# EmerGPU: Understanding and Mitigating Resonance-Induced Voltage Noise in GPU Architectures

Renji Thomas, Naser Sedaghati and Radu Teodorescu
Computer Science and Engineering
The Ohio State University
{thomasr, sedaghat, teodores}@cse.ohio-state.edu

***Abstract*—This paper characterizes voltage noise in GPU architectures running general purpose workloads. In particular, it focuses on resonance-induced voltage noise, which is caused by workload-induced fluctuations in power demand that occur at the resonance frequency of the chip's power delivery network. A distributed power delivery model at functional unit granularity was developed and used to simulate supply voltage behavior in a GPU system. We observe that resonance noise can lead to very large voltage droops and protecting against these droops by using voltage guardbands is costly and inefficient. We propose EmerGPU, a solution that detects and mitigates resonance noise in GPUs. EmerGPU monitors workload activity levels and detects oscillations in power demand that approach resonance frequencies. When such conditions are detected, EmerGPU deploys a mitigation mechanism implemented in the warp scheduler that disrupts the resonance activity pattern. EmerGPU has no impact on performance and a small power cost. Reducing voltage noise improves system reliability and allows for smaller voltage margins to be used, reducing overall energy consumption by an average of 21%.**

## I. Introduction

General-purpose graphics processing units (GPGPUs) are emerging as high-performance and energy-efficient alternatives to CPU-based computing. However, just like in the case of CPUs, power consumption is becoming a significant roadblock to the scaling of GPU performance. Lowering the chip supply voltage ($V_{\text{dd}}$) is one of the most effective techniques for improving energy efficiency. Unfortunately, supply voltage scaling is limited by several technological challenges. One of these challenges is voltage noise caused by abrupt and/or periodic fluctuations in power demand. These fluctuations are most often caused by variation in chip activity levels as a function of workload. If the voltage deviates too much from its nominal value, it can lead to so-called "voltage emergencies," which can cause timing and memory retention errors. To prevent these emergencies, chip designers add voltage margins that in modern processors can be as high as 20% [12], [17], [19], [27] and are expected to increase in the future, leading to higher power consumption than is desirable.

Previous work [7], [8], [9], [10], [15], [25], [26], [27] has proposed several hardware and software mechanisms for reducing the slope of current changes ($\delta I/\delta t$), which reduces voltage noise. This allows the use of smaller voltage guardbands, saving substantial amounts of power. Most previous work, however, has focused on single-core CPUs [7], [8], [10], [15], [25], [26] or low core-count systems [7], [16], [27]. Other previous work [22] has shown that, as the number of cores in future CMPs increases, the effects of chip-wide activity variation overshadow the effects of within-core workload variability. Voltage noise has been measured on production IBM systems by Bertran et al. [2] and on commercial GPUs by Leng et al. [17]. Other work has analyzed the issue of voltage noise [19] with new dedicated modeling tools [20] and proposed mitigation techniques tailored to GPU noise characteristics.

This paper focuses on characterizing voltage noise at resonance frequencies in GPU architectures. Resonance noise is caused by periodic fluctuations in curent demand that occur at certain frequencies. Resonance noise can be especially damaging to the chip's reliability because it generally leads to a pattern of very large and repeated voltage droops. This behavior has been documented in general-purpose CPUs by prior work [7], [25]. To our knowledge this is the first work to specifically target resonance noise in GPUs. A detailed model of the power delivery and distribution system for a GPU was developed for this purpose. The model captures GPU activity at functional unit granularity and generates spatial and temporal distributions of supply voltage. Using this model we simulate a range of workloads, from benchmark applications to microbenchmarks purposefully designed to induce resonant noise.

To address the chalenge of resonance noise we developed EmerGPU, a framework implemented in each of the GPUs streaming multiprocessors (SMs). The framework includes a resonance-aware warp-level scheduler, designed to anticipate workload activity patterns that could lead to voltage noise. When such patterns are encountered, the scheduler disrupts

This work was supported in part by the Defense Advanced Research Projects Agency under the PERFECT (DARPA-BAA-12-24) program and the National Science Foundation under grant CCF-1253933.

them by changing the order in which warps are selected or, if needed, by injecting "dummy" active warps in the SM pipeline for short periods of time. This solution has little impact on performance and a small power cost. Reducing voltage noise improves system reliability and requires smaller voltage guardbands, reducing overall energy consumption by an average of 21%.

Overall, this paper makes the following contributions:

- Characterizes resonance voltage noise in GPUs.
- Proposes a novel resonance-aware warp scheduling solution designed to eliminate resonance noise.

The rest of this paper is organized as follows: Section II describes the modeling infrastructure developed for this work. Section III characterizes resonance-induced voltage noise in GPU systems. Section IV details the EmerGPU design and implementation. Sections V and VI present an experimental evaluation. Section VII details related work and Section VIII concludes.

## II. Voltage Noise Modeling Infrastructure

Understanding voltage noise in GPUs requires detailed modeling of activity, power consumption and power delivery infrastructure.

### A. GPU Architecture

We model a GPGPU architecture with 128 SIMD lanes (4 SMs, each with 32 compute lanes) similar in size to Nvidia's GeForce 440 GPU. The SM design and floorplan is inspired by NVIDIA Fermi [6].

Each SM contains an L1 cache, read-only constant and texture caches, and a software-managed shared memory. There are 32 standard integer/floating-point ALUs (i.e. SPU) and 8 special functional units. To handle transfer of data between SIMD datapath and different memory modules, the memory subsystem is capable of handling 16 separate load-store requests. All the caches inside the SM are backed up by two shared L2 banks. The front-end contains instruction fetch (fetch unit plus instruction cache), a highly-banked register file with an associated operand-collector logic (for read/write arbitration and management), a thread scheduling and branching unit, instruction decode, and instruction issue and scoreboarding logic. Table I lists configuration details for this architecture.

### B. Power Delivery Network Model

Voltage noise is a characteristic of the chip's power delivery subsystem. Capturing its effects requires a detailed model of the relevant components of this subsystem. A chip's power delivery infrastructure consists of both off-chip and on-chip components. The off-chip network includes a voltage regulator, capacitors used to stabilize supply voltage and wires. Our off-chip power delivery model follows the design layout and component characteristics of the Intel Pentium 4 packaging used in prior work [4], [7], [19], [20]. This is one of the few commercial processors for which such data has been published. Our GPU die size and thermal characteristics are similar to that of the Pentium 4 and therefore we expect the component parameters to be a good match. These values are summarized in Table II. The circuit layout is shown in Figure 1. The impedance of this distributed model is characterized and noted to be very close to those obtained in previous work [4], [7] for similar chip characteristics.

