Predictive Guardbanding: Program-driven Timing Margin Reduction for GPUs

Jingwen Leng, *Member, IEEE,* Alper Buyuktosunoglu, *Fellow, IEEE,* Ramon Bertran, *Member, IEEE,* Pradip Bose, *Fellow, IEEE,* Yazhou Zu, *Member, IEEE,* and Vijay Janapa Reddi, *Member, IEEE,*

***Abstract*—Energy efficiency of GPU architectures has emerged as an essential aspect of computer system design. In this paper, we explore the energy benefits of reducing the GPU chip's voltage to the safe limit, i.e., V_{min} point, using predictive software techniques. We perform such a study on several commercial off-the-shelf GPU cards. We find that there exists about 20% voltage guardband on those GPUs spanning two architectural generations, which, if "eliminated"entirely, can result in up to 25% energy savings on one of the studied GPU cards. Our measurement results unveil a program dependent V_{min} behavior across the studied applications, and the exact improvement magnitude depends on the program's available guardband. We make fundamental observations about the program-dependent V_{min} behavior. We experimentally determine that the voltage noise has a more substantial impact on V_{min} compared to the process and temperature variation, and the activities during the kernel execution cause large voltage droops. From these findings, we show how to use kernels' microarchitectural performance counters to predict its V_{min} value accurately. The average and maximum prediction errors are 0.5% and 3%, respectively. The accurate V_{min} prediction opens up new possibilities of a cross-layer dynamic guardbanding scheme for GPUs, in which software predicts and manages the voltage guardband, while the functional correctness is ensured by a hardware safety net mechanism.**

***Index Terms*—Multi-core processors, single instruction and multiple data, GPU, voltage guardband, PVT variation.**

I. Introduction

General-purpose GPU (GPGPU) architectures are already essential mainstream computing elements. Heterogenous systems such as the next two world's fastest supercomputers *Summit* and *Sierra* that couple the throughput optimized GPUs with the latency optimized CPUs provide superior computational horsepower for modern killer workloads [2]. As such, the energy efficiency of GPU becomes critical for our society owing to its enormous impact on the economy because one such supercomputer can consume over 30 MW peak power consumption, and produce a monthly electricity bill of several millions of dollars.

State-of-the-art GPU power-saving efforts strongly reflect and follow the CPU trend. Typical optimizations include power gating and dynamic voltage and frequency scaling (DVFS) [29], [40]. These techniques mainly leverage a processor's supply voltage as a knob to balance the performance and power consumption because the supply voltage directly determines the power consumption [54]. However, none of them addresses the fundamental energy inefficiency that exists at the voltage guardband level.

Typically, designers allocate a large portion of supply voltage, i.e., voltage guardband, to combat the worst-case process, temperature and voltage variation (noise). This design-for-worst-case methodology leads to energy wastage because the chip could have operated at a lower supply voltage most of the time when the worst case condition rarely occurs [13], [17], [27], [46]. In the future, the voltage guardband relative to the nominal voltage is predicted to grow due to increased variations as technology scales [48], which requires us to actively manage the guardband to maximize the energy efficiency.

In this work, we provide a measurement-based study to quantify the energy saving potential of pushing the GPU supply voltage to its safe limit. We achieve so by conducting the V_{min} measurement on several off-the-shelf GPU cards spanning two architectural generations (Fermi and Kepler). At the V_{min} point, a program executes correctly but fails if the supply voltage is reduced any further. We make two key observations from measurement results. First, all studied GPUs have significant margins between the nominal voltage and the V_{min} point. For example, on a GTX 680 card, we can reduce the voltage by up to 18%, which indicates 25% energy saving potential. Second, the variability of the margin among programs is also significant (10%), which necessitates program-specific guardband optimizations.

To safely restore the potential for voltage guardband optimization, we must understand the cause of the program-specific V_{min} phenomenon. To this end, we make two critical contributions. We first determine the cause of the program dependent guardband behavior from the candidates of the process, temperature, and voltage variation, against which the voltage guardband mainly protects. We observe that the voltage noise has the largest impact, and causes the program dependent V_{min} behavior. Because voltage noise depends on program characteristics [16], [47], it also matches the program-dependent V_{min} observation. Second, we identify the critical voltage noise characteristics. We profile each program's performance characteristics, measure its power consumption, and use these measured program-driven metrics to study the interaction among performance, power, and V_{min}. We find that the di/dt droops caused by microarchitectural stall events during a kernel's execution determine its V_{min} value.

These experimentally derived insights lead to our finding that we can predict each kernel's V_{min} value with at most 3% error, using microarchitectural performance counters. We

J. Leng is with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, 200240.
E-mail: leng-jw@sjtu.edu.cn
A. Buyuktosunoglu, R. Bertran, and P. Bose are with IBM Research. Y. Zu is with Google and J. Reddi is with Harvard University.
Manuscript received XX XX, 2018; revised XX XX, 2018.

