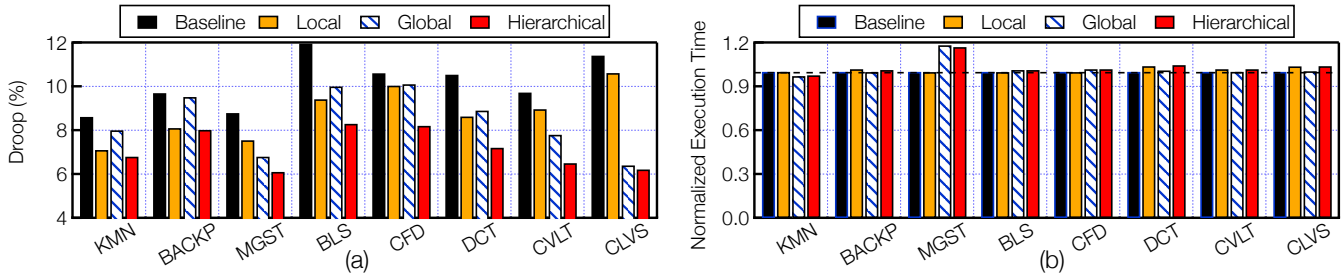


Fig. 19: Evaluation of local-only, global-only and hierarchical smoothing. (a) Worst-case droop. (b) Normalized execution time.

magnitude drops dramatically, as shown previously in Fig. 9b.

We leverage the GPU's thread block scheduler to make the aforementioned decision. Each SM has a communication channel to the thread block scheduler to inform when a thread block has finished the execution [21]. This channel is rarely used because a thread block normally takes at least thousands of cycles to finish. Thus, the SM can use it to report the number of active warps periodically (10 cycles in our implementation) to the thread block scheduler. The scheduler then performs detection based on the global information of all SMs. This centralized method makes global second-order droop detection practical and requires only a small hardware modification.

Actuator An intuitive actuator for mitigating global second-order droop is to throttle chip-wide activities for a long enough period. But this would cause large performance degradation. Other than throttling, our actuator delays each SM's execution by 50 cycles when a global droop is detected. This minimizes the performance degradation for two reasons. First, as shown in Fig. 9b, a misalignment of 50 cycles, which accounts for only 1/10 of the second-order droop duration, can effectively reduce the second-order droop. Second, we find the global alignment in GPUs often demonstrates a recurring pattern (e.g. Fig. 14 and Fig. 16). Staggering the execution not only mitigates the current alignment, but also the future recurring alignment. This reduces the number of times the global actuator is triggered, further minimizing the smoothing overhead.

6. Evaluation

In this section, we evaluate the proposed hierarchical voltage smoothing mechanism, considering its smoothing effect on the worst-case droop magnitude reduction. We then discuss the impact of voltage smoothing on energy improvements by reducing the voltage guardband. Besides the hierarchical mechanism, we also evaluate the two smoothing levels individually – local only and global only – and show that they are not as effective as the combined hierarchical smoothing. We also discuss the performance overhead and the sensitivity of the hierarchical smoothing to different packages characteristics.

Worst-Case Droop We evaluate the smoothing mechanism's effect on worst-case droop reduction because the the worst-case decides the amount of voltage guardband needed. Fig. 19a shows the worst-case droop of the simulated applications over baseline, local-only, global-only, and hierarchical voltage smoothing. The baseline has no smoothing applied. In Fig. 19a, the local-only smoothing can reduce the worst-case droop in local first-order-droop-dominant applications such as KMN and BACKP, but cannot mitigate the worst-case droop in global second-order-droop-dominant applications such as CVLT and CVLT. We observe the opposite effect for global-only smoothing. However, the combined hierarchical smoothing effectively reduces the worst-case droop of all applications. The reduction is 21% for the first-order-dominant application KMN, 29% for the balanced-droop-dominant application BLS, and 45% for the second-order-dominant application CVLS. The geometric mean of reduction for all applications is 27%.