On the chip, power is delivered through a set of pins and C4 pads. These connect to a network of wires that deliver the required voltage to the various chip components. We model the on-chip power grid using a distributed RLC network similar to those used by prior work [7], [30]. Figure 2 shows

| GPU Spec | |
|---|---|
| # SMs | 4 |
| L2 Cache | 2 Banks, 128KB/bank (64/128/16) |
| DRAM-C | FR-FCFS, 2 MCs, 2 banks/MC, each 8 byte/cycle |
| SM Freq. | 1440 MHz |
| **SM Spec** | |
| Core | 1440 MHz, in-order pipeline |
| #SIMD lanes | SPU/SFU/LSU : 32/8/16 |
| Text-Cache | 12KB (4/128/24) |
| Const-Cache | 8KB (64/64/2) |
| Inst-Cache | 2KB (4/128/4) |
| L1 Data Cache | 16KB (32/128/4) |
| Shared memory | 48 KB , 32 banks |
| Scheduling | GTO scheduler from [28] |
| Front-end | Max IPC=1, memory coalescing |
| Branch Divergence | Post-Dominator (PDOM) Stack [5] |
| Max Threads / CTA | 1024 |
| Max Threads / SM | 1536 |
| Max CTAs / SM | 8 |
| Max Warps / SM | 48 |
| Reg-file | 32684 $\times$ 32-bit registers, 16 banks |
| Regs / thread | 63 |

TABLE I: Details of the GPU architecture we model (cache structures are described in nsets/bsize/assoc).

| **Resistance** | | **Inductance** | | **Capacitance** | |
|---|---|---|---|---|---|
| $R_{pcb}$ | $94\mu\Omega$ | $L_{pcb}$ | $21pH$ | $C_{pcb}$ | $240\mu F$ |
| $R_{pcb_p}$ | $166\mu\Omega$ | $L_{pcb_p}$ | $19.536\mu H$ | $C_{pkg}$ | $26\mu F$ |
| $R_{pkg}$ | $1m\Omega$ | $L_{pkg}$ | $120pH$ | $C_{onDie}$ | $335nF$ |
| $R_{pkg_p}$ | $541.5\mu\Omega$ | $L_{pkg_p}$ | $5.61pH$ | | |
| $R_{grid}$ | $50m\Omega$ | $L_{grid}$ | $5.6fH$ | | |

TABLE II: RLC component values.

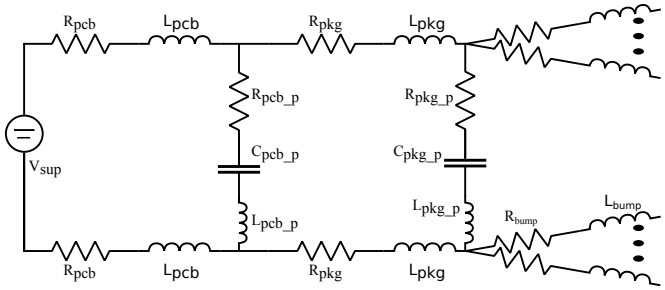


Fig. 1: Off-chip component of the power delivery network.

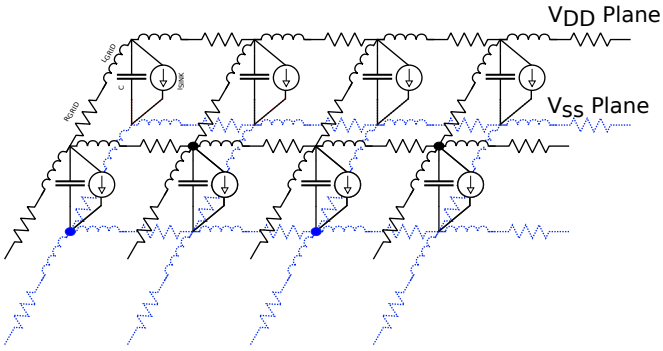


Fig. 2: Distributed model of the on-chip power delivery network.

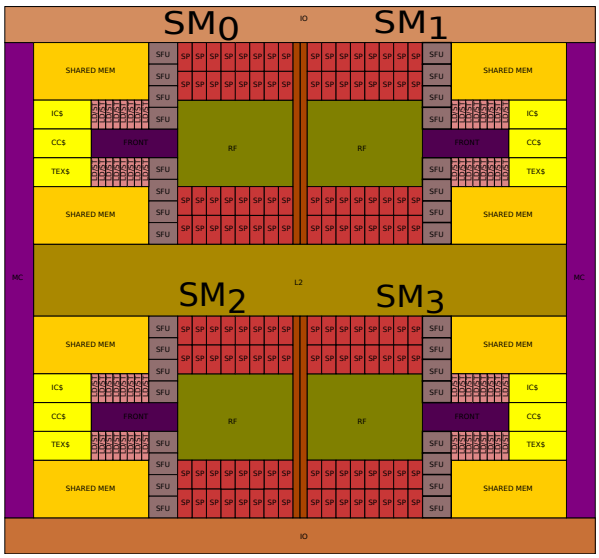


Fig. 3: Floorplan of the simulated GPU architecture.

the circuit layout of the network. Wiring is modeled as an RL network with two planes – one for the $V_{dd}$ and one for the $V_{ss}$ – connected by capacitors. Current sinks across the capacitors are used to model the current drawn by the various functional units, as in work by Herrell and Beker [11]. C4 bumps are placed uniformly throughout the entire chip.

The power delivery model relies on an RLC netlist that is die and package specific. Generating this netlist and assigning various parameters requires a detailed floorplan of the GPU. We base our floorplan on the Fermi die using publicly-available information about component locations and relative sizes. Some design choices are based on educated guesses. Figure 3 shows the floorplan of the GPGPU that we used in our simulations. There are 4 SMs along with a L2 in the middle, memory control blocks along left and right sides and IO blocks on top and bottom. Each SM consists of Shared Memory, Instruction Cache, Constant Cache, Texture Cache, Load Store units, SP units, Special Function units, Register File and Front End.

The inputs to the model consist of current traces at cycle granularity for each functional unit of the GPU. The model output is a trace of on-chip $V_{dd}$ distributions at cycle granularity. A circuit simulator such as SPICE can be used to resolve the power delivery network for each set of inputs. However, because the network is large and it includes many inductive elements, a traditional circuit solver like SPICE would be prohibitively expensive, especially given the need to simulate full benchmarks at cycle granularity. As an alternative, we use a specialized RLC solver based on the preconditioned Krylov-subspace iterative method that we developed based on models by Chen and Chen [3]. Our implementation is orders of magnitude faster than SPICE, allowing the simulation of longer benchmarks. We validate our optimized solver against SPICE simulations.

## III. Resonance Noise in GPUs

Voltage stability is primarily influenced by changes in chip activity dictated by workload behavior. Variation in activity leads to changes in the amount of curent drawn by various functional blocks across the chip. Large and/or rapid changes in current demand ($\delta I/\delta t$) cause the supply voltage ($V_{dd}$) to droop below nominal values.