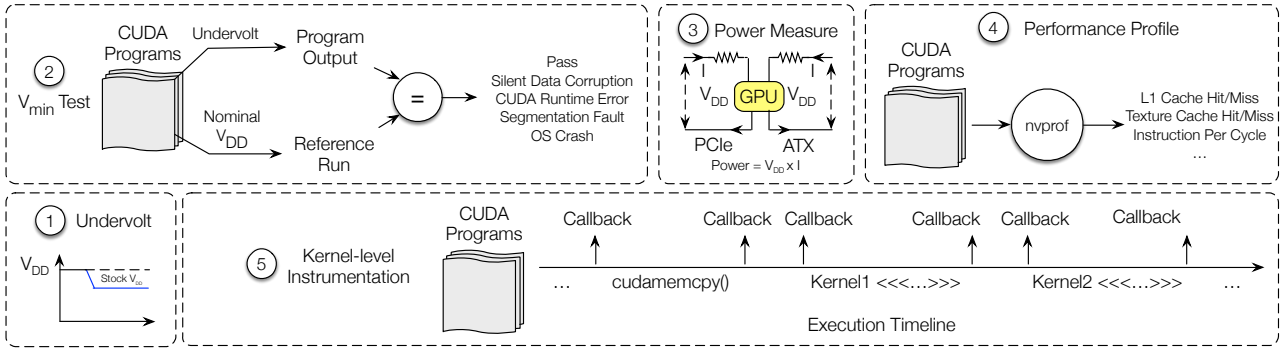


Figure 1: Overview of the experimental setup. 1. Undervolt: we use an overclocking tool to control the voltage. 2. V_{min} test: we measure the V_{min} point of each program by gradually undervolting the GPU and check the program output correctness. 3. Power measurement: we use custom power-sensing circuitry to measure GPU power. 4. Performance profile: we use nvprof to access performance counters. 5. Kernel-level instrumentation: we use the callbacks before and after each kernel invocation to measure the V_{min} and power of each kernel.

explore different V_{min} prediction methods, and show that a neural network based predictor and a hand-engineered piecewise linear model both achieve high prediction accuracy. Furthermore, the proposed V_{min} predictor enables the feasibility of a dynamic guardbanding system, where the GPU supply voltage is regulated on a kernel basis. We describe a conceptual system centered around the V_{min} predictor. In such a system, which we name as predictive guardbanding, the supply voltage of each kernel is determined by V_{min} predictor's output. We show that a predictive guardbanding system can closely match the kernel's voltage guardband to its characteristics, and therefore allows the kernel to operate with the reduced voltage and achieves significant energy saving benefits.

In summary, we make the following key contributions:

1) We find that there consistently exists a large amount of voltage margin across several off-the-shelf GPU cards, where the voltage margin is measured between a program's V_{min} and the nominal supply voltage.
2) We observe that the V_{min} values are strongly program-dependent as different programs V_{min} vary significantly. We experimentally determine that its root cause is voltage noise because it has a much greater impact compared to process and temperature variation. We further identify that the di/dt droop during the kernel execution is the dominant component of voltage noise.
3) We perform a quantitative study of the relationship between the program's performance characteristics and V_{min}, and study methods on how to use a kernel's performance counters to predict its V_{min} accurately.
4) We demonstrate the significant energy-savings potential of a conceptual exemplary design of predictive guardbanding that adopts the derived V_{min} prediction model through measurement on a real GPU. We also show the total cost ownership (TCO) improvement benefits in a datacenter that deploys multi-GPU servers.

We organize the paper as follows. Section II describes our experimental setup. Section III presents the V_{min} measurement results and analysis. Section IV analyzes the root cause of the large V_{min} variability. Section V studies the V_{min} prediction and Section VI details the proposed predictive guardbanding. Section VII evaluates our V_{min} predictor, and shows the GPU energy savings as well as a datacenter's TCO improvements achieved by a conceptual predictive guardbanding system centered around a V_{min} predictor. Section VIII discusses related work, and Section IX gives the conclusion remark.

II. Experimental Setup

This section describes our experimental setup in Figure 1. The central piece of our setup is the fine-grained voltage guardband exploration test, i.e. the V_{min} test (labeled using ① and ② in Figure 1). We also measure the program's power consumption (label ③) and profile its performance characteristics (label ④ and ⑤) to study the interaction of a program's V_{min} and its performance and power.

A. Voltage Guardband Exploration

We explore the voltage guardband on several off-the-shelf GPU cards and a large set of representative programs via V_{min} measurements. We describe the details of V_{min} test, studied GPU cards and programs.

***V_{min}* Test** We quantify the voltage guardband for each program by measuring its V_{min} point, a supply voltage point at which a program executes correctly but fails when the voltage is reduced any further. The V_{min} test includes two parts. In the first part (① in Figure 1), we decrease the GPU's operating voltage from its stock setting. For example, the stock setting of a studied GPU cards, GTX 680 card, is 1.09 V at the frequency of 1.1 GHz. Table I lists the stock settings of other GPU cards. We use a publicly available utility, MSI Afterburner [1] version 2.3, to control the GPU chip's voltage at a fixed frequency. We control the voltage at the granularity of 12 mV, and we do not modify the memory voltage and frequency.

We then measure each program's V_{min} point with a step of 12 mV undervolting, as the ② part in Figure 1 shows. At each step, we run each program and check its correctness by validating its output against the reference run at the nominal voltage. We consider each run as "pass" if i) for integer programs, the output is identical to the reference run, or ii) for floating-point programs, the output is within $10^{-2}\%$ of the reference run. We consider a voltage level as a working level if the program passes 1,000 times. V_{min} is the minimal

working voltage. We also study the error behavior for each program operating below its V_{min} point, but we run it only 100 times for each voltage level due to long experimental time.