Performance Overhead We also evaluate the performance overhead of our smoothing mechanism. Fig. 19b shows the normalized execution time for the same four scenarios as in Fig. 19a. Generally, the geometric average execution time increase is 3.5% for hierarchical voltage smoothing. We also show the overhead of local-only and global-only smoothing, which is only 1.2% and 2.3%, respectively. Note that global smoothing increases the execution time of MGST by 16% because the global synchronization in the application does not have a regularly recurring pattern and that makes global smoothing trigger many more times. Overall, the overhead is small because large voltage droops are rare and do not occur frequently during execution. Thus, the benefit of hierarchical voltage smoothing is that it does not hurt performance when droops are occurring infrequently, and performance degradation is only experienced when droops are predicted.

Energy Consumption We also evaluate the energy saving benefit of voltage smoothing by lowering the supply voltage. With hierarchical smoothing in place, the worst-case droop among all applications decreases from 12% to 8.3% in BLS. So we assume the supply voltage can reduce from the nominal 1 V to 0.96 V. Fig. 20 shows the normalized energy consumption from core and noncore (L2 cache, NoC, and memory controller) for all applications. Note that we assume that the noncore components do not share the PDN with cores; thus, there is not much energy reduction for noncore components. The average energy saving after lowering the supply voltage is 11.8% for all cores and 7.8% for the whole processor.

Package Sensitivity We also consider the sensitivity of the hierarchical smoothing mechanism to package characteristics and show that it can still effectively reduce the worst-case

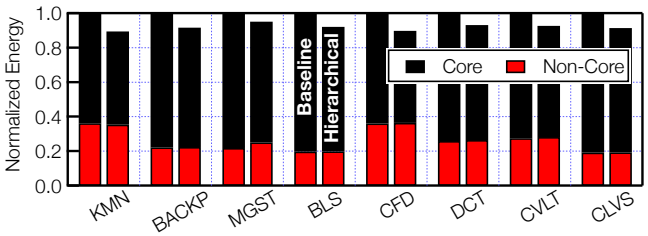


Fig. 20: Normalized energy savings of smoothing.

droop of all applications, regardless of package parameters. Due to space constraints we summarize only our key results.

We performed the sensitivity study on the three packages described in Sec. 2. The hierarchical voltage smoothing reduces the worst-case droop of all applications by 33.8%, 37.7%, and 22.75% for three packages. The average worst case droop reduction is 28%, 29%, and 21.6%. We notice that hierarchical smoothing is relatively more sensitive to the first-order impedance. Recall that the local prediction model was trained offline using the default package droop values, and that makes the model less effective for predicting large local droops in a package with higher first-order droop frequency. The model requires re-training to perform better for a different package.

Smoothing Aggressiveness Our smoothing mechanism can be tuned to target different operating voltage margins. Smoothing voltage noise aggressively lets the processor operate with a tighter margin, which can reduce energy consumption further. But the trade-off is the added performance overhead.

We discuss such a trade-off using Fig. 21. We use the target droop threshold (x-axis) as the metric for evaluating hierarchical smoothing aggresiveness. The 12% target is the baseline, i.e., no smoothing applied. The 8% target is the smoothing target evaluated previously. The 5% target is an aggressive optimization goal.

The worst-case droop of all benchmarks decreases as smoothing aggressiveness is increased, as shown in the left subplot of Fig. 21. However, performance overhead can be high. For example, the execution time (middle subplot) of DCT increases to 1.35× the baseline under the most aggressive case. The increased slowdown neutralizes energy improvement, resulting in no energy reduction for DCT (right subplot). We also note workload-dependent behavior: KMN's execution also increases to 1.25× under the most aggressive case but it still has over 10% energy reduction. This is caused by the difference in ratio between dynamic and static energy. KMN has a high portion of static energy, which is greatly increased during smoothing, whereas DCT has a high portion of dynamic energy, which is not increased as much during smoothing.