### A. Voltage Noise at Resonant Frequencies

In addition to the magnitude of the $\delta I/\delta t$, the pattern of the current changes plays a crucial role in $V_{dd}$ stability. In particular, voltage droops can be significantly amplified by periodic changes in $\delta I/\delta t$ that occur at certain frequencies. $V_{dd}$ is a function of both current demand and the chip's impedance($Z$), as follows: $\delta V = \delta I * Z$. Although the ideal $Z$ would be flat across its frequencies of operation, a number of design constraints make that ideal hard to achieve. In reality, the impedance of a chip varies significantly with the frequency of $\delta I/\delta t$ changes. Figure 4 shows the impedance as a function of frequency for the GPU chip we model. We can see that impedance varies by over 10× across the spectrum of frequencies we test (1MHz-1GHz). This means that the same $\delta I/\delta t$ events that occurs with a frequency corresponding to a high $Z$ will lead to much bigger droops than events that occur with frequencies corresponding to low $Z$s. For this chip, a peak is observed around 100 MHz indicating we can expect higher droops for $\delta I/\delta t$ events occurring at (or close to) that frequency – generally refered-to as a "resonance frequency." This property has been well documented by prior work on CPUs [2], [4], [7].

### B. GPU Voltage Noise Characterization

*1) CUDA Resonance Microkernel:* In order to characterize voltage noise behavior in GPUs, we constructed microkernels designed to exhibit workload oscillation at (or around) the resonance frequency. An example of such a microkernel (in NVidia PTX assembly) is shown in Figure 5. The kernel is composed of sequence of floating-point additions (e.g.

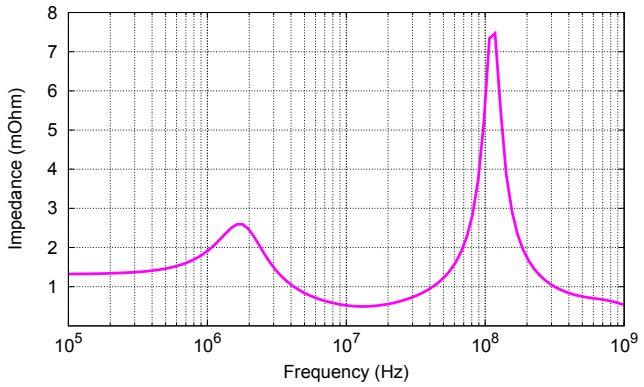


Fig. 4: Impedance of the GPU power delivery network versus the frequency of activity change.

```
...

add.f32          %f10, %f1, %f2;
sin.approx.f32   %f21, %f1
add.f32          %f11, %f1, %f3
cos.approx.f32   %f22, %f2
add.f32          %f12, %f1, %f4
sin.approx.f32   %f23, %f3
add.f32          %f13, %f2, %f3
cos.approx.f32   %f24, %f4
add.f32          %f14, %f2, %f4
sin.approx.f32   %f25, %f5
add.f32          %f15, %f2, %f5
cos.approx.f32   %f26, %f6

...
```

Fig. 5: PTX micro-kernel demonstrating the activity pattern to stress the resonance frequency.

"add.f32") and approximate transcendental instructions (e.g. "sin.approx.f32"). All the FP additions are executed by normal ALUs while the sin/cos instructions will be issued to special function units (SFUs). The kernel is carefully tuned to alternate between high and low power phases. To maximize power consumption and control the frequency of oscillation, the kernel avoids unnecessary register-spilling, it includes no RAW dependences between any two SP/SFU instructions (avoiding stalls from the scoreboard logic), and avoids write-back conflicts. The input operands of the instructions are chosen to provide maximum freedom for the instruction issue logic. The appropriate kernel dimensions are found after trying several iterations of the same sequence and considering the amount of SM resources used by each thread.

Using the frequency vs. impedance experiment illustrated in Figure 4 we identify the resonance frequency that generates the highest impedance, which is about 100MHz. We tune the rate of activity changes in the microkernel to match

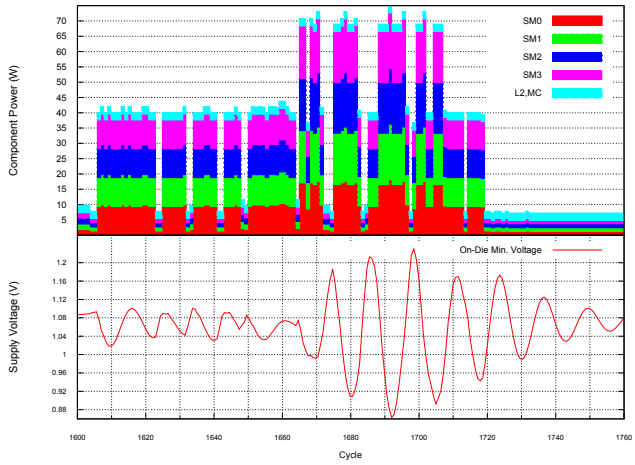


Fig. 6: Power consumption of the resonance microkernel running on the GPU and the effects on supply voltage noise.

this frequency. We run the microkernel trace through our power delivery simulator and measure supply voltage across the die. Note that the chip frequency is 1.4GHz so the period of the workload oscillation is 14 GPU cycles. Figure 6 shows the power trace of the resonant microkernel running on four SMs of the GPU and the supply voltage measured on the die location with the worst droop.

The microkernel activates both SP and SFU units simultaneously around cycle 1660. This raises chip power to close to 70W for a few cycles. The power drops for a brief period of time followed by another parallel activation of both SP and SFU units in each SM. This pattern repeats at a frequency that is close to the resonance frequency of the die. This triggers large swings of the supply voltage with the largest droops exceeding 220mV, representing 20% of the nominal $V_{\text{dd}}$ of 1.1V. This shows that certain execution patterns could indeed lead to workload fluctuation at the resonance frequency, which in turn can cause catastrophic voltage noise.

*2) Worst-Case Chip-Wide Resonance:* We also generated a worse case scenario under which all units on chip switch from leakage power to maximum power, fully synchronized at the resonance frequency. Figure 7 shows the results of this experiment. The top chart shows the oscillations in chip power consumption over time. The low-to-high and high-to-low power transitions occur over a single GPU cycle, which generates the highest $\delta I/\delta t$ that the chip could theoretically experience. The bottom chart in the figure shows the supply voltage. We can see that, as the power oscillates at the resonance frequency, the amplitude of the voltage fluctuations increases rapidly, with voltage droops reaching 280mV, or about 25% of the nominal voltage.

We repeat the experiment with the same artificial trace oscillating at different frequencies. Figure 8 shows the results for a trace oscillating at 240MHz, which is safely outside the resonance region. In this case, even through the $\delta I/\delta t$ of

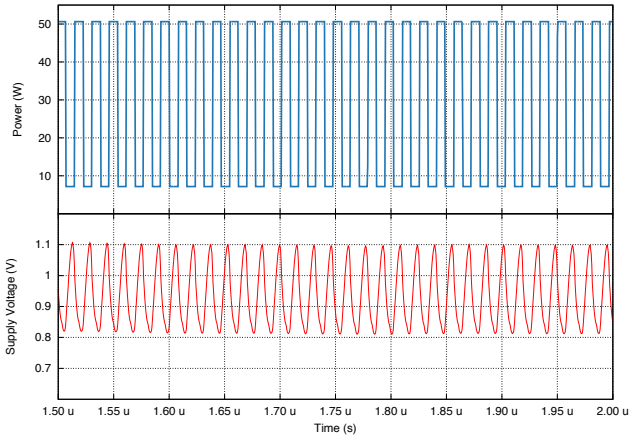


Fig. 7: Workload oscillation at resonance frequency and its effects on supply voltage.