Measurement Noise Control We control temperature and background activities on the GPU that may impact or skew the results. For temperature control, we adjust the fan speed to stabilize it at 40 °C when the program starts execution. This guarantees similar measurement temperature across studied programs: we observe only a small temperature change during program execution given its short execution time. We report all programs' V_{min} value measured at 40 °C and explicitly point out results measured at other temperatures. We nullify irrelevant system activities during the experiment, specifically the graphics activities, by installing another GPU card dedicated to graphics tasks. We do not control the activities on the CPU because they do not affect the V_{min} on the stand-alone GPU card (see Section IV-B for our results and analysis).

GPU Cards We perform V_{min} measurements on several off-the-shelf GPU cards. The studied cards span two architectural generations: Fermi (GTX 480 and 580) and Kepler (GTX 680 and 780).Table I lists their key microarchitectural specifications [39], [40]. Note that "core" refers to SM in Fermi and SMX in Kepler. Five different GTX 780 cards are used to verify the result reproducibility and to study if there is an observable difference related to process variation.

CUDA Programs We study a set of 58 representative programs from the CUDA SDK [38], Rodinia [9], AlexNet [25] on Caffe framework [22], and Lonestar [8] benchmark suites. These programs have diverse performance characteristics as they include computation and memory intensive programs, and regular and irregular programs. Their diverse performance characteristics leads to distinctive V_{min} behaviors, which lets us make insightful observations of the interaction between program characteristics and V_{min}. We will provide the complete list of the programs later.

B. Power Measurement

We measure each program's power consumption to study the relationship between the program's power behavior and V_{min}, and quantify the energy-saving benefits of operating at the V_{min} point. The part ③ in Figure 1 shows our power measurement setup. The GPU card consumes power from the PCIe connection and the ATX power supply. We measure the power consumption of both sources and add them up to get the GPU power. We insert a 25 mOhm shunt resistor at each connection to measure the instantaneous current and voltage and calculate the power consumption. We use the data acquisition unit NI DAQ 6133 [35] to record the data at a rate of 2 million samples per second. This power measurement setup is independent of the GPU card and lets us switch cards and measure their power consumption. Note that the measured power consumption is at board-level, which includes the GPU chip, DRAM and peripheral circuits such as voltage regulator.

GPU	GTX 480	GTX 580	GTX 680	GTX 780
Architecture	Fermi		Kepler	
Core Counts	15	16	8	12
Core Clock (MHz)	700	875	1100	1100
Register Per Core (KB)	128	128	256	256
L1 Cache (KB)	48/16 (Configurable)			
L2 Cache (KB)	768	768	512	1536
TDP (W)	250	250	195	250
Technology (nm)	40		28	

Table I: Microarchitectural specifications of four GPUs.

C. Profiling and Instrumentation

We use the NVIDIA profiler nvprof [41] to access GPU's performance counters. The counters profiled include various cache misses and functional unit utilization. We collect them at the kernel level; the run-to-run variation of these counters reported by nvprof is within 1%. In the V_{min} test, we rely on kernel-level instrumentation to control the voltage during each kernel's execution to measure its V_{min}. The CUPTI (CUDA profiling tools interface) library [37] provides instrumentation capability by registering the custom callbacks before and after each kernel and runtime API call. We implement our own callbacks to control each kernel's voltage.

III. Program-specific V_{min} Measurement

In this section, we use the measurement results to explain why it is desirable to optimize the guardband for the individual program, i.e., program-driven guardbanding. In particular, we study how far we can reduce the GPU's supply voltage from its nominal level to the safe limit that still satisfies the correctness level assumed by an application developer. Our results demonstrate the existence of significant guardband optimization potential, i.e., the margin between the nominal level and the safe limit for all GPUs, which can translate to substantial energy savings. Moreover, we also find a significant variability of required guardband for different programs, which suggests the need for program-driven guardbanding to fulfill the optimization potential. We also explore the feasibility of more aggressive optimization that lowers the voltage further below the safe limit and allows the error to happen. We observe that the catastrophic failure at the system level only occurs at a voltage level much below the safe limit, which suggests the feasibility of the more aggressive guardband optimization.

A. Quantifying the Potential with V_{min} Measurement

We perform the V_{min} measurement to quantify the voltage guardband reduction opportunity that still guarantees the program's reliable execution same as the nominal voltage level. At the V_{min} level, the program executes correctly but fails when the voltage is reduced any further. As such, we can reduce the supply voltage to the V_{min} level without affecting the program's correct execution, leading to substantial energy savings or performance improvements as we will discuss later.

We study both the card-specific and program-specific optimization opportunity. Specifically, we follow the methodology described in Section II-A, and conduct V_{min} measurement using the 58 representative programs on four GPU cards listed in Table I spanning two architectural generations (Fermi and Kepler architecture). The comprehensive measurement helps us to study and quantify the program-specific voltage

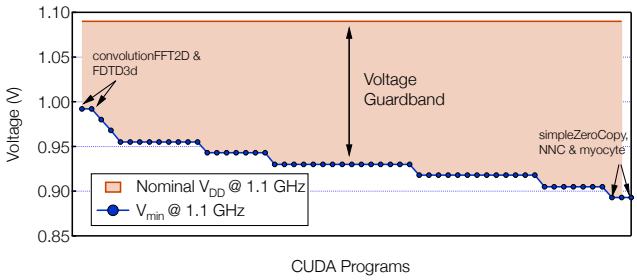


Figure 2: V_{min} measurements for 58 programs on the GTX 680.