7. Related Work

To the best of our knowledge, this work represents the first comprehensive voltage noise characterization and smoothing effort for voltage noise in GPU architectures. We first study the voltage droop types categorized by our spatial and temporal analysis framework and subsequently analyze voltage

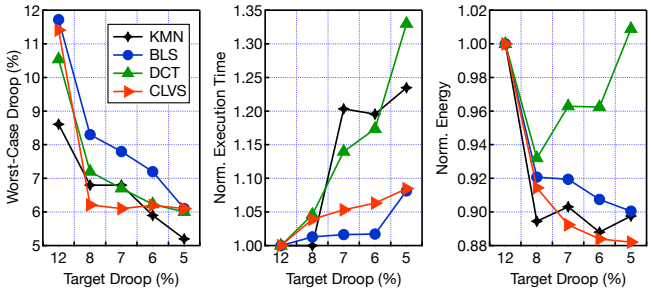


Fig. 21: Worst-case droop (left), performance (middle), and energy (right) trade-off with different smoothing aggressiveness.

interference in the manycore architecture. Following that, we study the microarchitectural root cause for each type of voltage droop, and propose a hierarchical voltage smoothing mechanism. To effectively smooth voltage noise, we present a systematic approach involving four key aspects. We compare and contrast with prior work in CPUs for all of the above.

Spatial and Temporal Analysis Most prior works in CPUs demonstrate the severity of the first-order voltage droop [14, 23, 24]. However, the scope of these works is only at the single-core level. Thus, their insight is mostly relevant to the local first-order droop in a multicore CPU. *VRSync* studies the global second-order droop in a 32-core CPU processor. The authors do not mention that they target on second-order droop, but we infer that by examining the droop duration in the paper.

In contrast, our work studies both the spatial and temporal properties of voltage droops in GPUs. The characterization effort reveals that both second-order and first-order droops are important. While the second-order droop is omitted by most works in the CPU domain, the second-order droop can be dominant in certain workloads for GPUs. Further more, our analysis demonstrates that only two types of voltage droop in GPUs matter: local first-order and global second-order droop.

Voltage Interference A shared power delivery network in the manycore architecture can amplify the amount of voltage droop. Both temporal alignment and spatial distance of cores' activities impact the amount of voltage nose interference that is experienced by the processor. Prior works in CPUs [10, 13, 27] focused only on the temporal alignment effect.

Our work shows that the GPU's manycore nature increases the role of spatial distance between cores' activities on voltage noise. Spatial distance combined with temporal alignment can lead to the constructive build-up of voltage noise that leads to both the local first- and global second-order droops in GPUs.

Microarchitectural Root Cause The local first- and global second-order voltage droops have different microarchitectural causes in both the CPU and GPU. In the CPU, the current surge after pipeline stall is the root cause of first-order droop. The microarchitectural events that cause pipeline stall include cache & TLB misses, and branch misprediction [7, 9, 14, 23].

Our work shows the complete pipeline stall does not cause the large first-order droop in GPUs. Instead, we identify that

stalls in last scheduler (out of three schedulers) – i.e., the dispatch unit – causes the large first-order droop. Besides that, the large power-consuming register file is another source.

In multicore CPUs, explicit synchronization such as barrier synchronization has been identified as a cause of the global second-order droop [20]. However, our analysis attributes implicit synchronization as a major root cause of such droop in GPUs. The microarchitectural events that cause implicit synchronization include the I- & D-cache miss and thread block launch. Moreover, explicit synchronization causes only a one-time voltage droop, whereas we find that implicit synchronization tends to induce *recurring* voltage droops in GPUs.

Smoothing Various hardware and software methods have been proposed to mitigate CPU voltage noise [9,11,20,23–26]. Our work differs from these in that our smoothing mechanism is hierarchical, more specifically two-level, and targets the GPU's unique microarchitecture-level voltage-droop causes.