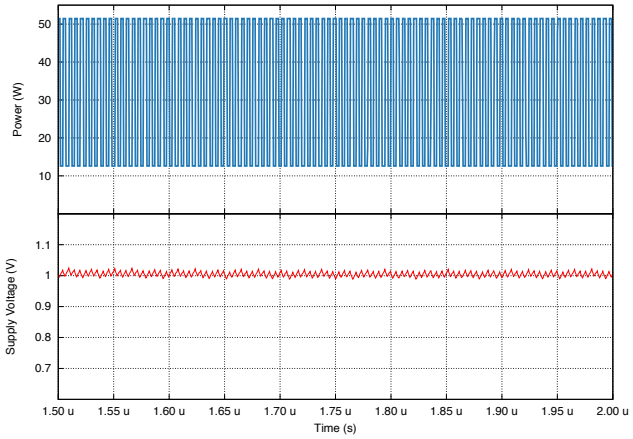


Fig. 8: Workload oscillation outside the resonance frequency and its effects on supply voltage.

this workload is identical to the resonance case, the voltage droops are significantly lower at around 100mV or less than 10% of the nominal $V_{\text{dd}}$. This shows that resonance noise, when present in the workload, dwarfs the effects of even the worst non-resonating $\delta I/\delta t$.

## IV. Mitigating Resonance Noise in GPUs

To mitigate noise effects we designed EmerGPU, a simple and robust mechanism implemented at SM level. Our solution detects activity patterns that oscillate at the resonance frequency of the chip. When such activity is encountered, EmerGPU disrupts the resonant activity through careful rescheduling of warps and artificial load injection.

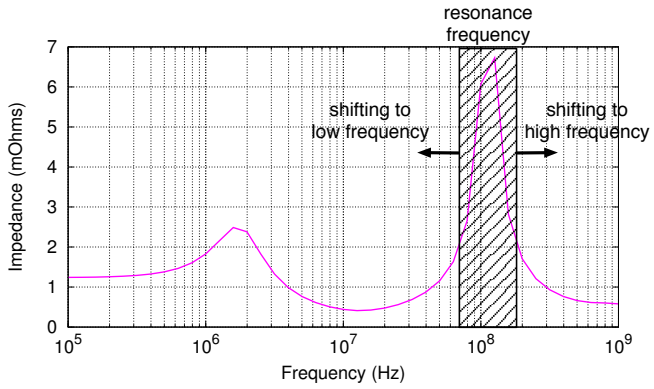


Fig. 9: Eliminating resonance by shifting the workload oscillation to either lower or higher frequencies.

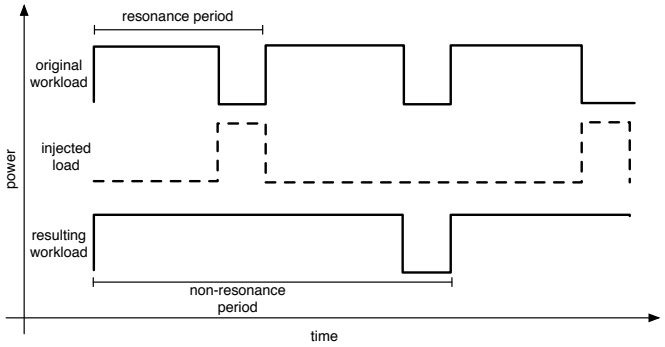


Fig. 10: Eliminating resonance by shifting the workloads with high duty cycles to low frequency through instruction scheduling or load injection.

### A. Resonance Disruption Through Frequency Shifting

EmerGPU shifts the workload oscillation pattern to frequencies that are either higher or lower than the damaging resonance frequencies. The desired effect of our resonance mitigation solution is illustrated in Figure 9.

The choice of whether to shift a potentially resonant activity pattern towards a higher or a lower frequency is made based on the duty cycle observed in the workload. If the duty cycle of the resonant signal is high, i.e. the power is high for most of the cycle period, the signal is shifted towards lower frequencies. Figure 10 shows an example of how this is accomplished. The process involves altering the workload such that some low-power instructions are replaced by high power ones. This can be accomplished by the warp scheduler. If no high power warps/instructions are available, artificial load is injected. The resulting signal has longer periods corresponding to frequencies that are lower that the resonance region.

If, on the other hand, the duty cycle of the resonant signal is low, i.e. the power is low for most of the cycle period, the signal is shifted towards a higher frequency, as

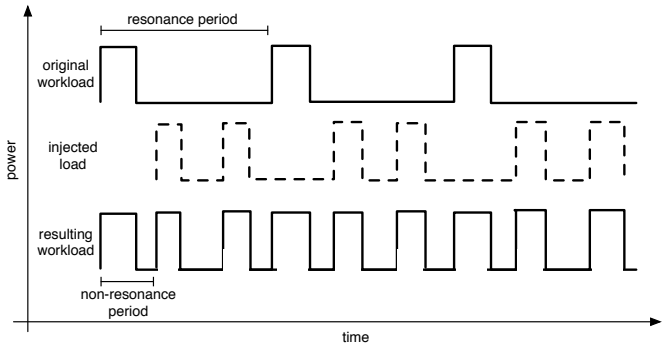


Fig. 11: Eliminating resonance by shifting the workloads with low duty cycles to high frequency.

shown in Figure 11. This is done to minimize the power overhead introduced by the injected load. Since extending the low duty-cycle signal to longer non-resonant periods would require significant amounts of injected power, we opt instead for shorter periods which require less power. The process involves scheduling or injecting a high frequency sequence of instructions into the original workload, as illustrated by the dotted line in Figure 11. The resulting signal of the new workload has a frequency of oscillation that is higher than the resonance frequency.

### *B. EmerGPU Design and Implementation*

EmerGPU is implemented in the issue and execute stages of each SM of the GPU. Figure 12 highlights the microarchitectural modifications/additions made to the SM pipeline. EmerGPU consists primarily of two components: the Resonance Monitor and the Resonance-Aware Warp Scheduler.

*1) Resonance Monitor:* The Resonance Monitor is a hardware unit that looks for sequences of instructions that oscillate between high and low power at frequencies close to resonance. To detect these cases, the unit examines the active mask of the warp scheduled for execution in the current cycle and first classifies the warp as *high* or *low* power. A warp is considered high-power when its active lane count is above a certain threshold, and low-power otherwise. If the warp is determined to be high-power, the current cycle is marked as a *high-activity* cycle.

When execution transitions from a low-activity to a high-activity phase, a period counter is started. If the next low-to-high transition occurs within a number of cycles that is lower than the resonance period of the chip, the resonance monitor enteres "resonance mitigation mode" and stays in this mode as long as the oscillation period is within the resonance region. A duty cycle counter records the ratio of high vs. low activity cycles. Depending on the duty-cycle, the resonance monitor chooses either *high-frequency* or *low-frequency* mitigation.