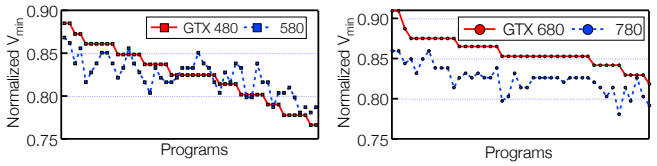
guardband behavior because the voltage guardband that exists for a program is the difference between the card's nominal voltage and the program's V_{min} value.

We first study the voltage guardband of the 58 programs on a GTX 680 card. Recall that the voltage guardband for a program is the margin between the nominal voltage and its V_{min} point. The voltage stock setting of the studied GTX 680 is 1.09 V at a frequency of 1.1 GHz. Figure 2 plots the measured results, from which we make two fundamental observations. First, a relatively large amount of guardban d optimization potential exists for all the studied programs. The measured V_{min} value varies from 0.89 V to 0.99 V. Considering that the nominal supply voltage of the GTX 680 card is 1.09 V, we can calculate that a relatively large percentage of the supply voltage (i.e. 9.2% to 18.3%) can be reduced without affecting the program's functional correctness. The magnitude is similar to the measured voltage guardband percentage on an Intel Core 2 Duo processor reported in prior work [48].

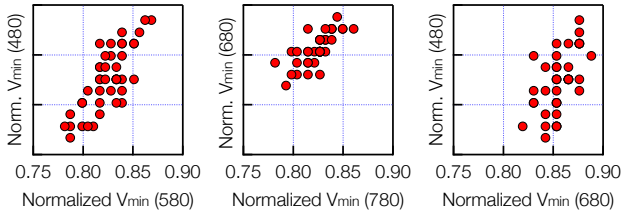
Second, Figure 2 shows a large variability in the V_{min} values of studied programs, indicating that a program's V_{min} value strongly depends on its characteristics. The difference between the highest V_{min} value (0.99 V) and the lowest value (0.89 V) is 0.1 V for the studied programs. Two programs (FDTD3d and convolutionFFT2D) have the highest V_{min} value, and three programs (simpleZeroCopy, NNC and myocyte) have the lowest value, as labelled in Figure 2. Most of the programs have a V_{min} value of about 0.93 V.

We further find that these two observations, i.e., the relatively large voltage guardband and the program-dependent V_{min} behavior, exist on different GPU architectures. In total, we perform V_{min} measurements on four GPU cards: GTX 480 and GTX 580 (Fermi architecture) and GTX 680 and GTX 780 (Kepler architecture). Their specifications are described in Table I. Because each card has a different nominal voltage, we normalize each card's V_{min} to its nominal voltage for comparison. Figure 3 plots the normalized V_{min} on four cards and their comparison. The range of voltage guardband is similar across these cards: 11.5% - 23.3% on GTX 480, 11.6% - 20.3% on GTX 580, 9.2% - 18.3% on GTX 680 and 14% - 22.5% on GTX 780, as shown in Figure 3a.

We also observe the existence of program-dependent V_{min} behavior, i.e., different programs with different V_{min} values, across all the four cards. Moreover, V_{min} values on cards with same architecture are more correlated than those on cards with different architectures. In Figure 3b, V_{min} values on GTX 480 and 580, and on GTX 680 and 780, are more correlated than V_{min} values on GTX 480 and 680. We can explain the lower



(a) Similar V_{min} range on different cards.



(b) V_{min} correlation on different cards.

Figure 3: Measured normalized V_{min} across four GPU cards.

correlation of V_{min} between two different architectures by program's runtime behavior. A program's characteristics may change when it is running on a different architecture, leading to a different V_{min} value.

In summary, we demonstrate that there is a substantial voltage guardband optimization potential for GPUs. Moreover, we also find that the variability of different programs' guardband requirement is significant (e.g., 0.1 V in the GTX 680 card). As such, we must design a program-driven voltage guardbanding scheme to realize the optimization potential.

B. Agressive Optimization Beyond the V_{min}

We further explore whether it is possible to operate aggressively below the safe voltage limit, i.e., the V_{min}. In specific, we conduct experiments to inspect how the errors manifest for different programs when they operate below their V_{min} levels. We observe four dominant types of error events from our experiments. The most catastrophic error is the operating system crash; while the rest three, i) silent data corruption, ii) CUDA runtime errors, GPU driver fault or segmentation fault, ii) infinitely long execution are relatively benign and easier to handle compared to the operating system crash. We describe the details of error events and how we detect their occurrence and measure their probabilities in this section.