8. Conclusion

We propose a hierarchical mechanism to smooth out voltage noise in GPU architectures. Our mechanism mitigates local first-order droops with a per-core voltage droop predictor and reduces global second-order droops by staggering the execution of core activities such as thread block issuing. The hierarchical mechanism is motivated by our voltage noise space-time characterization and the associated microarchitecture-level root-cause analysis. Our evaluation shows that the smoothing mechanism reduces up to 31% of the worst-case droop, which translates to 11.8% core-level energy reduction. But more importantly, the characterization work can enable future effort in exploring more aggressive guardbanding solutions for larger energy gain. Our work opens up new research possibilities, such as identifying the ideal operating voltage guardband for individual GPU kernels, dynamically tuning the guardband for various kernels during execution, performing timing speculation in GPUs using fail-safe hardware recovery mechanisms, smoothing voltage noise using novel compiler techniques, etc.

Acknowledgments

This work is supported by the National Science Foundation grant CCF-1218474. The views expressed in this paper are those of the authors only and do not reflect the official policy or position of the NSF or the U.S. Government.

References

[1] "Green500 List," www.green500.org, [Last accessed: Sep. 12, 2014].
[2] A. Bakhoda, G. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "GPGPU-Sim," www.gpgpu-sim.org.
[3] ——, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.
[4] M. Burtscher, R. Nasre, and K. Pingali, "A Quantitative Study of Irregular Programs on GPUs," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2012.
[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2009.
[6] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2003.
[7] E. Grochowski, D. Ayers, and V. Tiwari, "Microarchitectural simulation and control of di/dt-induced power supply voltage variation," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2002.
[8] M. Gupta, "Variation-Aware Processor Architectures with Aggressive Operating Margins," Ph.D. thesis, Harvard, 2009.
[9] M. S. Gupta *et al.*, "An event-guided approach to handling inductive noise in processors," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2009.
[10] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks, "Understanding Voltage Variations in Chip Multiprocessors Using a Distributed Power-delivery Network," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2007.
[11] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "Towards a software approach to mitigate voltage emergencies," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2007.
[12] ——, "Decor: A delayed commit and rollback mechanism for handling inductive noise in processors," in *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*. IEEE, 2008, pp. 381–392.
[13] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie, "Comparison of Split-Versus Connected-Core Supplies in the POWER6 Microprocessor," in *International Solid-State Circuits Conference (ISSCC)*, 2007.
[14] R. Joseph, D. Brooks, and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2003.
[15] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active Management of Timing Guardband to Save Energy in POWER7," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2011.
[16] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch," gpuwattch.ece.utexas.edu.
[17] ——, "GPUWattch: Enabling Energy Optimizations in GPGPUs," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2013.
[18] J. Leng, Y. Zu, and V. J. Reddi, "Energy Efficiency Benefits of Reducing the Voltage Guardband on the Kepler GPU Architecture," in *Workshop on Silicon Errors in Logic - System Effects (SELSE)*, 2014.
[19] J. Leng, Y. Zu, M. Rhu, M. Gupta, and V. J. Reddi, "GPUVolt: Modeling and Characterizing Voltage Noise in GPU Architectures," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2014.
[20] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: Characterizing and Eliminating Synchronization-induced Voltage Emergencies in Many-core Processors," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012.
[21] NVIDIA, *Fermi Compute Architecture Whitepaper*, 2009.
[22] NVIDIA Corporation, "NVIDIA CUDA Programming Guide," 2011.
[23] M. D. Powell *et al.*, "Pipeline damping: a microarchitectural technique to reduce inductive noise in supply voltage," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003.
[24] M. D. Powell and T. N. Vijaykumar, "Pipeline Muffling and a Priori Current Ramping: Architectural Techniques to Reduce High-frequency Inductive Noise," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2003.
[25] V. J. Reddi, M. S. Gupta, M. D. Smith, G.-y. Wei, D. Brooks, and S. Campanoni, "Software-assisted hardware reliability: abstracting circuit-level challenges to the software stack," in *Proceedings of the Design Automation Conference (DAC)*, 2009.
[26] 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 *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2009.
[27] V. Reddi, S. Kanev, W. Kim, S. Campanoni, M. 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 International Symposium on Microarchitecture (MICRO)*, 2010.
[28] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-Conscious Wavefront Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2012.
[29] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU Microarchitecture Through Microbenchmarking," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2010.