In high-frequency mode (which corresponds to a low duty-cycle), the monitor disrupts the resonant activity by

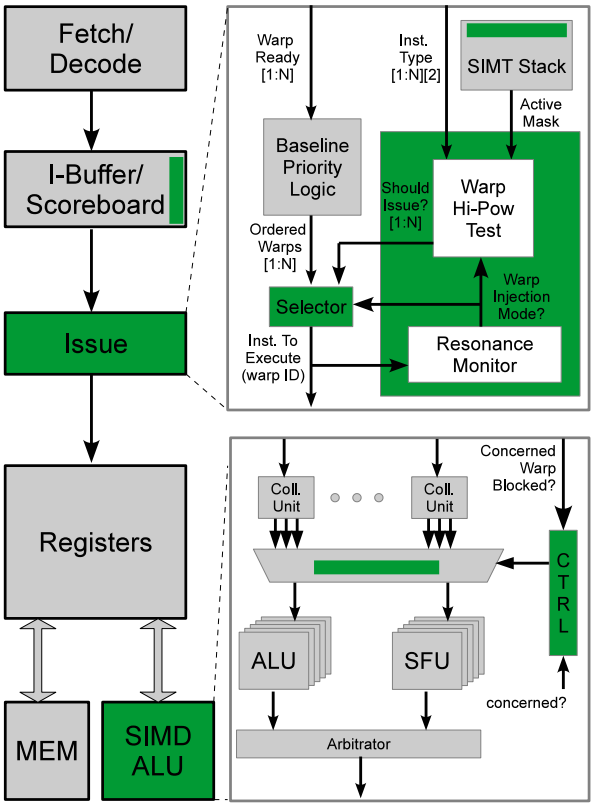


Fig. 12: EmerGPU changes to the baseline SM microarchitecture (highlighted in dark green).

shifting it towards higher-frequencies as in Figure 11. This is accomplished by raising a *warp injection* flag in short bursts of only a few cycles. In low-frequency mode, the monitor raises the *warp injection* signal immediately in order to shift the activity pattern to a lower-frequency as in Figure 10.

*2) Warp Power Classifier:* When the resonance-monitor detects a region of the workload where warp injection is needed to disrupt resonance, a Warp Classifier unit is activated to identify high power instructions/warps in the list of available instructions. The classifier relies on the active mask (coming from the SIMT Stack) and the instruction type (from the instruction buffer) to make this determination. If the instruction is of compute type (ALU or SFU) and the number of active lanes is greater than a threshold (e.g. 32 lanes for ALU or 8 lanes for SFU), the instruction is considered high power and can be used to disrupt resonance. If the instruction is of memory type (load, store, mem-barrier, etc), it is classified as low power.

Given N warps in the list of active warps, the Warp Classifier generates an N-bit vector where element $k$ determines if the $k^{th}$ warp is high (1) or low-power (0). This mask will be used by the selection logic to select high-power warps when the SM is in "resonance mitigation mode."

*3) Resonance-Aware Warp Selection:* A Warp Selector unit is responsable for managing the warp injection process when the "resonance mitigation mode" is active. If the *warp injection* flag is set, the Warp Selector looks for a candidate in the sub-list of high-power warps. If one is found, it is immediately dispatched for execution. Otherwise, the Warp Selector examines the available low-power warps. If a ready low-power warp exists, a *power boost* flag is set for that warp and it is then dispatched for execution. The *power boost* flag will signal to the execution unit to boost its power consumption (e.g. by activating the lanes that are masked off) when this warp arrives for execution. The *warp injection* flag is attached to all warps dispatched when that mode is active. This will indicate to the execution unit to take additional measures if these warps are stalled due to dependencies.

If the Warp Selector cannot find any warps that are ready for dispatch and the *warp injection* flag is still set, the selector will choose a "dummy" warp which contains special high power NOP instructions. These instructions have no effect on the execution but are designed to activate functional units to bring the power consumption up as requited during warp injection cycles.

*4) Resonance-Aware Warp Execution:* At the execute stage of the pipeline EmerGPU examines the flags set at dispatch. If a warp is marked with *power boost*, indicating it may have insufficient power, the clock gating signal of inactive lanes is canceled in order to increase power consumption. If the warp is marked as *dummy*, clock gating is also disabled for all lanes.

If an instruction with the *warp injection* flag set is stalled at register access (e.g. due to a bank conflict), clock gating will be disabled for the current cycle as a substitute for the stalled warp.

### C. Hardware Overhead

EmerGPU requires few modifications to the pipeline microarchitecture. In the instruction buffer, each instruction will have an additional 2-bit entry to specify its type (0=mem, 1=ALU, 2=SFU). For "dummy" warps, we dedicate a single-entry stack (no divergence, so one entry is sufficient) which contains the active mask and PC. The Resonance Monitor requires one counter for the period and two counters to keep track of high- and low-power phases of activity, plus compare logic to detect the power level of a given mask (by counting the number of 1s). The Power Classifier unit requires a compare (bit-count) logic for warp power classification. The Warp Selector logic only needs logic to check warp readiness and power levels.

## V. Methodology

Our modeling framework includes a GPU simulator used to generate activity and power traces and a detailed power delivery model, coupled with the simulator, that models the runtime $V_{\text{dd}}$ behavior as a function of workload. The Figure 13 shows an overview of our evaluation infrastructure.

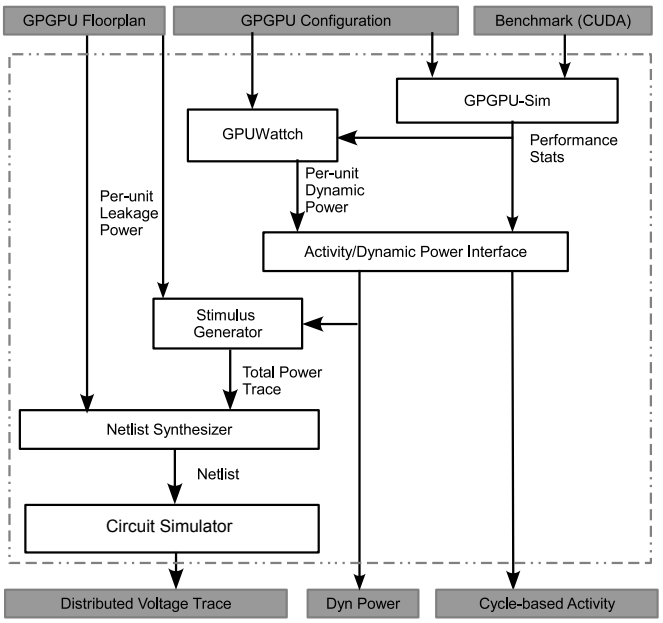


Fig. 13: Overview diagram of the evaluation infrastructure.

| Application | Abbr. | #Insts |
|---|---|---|
| AES Cryptography | AES | 28M |
| BFS Graph Traversal | BFS | 17M |
| Coulombic Potential | CP | 126M |
| LIBOR Monte Carlo | LIB | 907M |
| 3D Laplace Solver | LPS | 82M |
| MUMmerGPU | MUM | 77M |
| Neural Network | NN | 68M |
| N-Queens Solver | NQU | 2M |
| Ray tracing | RAY | 71M |
| StoreGPU | STO | 134M |
| Weather Prediction | WP | 215M |

TABLE III: Benchmarks used in the evaluation.