Silent data corruption (SDC) [10] refers to when a program finishes execution without any warning but produces an incorrect end result. We detect it by comparing the test output from the undervolt run against a *golden* output from a reference (fault-free) run. We examine the integer and floating-point output separately, as described earlier. CUDA runtime errors refer to the erroneous execution of a program that fails at runtime (e.g., memory and stream management). The CUDA runtime explicitly reports such errors. Driver fault occurs when the GPU driver code executed by the CPU loses communication with the GPU. Often this results in a screen freeze followed by an automatic hard reset of the GPU. These two types of errors can be detected from the standard error output. The harshest error is the OS crash, after which a manual reboot is required. We stop the voltage reduction experiment once an

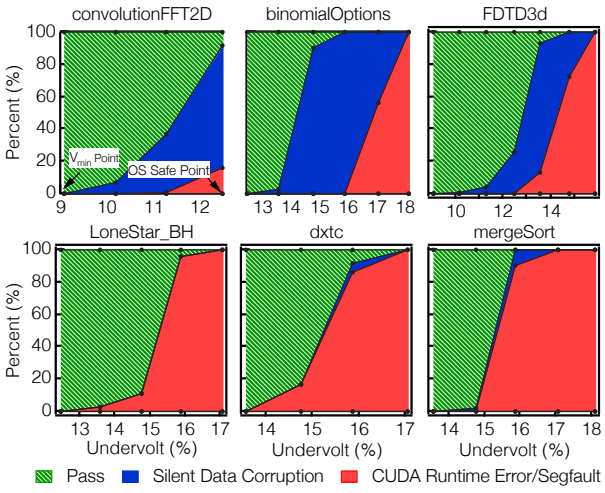


Figure 4: The distribution of runs for pass, SDC, CUDA runtime error or segmentation fault when increasing undervolt percent.

OS crash happens. Some programs, such as BFS and DMR, operate on graph data structures and use iterative algorithms to converge to the final output. An error may cause it to deviate from convergence, and its execution time becomes longer or infinitely long. Due to its rare occurrence, we manually detect the error and do not study its probability.

We gradually increase the undervolt percent level. At each undervolt level, we run the program 100 times and record the outcome. Figure 4 shows the undervolting experiment results for six representative programs. In each subplot, the x-axis shows the undervolt percent, i.e., percent reduction from the nominal voltage. The percent value at the leftmost x-axis point corresponds to the program's V_{min} point: the V_{min} of convolutionFFT2D is 0.99 V, which corresponds to 9% undervolting, marked as "V_{min} Point" in Figure 4. The rightmost x-axis point is marked as "OS Safe Point," beyond which the program can cause an OS crash. The y-axis plots the distribution of 100 runs that result in a pass, SDC, CUDA runtime error or segmentation fault. E.g., at the 11.3% undervolt level, convolutionFFT2D has 63 runs that lead to a pass, 36 runs of SDC, and one run of runtime error.

We summarize three key observations from this experiment. First, an additional 4-5% undervolt percent below the V_{min} point usually causes an OS crash. In other words, the OS safe point is 4-5% lower than the V_{min} point. Second, we observe two program categories through their different failure behaviors. The top three programs in Figure 4 have significant SDC incidence during undervolting, whereas the bottom three suffer primarily from crash failures (runtime error or segmentation fault). As such, we call the first category as "SDC-prone," and the second as "crash prone." In our study, there are 37 and 20 programs for each category, respectively. We inspect their source codes to diagnose the possible cause of their behavioral differences. We find that the most noticeable difference between the two categories is the intra-loop control dependency (i.e., conditional branches and embedded function calls). Programs with large such dependency are prone to crash errors. Instead, programs with minimal such dependency and fixed loop counts have more significant SDC incidences before the onset of crash errors during undervolting. This observation matches the common intuition that control-intensive codes have higher crash probability, because of the higher likelihood of illegal memory address references.

Third, the program failure probability increases as the undervolt level increases. The possible reason is that a lower voltage translates to less timing margin and therefore a higher error probability. Moreover, we observe an avalanche error effect when the voltage is below a particular value. For example, the error probability of FDTD3d increases from 3% to 90% when the undervolt percent increases from 10% to 12%. The common design practice that most paths are skewed toward the critical timing specification of the processor [21] may cause this avalanche error effect. When the voltage goes below V_{min}, most paths would have a timing violation, causing an avalanche error effect.

In summary, more aggressive optimization by lowering the voltage further is possible because the program can still execute correctly at times below the V_{min} point. However, the challenge is to detect the execution error such as SDC. Moreover, the potential improvement might be marginal given the small additional margin and the avalanche error effect. As such, we focus on pushing the guardband for energy improvement while still operating above the V_{min}.

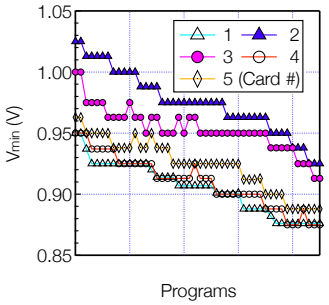
IV. Root Cause Analysis of V_{min} Variability

In this section, we analyze the *root cause* of the significant program-level V_{min} variability. We analyze two crucial aspects of the root cause that can impact how we design the program-driven guardbanding scheme. First, our measurement-based analysis indicates that among the candidates of the process, voltage, temperature variation (PVT), and aging, which the voltage guardband mostly protect against, voltage variation (i.e. voltage noise) is the cause of the program-level V_{min} variability. Second, we identify that the GPU microarchitecture activities with every kernel's execution are the sources of large voltage noise events. This observation is non-intuitive owing to the existence of complicated activity patterns, such as runtime execution, initial kernel launch, and kernel-to-kernel transition. Our observation encourages to manage the guardband at the granularity of individual kernel as we will detail later.