CUDA benchmarks are compiled with NVIDIA C Compiler (nvcc) version 4.0. The generated binary is run on GPGPU-Sim (version 3.2.1 [1]) that simulates our baseline architecture. As an output, the simulator is modified to generate a detailed cycle-accurate trace of activity for different SM- and GPU-level components.

To estimate per-unit dynamic and leakage power consumption, we use GPUWattch [18] (with a modified McPAT [21]) for functional units and other pipeline internals (e.g. memory coalescer). For the memory modules (e.g. caches and register file), we use CACTI version 6.0 [23]. A total power trace is generated and converted to piece-wise linear current trace that is used as an input for the chip power delivery model detailed in Section II.

Table III shows the set of benchmarks we use to evaluate EmerGPU. These represent a diverse set of CUDA applications, broadly chosen from different suites (Rodinia, Parboil and NVIDIA Compute SDK).

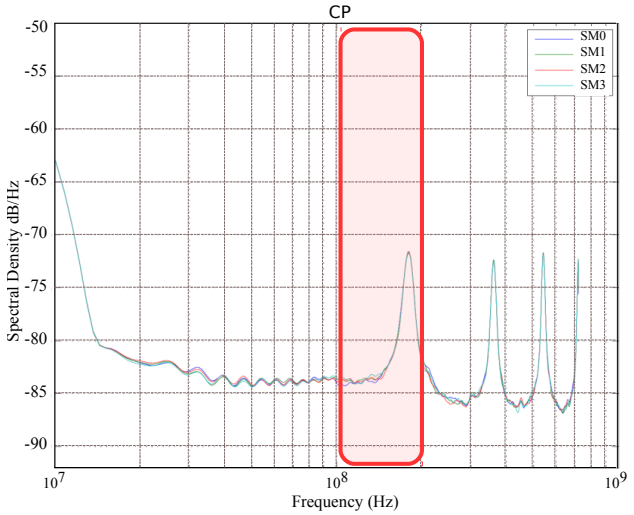


Fig. 14: Power spectral density analysis for baseline *CP*.

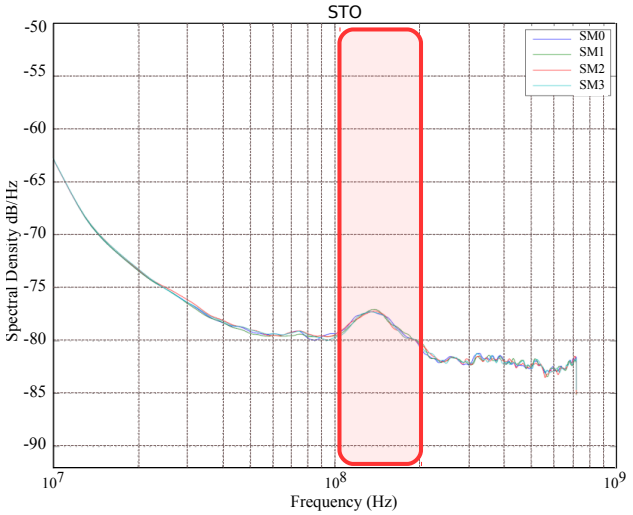


Fig. 15: Power spectral density analysis for baseline *STO*.

## VI. Evaluation

In this section we characterize resonance noise in GPU benchmarks and evaluate the effectiveness of EmerGPU in reducing that noise. We also present the power and energy savings derived from the reduction in voltage margins enabled by the noise mitigation.

### A. Resonance Noise in GPU Benchmarks

Isolating resonance noise from other effects in real benchmarks is challenging because voltage fluctuations are caused by $\delta I/\delta t$ oscillations at a broad range of frequencies. While resonance noise does dominate other effects when present in the workload, it effects can be hard to visualize. In order to isolate resonance noise effects in our benchmarks we run a spectral density analysis on the power traces to measure energy in the resonance range. The power spectral density is obtained by running a fast Fourier transform (FFT) on the power trace in order to isolate the energy in each sub-frequency component. High energy in the resonance frequencies indicates that the benchmark will experience high resonance noise. Figure 14 shows the results of the power spectral density analysis for benchmark *CP*. The figure plots power spectral density in dB/Hz for each frequency subcomponent ranging from 10MHz to 1GHz.

Recall the resonance frequency for our chip is around 100MHz (area highlighted in red on the plot). As we can see from Figure 14, benchmark *CP* exhibits very high energy at the top end of its resonance band relative to other frequencies. This indicates that *CP* experiences significant resonance-induced noise.

Other benchmarks such as *STO* also exhibit high energy in the resonance region (Figure 15), although not quite as high as *CP*. Not all benchmarks however display this characteristic. For instance, *BFS* reveals a relatively flat

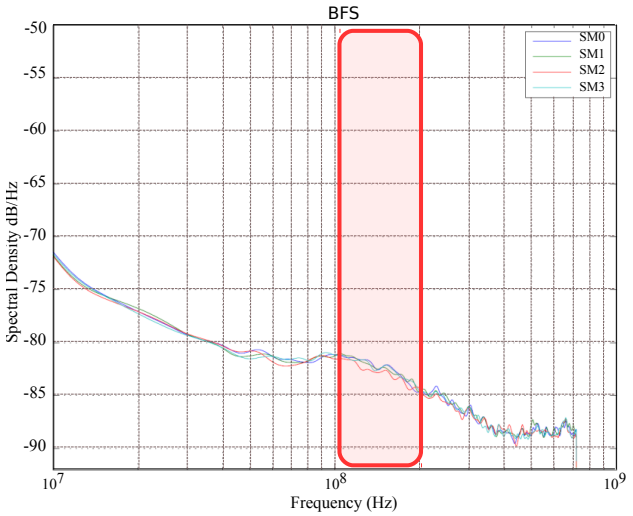


Fig. 16: Power spectral density analysis for baseline *BFS*.

energy distribution across its frequency spectrum (Figure 16). This makes it less likely that it will suffer from resonance induced noise.

Figure 17 shows an example of a resonating section of benchmark *AES*. The top chart shows the supply voltage in the baseline system, with the resonating section highlighted. The bottom chart shows the supply voltage for the same benchmark section running on the EmerGPU system. We can see that resonance is completely eliminated.