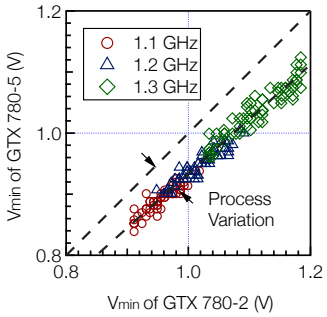
A. Identifying the Dominant Variation Kind

We first determine which kind of variation causes the program V_{min} variability, from the candidates of the process, temperature, voltage variation and aging. Understanding this problem is the first step towards designing the most appropriate guardbanding scheme because each type of variation has a different implication for optimization. After identifying the voltage variation (noise) as the cause of the program V_{min} variability, we further analyze its dominant component (IR drop or di/dt droop).

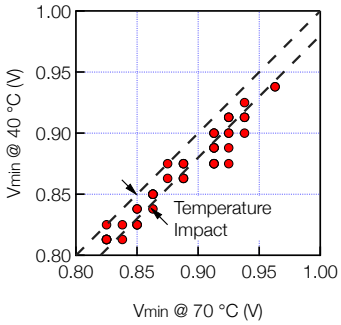
Process Variation Process variation, usually caused by imperfect lithography [7], [43] and dopants diffusion [7], [51], can lead to variable transistor thresholds and speeds, and therefore different V_{min} values. It can be further divided into inter-die



(a) Process variation impact on V_{min} of five GTX 780 cards.



(b) Process variation impact on V_{min} of two cards and three frequencies.



(c) Temperature variation (40 °C and 70 °C) impact on V_{min}.

Figure 5: (a) and (b) study process variation impact on V_{min}, and (c) studies temperature variation impact on V_{min}.

variation, i.e., variable features of the same transistor from different dies, and intra-die variation, i.e., variable features of transistors on the same die [7], [42].

We use five GTX 780 cards to study the impact of process variation. Figure 5a plots the V_{min} values of studied programs, which are measured at 40 °C. Program names are omitted because of space constraints and are sorted in the descending order of Card 2's V_{min}, the highest among all cards. The largest noticeable difference of V_{min} among the five cards is that the V_{min} values of all programs on one card shift up or down by a relatively constant value compared to the values on the other card. The largest V_{min} difference of the same programs between two cards is about 0.07 V.

We also measure Card 2 and 5's V_{min} values at three frequency points: 1.1, 1.2, and 1.3 GHz. Each marker in Figure 5b plots the V_{min} of the same program running on two cards at a frequency point. If there were no variation, a program's V_{min} would be identical on both cards, which would result in markers lying on the 45-degree diagonal line in Figure 5b. However, in fact, the V_{min} values on the Card 2 are consistently higher than those on the Card 5 by a relatively constant offset, which increases slightly as the frequency increases. We also find that the magnitude of the V_{min} difference between two cards is not identical for all programs in Figure 5a and Figure 5b: some programs have a greater V_{min} difference between two cards than others.

Given the same experimental conditions for other factors, we attribute the cause of V_{min} disagreements over different GPUs to process variation. Both the inter-die and the intra-die variation have a systematic and a random component. The systematic component can explain the constant offset of the V_{min} differences over two cards and the random component can explain the small "turbulence" of the V_{min} differences by causing a critical timing path shift. Note that additional measurements on more GPUs are required to draw a more statistically sound conclusion.

Temperature Variation The processor's temperature constantly changes over time due to its time-varying power consumption profile. Because of the processor's cooling system's inability to completely remove the generated heat, the temperature increases when it continuously consumes high power and decreases when it idles down [20]. As the transistor speed varies as temperature increases [55], designers must add voltage margin to offset its effect for reliable operation.

We measure the V_{min} at two temperatures (40 °C and 70 °C) to study the temperature variation impact. The former is the nominal temperature while the latter is the highest temperature when running a stress test at the highest frequency and lowest fan speed. Thus, the 70 °C is an improbable worst-case scenario for regular CUDA programs. Figure 5c shows the results. We observe a similar impact on V_{min} as the process variation but with a smaller magnitude: V_{min} at 70 °C is consistently about 0.02 V higher than those at 40 °C.

Aging Effect In our study, we cannot directly measure the impact of aging on V_{min}. We believe it is unlikely that it is the aging effect that causes such large V_{min} variability among the programs. The published measurement results on a recent IBM z System shows that the circuit speed degrades only 1-2% in the long term [32]. Moreover, all our experiments are done within a few months. Thus the impact of aging would be even smaller. Hence, the aging effect cannot explain the magnitude of observed V_{min} variability between programs and cards.

Voltage Noise In summary, we observe that both process and temperature have a relatively uniform impact on the V_{min} across all programs. Neither can explain the large program-level V_{min} variability. The voltage noise remains the only possible cause, per method of exclusion. In other words, the program with a higher V_{min} value is due to a higher magnitude of voltage noise. This also matches with the established knowledge that voltage noise results from the interaction between program activity and the processor power delivery network [16], and thus V_{min} depends on the program characteristics. Note that measuring the voltage noise directly through the oscilloscope or on-chip sensors [6], [48] can directly prove our observation. We leave it for future work.

We also observe that voltage noise has a more substantial impact on the voltage guardband compared to process and temperature variation. The measured V_{min} values range from 0.89 to 0.99 V on the same card with the same temperature, which indicates the magnitude of voltage noise of 0.1 V. This is larger than the measured process variation impact of 0.07 V and temperature variation impact of 0.02 V. In the rest of the

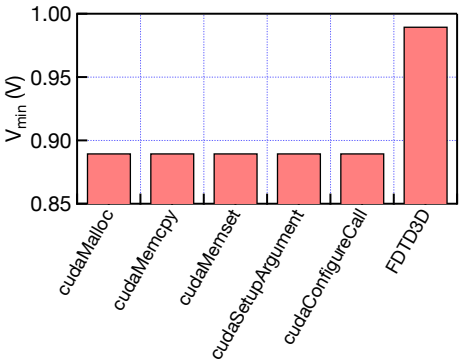


Figure 6: CUDA runtime functions V_{min} measurement results.

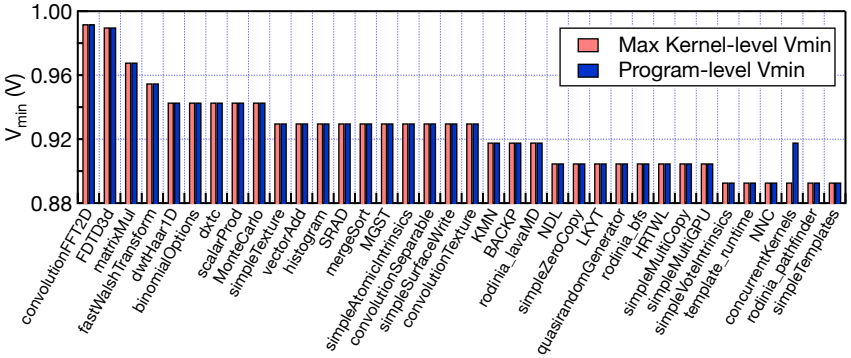


Figure 7: Program-level and max. kernel-level V_{min} match. Thus, individual kernel acitivity decides V_{min}.

paper, we focus on voltage noise analysis on the GTX 680 card, unless it is explicitly mentioned.

We further analyze the impact of different components of voltage noise, including IR drop and di/dt droop in Equation 1. The IR drop component is determined purely by the instantaneous current draw, whereas the di/dt droop component is determined by the current draw's increasing rate. Owing to their distinctive properties, we must understand which component has a more dominant impact to determine the proper optimization effort.

$$V_{actual} = V_{DD} - I \times R - L \times \frac{di}{dt} \quad (1)$$

We leverage the IR drop's property to test the hypothesis on its dominant impact. Owing to the space limitation, we briefly describe the process and refer to our prior publication for more details [28]. We measured the GPU power consumption and collected various power-related performance counters such as IPC and DRAM utilization. We find that there is no direct correlation between a program's V_{min} and its power consumption. Thus, the di/dt droop component has a more dominant impact on the V_{min} than the IR drop component.

B. Identifying the Dominant Program Activity

After identifying the di/dt droop as the root cause of the large program-level V_{min} variability, we further analyze which program activity pattern causes such di/dt droop. Given the existence of complicated interleaved CPU-GPU activity, we first categorize program activities to different types centering around kernels as a kernel is the scheduling unit from the CPU to GPU. Our analysis unveils that the activity during individual kernel execution has the most dominant impact on the V_{min}. This encourages to explore a dynamic guardbanding scheme at the granularity of kernel.

CUDA Runtime We find that many programs spend a significant amount of time on CUDA runtime functions such as transferring data back and forth between CPU and GPU. But it is not intuitive what their impact on V_{min} is. Thus, we study several runtime functions that commonly exist in CUDA programs. They are cudaMalloc, cudaMemset, cudaMemcpy, cudaSetupArgument and cudaConfigureCall. The first two allocate and set the specified size of global memory space in a GPU to a certain value, and cudaMemcpy transfers data between the CPU and the GPU. The last two push the kernel invocation arguments to the GPU stack memory [36], which are common before a kernel execution.

To verify if any of these functions is the source of large di/dt droop, we measure each function's V_{min}. We use the CUPTI library (see Section II) to register a callback, which controls the voltage, before every runtime function invocation. We measure the five functions' V_{min} by only performing undervolting during the tested function execution. Figure 6 plots the results. The V_{min} of these functions is 0.89 V, which is 0.1 V lower than the highest measured V_{min} of all programs (FDTD3d). Therefore, the activity during runtime functions is not the source of large di/dt droops.

Kernel Activity The kernel activity is another potential source of large di/dt droops, which can be divided into inter-kernel activity and intra-kernel activity. The former refers to the consecutive launch of kernels in GPU programs while the latter refers to the pipeline activation and stall owing to various microarchitectural events [30]. Both can result in repetitive current fluctuations (ramp-up/ramp-down) and therefore can cause large di/dt droops. We measure the V_{min} value of each kernel to study which kernel activity is the main source of large di/dt droops. The methodology is similar to the one used in runtime V_{min} measurement. We define the measured V_{min} as *kernel-level* V_{min} to distinguish from the *program-level* V_{min}, which is measured by performing undervolt for the entire program execution.