### B. Resonance Mitigation with EmerGPU

The goal of EmerGPU is to disrupt resonance frequencies by shifting their energy to either low or high frequencies. To measure how effective EmerGPU is at achieving this goal we again use spectral density analysis. Figure 18 shows the

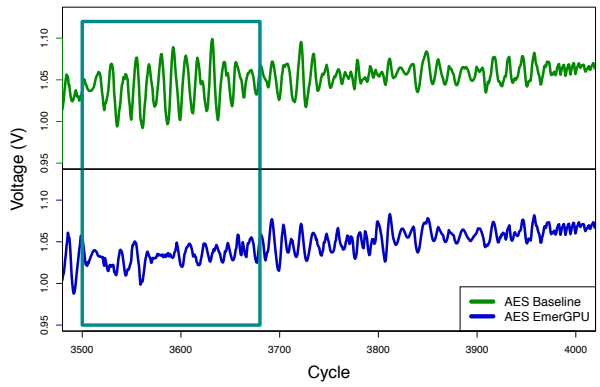


Fig. 17: Example of resonance noise in AES baseline (top) and the mitigation effects of EmerGPU (bottom).

| Application | Energy Reduction |
|---|---|
| RAY | −3dB |
| NQU | −12dB |
| NN | −7dB |
| WP | −5dB |
| STO | −4dB |
| MUM | −3dB |
| LPS | −4dB |
| CP | −13dB |
| BFS | −3dB |
| AES | −7dB |
| LIB | −10dB |

TABLE IV: Reduction in energy at resonance frequencies with EmerGPU.

power spectral density for *CP* running on the baseline system (a) and with EmerGPU (b). We can see that EmerGPU is very effective at removing the energy peak at resonance. The reduction in energy in that region is about -13dB or about 20× lower energy relative to the baseline system. This effectively eliminates resonance noise from *CP*.

Table IV summarizes the reduction in energy in the resonance frequencies for all the benchmarks. Energy reduction varies greatly from benchmark to benchmark but is at least 50% (-3dB) and can be as high as 20× as in the case of *CP*. Some benchmarks experience a smaller energy reduction because they don't have much energy in the resonance region.

### C. Voltage Margin Reduction

EmerGPU virtually eliminates activity-induced resonance noise. This allows a substantial reduction of the large voltage margins that would otherwise be required to ensure reliable execution. To measure the reduction in voltage margins we measure maximum voltage droop for the worst case resonance (load oscillation at 100MHz, 50% duty cycle). This is the worse-case guardband that would be required for this system. We determine this guardband to be 280mV. This is consistent with prior work [12], [27]. We run the same experiment with EmerGPU and measure the worst-case droop. We determine the guardband for EmerGPU to be 120mV. As a result, EmerGPU reduces the guardband required for this chip by a very significant 160mV.

### D. Power Savings

The margin reduction enabled by EmerGPU leads to substantial power savings. Figure 19 shows the power for each benchmark relative to the baseline system. The top bar in the stack (in red) represents the additional power introduced by EmerGPU in *warp injection* mode. It accounts for the hardware overhead, the cost of "dummy" warp injection, and the boosting of warps with insufficient power. This overhead averages 8% over the baseline. However, the reduction in voltage margins enabled by EmerGPU more that makes up for the power overhead. Overall EmerGPU reduces power consumption by 21% on average.

EmerGPU has virtually no performance overhead. Since it injects "dummy" warps when the pipeline is stalled, performance is not impacted. EmerGPU does occasionally change the warp execution order set by the baseline scheduler. This results in marginal slowdowns for some applications and actually speeds up others. On average EmerGPU's performance is within less than 0.25% of the baseline.

## VII. Related Work

Extensive prior research exists on the topic of voltage noise in CPUs. Reddi et al. [26] employ heuristics and a learning mechanism to predict voltage emergencies from architectural events. When an emergency is predicted, execution is throttled, reducing the slope of current changes. Gupta et al. [10] proposed an event guided adaptive voltage emergency avoidance scheme. Recurring emergencies are avoided by initiating various operations such as *pseudo-nops*, prefetching, and a hardware throttling mechanism on events that cause emergencies. Gupta et al. also proposed DeCoR [8], a checkpoint/rollback solution that allows voltage emergencies but delays the commit of instructions until they are considered safe. A low voltage sensor, of known delay, signals that an emergency is likely to have occurred and the pipeline is flushed and rolled back to a safe state.

Powell and Vijaykumar [25] proposed two approaches for reducing high-frequency inductive noise caused by processor pipeline activity. Pipeline muffling reduces the number of functional units switching at any given time by controlling instruction issue. A priori current ramping slowly ramps up the current of functional units before they are utilized in order to reduce the amplitude of the current surge. A software approach to mitigating voltage emergencies was proposed by Gupta et al. in [9]. They observe that a few loops in SPEC benchmarks are responsible for the majority of emergencies in superscalar processors. Their solution involves a set of compiler-based optimizations that reduce or eliminate architectural events likely to lead to emergencies such as cache or TLB misses and other long-latency stalls.

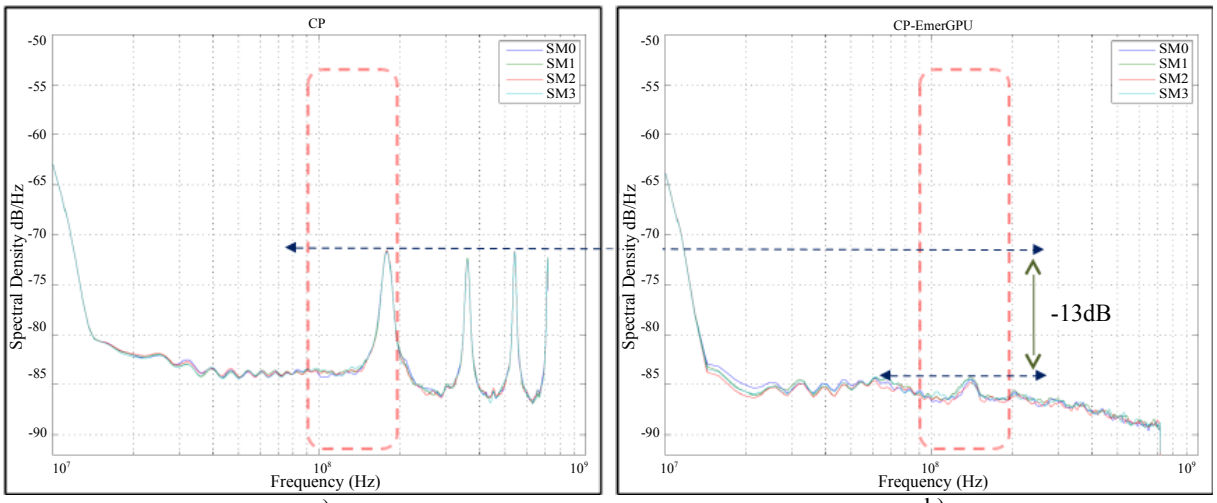


Fig. 18: Power spectral density analysis for *CP* running on the baseline system (a) and with EmerGPU (b).

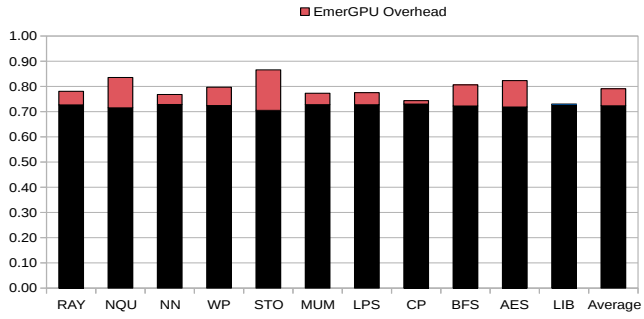


Fig. 19: Power consumption with EmerGPU for each benchmark relative to the baseline.

A few previous studies have examined voltage emergencies in multicore chips. Gupta et al. [7] characterize within-die voltage variation using a detailed distributed model of the on-chip power-supply grid. Reddi et al. [27] evaluate voltage droops in a commercial dual-core microprocessor. They propose co-scheduling threads with complementary noise behavior, to reduce voltage droops. Miller et al. examine synchronization-induced voltage emergencies in CMPs with large number of cores [22]. They find that, as the number of cores increases, the effects of chip-wide activity variation such as that introduced by barrier synchronization leads to high $\delta I/\delta t$ and voltage emergencies. They propose changes to synchronization libraries that reduce $\delta I/\delta t$ to avoid emergencies. Recent work by Kim et al. [16] presents a rigorous testing framework for generating kernels that stress the power delivery network and cause worse-case voltage noise. They test and evaluate their framework on real hardware.

Voltage noise in GPUs has received relatively little attention. Leng et al. [19], [20] studied available margins in modern GPUs and also characterized the causes of typical $\delta I/\delta t$ events in GPU workloads. They present solutions for controlling coordinated activity across SMs. They also evaluate the effect of the large Register File present in GPUs on voltage noise. They propose methods of disrupting these patterns of activity across SMs and within an SM to mitigate voltage noise. Recent work by Thomas et al. [29] proposed the joint mitigation of voltage noise and process variation using a technique called "Core Tunneling" that clock gates individual SMs when voltage drops below a safe threshold.

To the best of our knowledge this is the first work to focus on characterizing and mitigating resonance noise in GPUs.

Various performance-centric warp scheduling techniques for GPUs have been developed in prior work. Their goal is generally to more efficiently hide the latency of the memory subsystem. Examples include the two-level warp scheduler for improving memory latency developed by Narasiman et al. [24]), cache-conscious wavefront scheduling for cache-sensitive applications by Rogers et al. [28], warp scheduling for reducing cache or DRAM contention by Jog et al. [13], or scheduling techniques to improve efficiency of the prefetchers by Jog et al. [14]. Our work employs noise-aware warp scheduling to eliminate resonance noise without impacting performance.

## VIII. Conclusion

This paper characterized resonance noise in a GPU architecture and showed that workload variability at resonant frequencies can lead to very large voltage droops. We proposed EmerGPU, a technique that disrupts resonance activity patterns, reducing voltage droops and allowing lower voltage guardbands. EmerGPU reduces power consumption by an average of 21% with no performance overhead. We hope this work will also serve as a foundation for more research on resonance noise in GPU environments.

## REFERENCES


[1] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2009, pp. 163–174.

[2] R. Bertran, A. Buyuktosunoglu, P. Bose, T. Slegel, G. Salem, S. Carey, R. Rizzolo, and T. Strach, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *International Symposium on Microarchitecture (MICRO)*, December 2014, pp. 368–380.

[3] T.-H. Chen and C. Chen, "Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 559–562.

[4] W. El-Essawy and D. H. Albonesi, "Mitigating inductive noise in SMT processors," in *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2004, pp. 332–337.

[5] W. W. L. Fung, I. Sham, G. Yuan, and T. M. Aamodt, "Dynamic warp formation and scheduling for efficient GPU control flow," in *International Symposium on Microarchitecture (MICRO)*, December 2007, pp. 407–420.

[6] P. N. Glaskowsky, "NVIDIA's Fermi: The first complete GPU computing architecture," September 2009, White Paper.

[7] M. S. Gupta, J. Oatley, R. Joseph, G.-Y. Wei, and D. Brooks, "Understanding voltage variations in chip multiprocessors using a distributed power-delivery network," in *Design Automation and Test in Europe (DATE)*, April 2007, pp. 624–629.

[8] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "DeCoR: A delayed commit and rollback mechanism for handling inductive noise in processors," in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2008, pp. 381–392.

[9] ——, "Towards a software approach to mitigate voltage emergencies," in *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2007, pp. 123–128.

[10] M. S. Gupta, V. Reddi, G. Holloway, G.-Y. Wei, and D. Brooks, "An event-guided approach to reducing voltage noise in processors," in *Design Automation and Test in Europe (DATE)*, April 2009, pp. 160–165.

[11] D. Herrell and B. Beker, "Modeling of power distribution systems for high-performance microprocessors," *IEEE Transactions on Advanced Packaging*, vol. 22, no. 3, pp. 240–248, 1999.

[12] 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)*, February 2007, pp. 298–604.

[13] A. Jog, O. Kayiran, N. Chidambaram Nachiappan, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "OWL: Cooperative thread array aware scheduling techniques for improving GPGPU performance," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013, pp. 395–406.

[14] A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Orchestrated scheduling and prefetching for GPGPUs," in *International Symposium on Computer Architecture (ISCA)*, 2013, pp. 332–343.

[15] R. Joseph, D. Brooks, and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors," in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2003, pp. 79–90.

[16] 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 *International Symposium on Microarchitecture (MICRO)*, December 2012, pp. 212–223.

[17] 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 *International Symposium on Microarchitecture (MICRO)*, 2015, pp. 294–307.

[18] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: Enabling energy optimizations in GPGPUs," in *International Symposium on Computer Architecture (ISCA)*, June 2013.

[19] J. Leng, Y. Zu, and V. Reddi, "GPU voltage noise: Characterization and hierarchical smoothing of spatial and temporal voltage noise interference in GPU architectures," in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2015, pp. 161–173.

[20] J. Leng, Y. Zu, M. Rhu, M. Gupta, and V. J. Reddi, "GPUVolt: Modeling and characterizing voltage noise in GPU architectures," in *International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 2014, pp. 141–146.

[21] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *International Symposium on Microarchitecture (MICRO)*, December 2009, pp. 469–480.

[22] T. N. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: Characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in *International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 249–260.

[23] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A Tool to Model Large Caches," HP Labs, Tech. Rep. HPL-2009-85, 2009.

[24] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU performance via large warps and two-level warp scheduling," in *International Symposium on Microarchitecture (MICRO)*, December 2011, pp. 308–317.

[25] M. D. Powell and T. N. Vijaykumar, "Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise," in *International Symposium on Low Power Electronics and Design (ISLPED)*, August 2003, pp. 223–228.

[26] V. J. Reddi, M. S. Gupta, G. Holloway, G. yeon Wei, M. D. Smith, and D. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2009.

[27] 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 *International Symposium on Microarchitecture (MICRO)*, December 2010, pp. 77–88.

[28] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *International Symposium on Microarchitecture (MICRO)*, 2012, pp. 72–83.

[29] R. Thomas, K. Barber, N. Sedaghati, L. Zhou, and R. Teodorescu, "Core Tunneling: Variation-aware voltage noise mitigation in GPUs," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 1–13.

[30] X. Zhang, T. Tong, S. Kanev, S. K. Lee, G.-Y. Wei, and D. Brooks, "Characterizing and evaluating voltage noise in multi-core near-threshold processors," in *International Symposium on Low Power Electronics and Design (ISLPED)*, 2013, pp. 82–87.