We compare the program-level V_{min} and its maximum kernel-level V_{min} to test the hypothesis that the inter-kernel activity determines the program's V_{min} value. If the hypothesis holds true, the measured kernel-level V_{min} would be much smaller than the program-level. However, we observe that the kernel-level V_{min} of all programs except concurrentKernel match the program-level as shown in Figure 7. Thus, neither of the two types of activities causes large di/dt droops. The mismatch for concurrentKernel can be attributed to the side-effect of kernel-level V_{min} measurement due to the use of CUPTI library, that is, serialized execution of all kernels. That program originally has multiple kernels which run concurrently. The serialization side effect reduces the activities on GPU, and therefore also reduces its V_{min}.

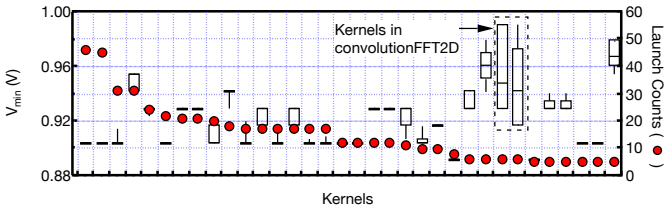


Figure 8: The V_{min} variability with different inputs.

In summary, the di/dt droop caused by the intra-kernel microarchitectural events is the determinant of the kernel's V_{min}, whose maximum value also determines each program's V_{min}. Such observation lays the foundations for managing the voltage guardband for at the granularity of a kernel according to its microarchitectural performance characteristics.

V. V_{min} Prediction

In this section, we demonstrate how a program can predict its V_{min} value accurately. We study V_{min} prediction at the kernel granularity using each kernel's microarchitectural performance counters because our previous analysis shows that di/dt droops caused by microarchitectural events during the kernel execution cause the large V_{min} variability. We discuss two approaches that use a kernel's performance characteristics metrics, i.e., performance counters to predict its V_{min} value accurately. The two approaches feature a top-down and a bottom-up methodology. The top-down method treats the predictive model as a black box and disregards the causal relationship between kernel's performance characteristics and V_{min} during construction. The bottom-up approach tries to interpret the contribution of different performance characteristics instead and involves more human effort in constructing the model.

A. Program Characteristics and Input Impact on V_{min}

To study the impact of the input, we measure the V_{min} value of different invocations of the same kernel. In CUDA programs, a kernel can be launched multiple times with different input data. Figure 8 shows the box plot for the V_{min} of kernels with multiple invocations. The results show that most kernels' V_{min} values only vary about 1%. Kernels in convolutionFFT2D have the greatest variability, ranging from 0.92 to 0.99 V. We inspect its source code and find that invocations with much lower V_{min} do not contribute to the program output. Thus, our methodology of checking program output cannot measure these launches' V_{min} values accurately. The V_{min} variability of other launches that contribute to the program output is within 0.01 V.

In summary, most kernels have small V_{min} variability regarding the input, and some kernels' V_{min} is more sensitive to the input because of the sensitivity of their performance characteristics to the input. For example, we observe that the V_{min} of some kernels varies up to 0.04 V with different input in Figure 8. Their performance characteristics (counters) also have larger variability than other programs.

In the rest of this subsection, we study the V_{min} prediction at the granularity of a kernel owing to three reasons. First, predicting each kernel's V_{min} value will incorporate information

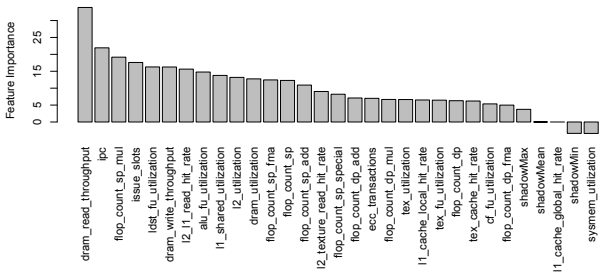


Figure 9: The importance of all performance counters ranked by the Boruta algorithm.

of the program's V_{min} value because the program-level V_{min} is determined by the maximum kernel-level V_{min} as shown in Section IV-B. Second, predicting at the kernel-level exposes larger optimization scope because a program can have multiple kernels and each kernel can have a different V_{min} value. Third, a kernel's V_{min} value is insensitive to its input. In other words, we can simply use the performance counter without the input information to predict the V_{min} value.

B. Top-Down Approach for V_{min} Prediction

We first study two top-down V_{min} prediction approaches using the performance counter, which let us to quickly evaluate its feasibility and accuracy. The top-down approach does not require any prior knowledge about the interaction between performance counters and V_{min}. Instead, it uses all possible performance counters from a large number of programs and automatically constructs the prediction model, allowing a fast implementation and evaluation. We consider two methods: linear regression and neural network. The former produces a linear model, and the latter produces a nonlinear model.

We collect 28 performance counters to construct the prediction model, including instruction per cycle (IPC), utilization level of different functional units (arithmetic logic, floating-point and load-store units), single-precision (SP) and double-precision (DP) floating-point operations per second (FLOPS) and hit and miss rates of various caches (instruction, data and texture cache). These counters are input features for both the linear regression and neural network prediction models.

For the linear regression, we collect the 28 performance counters from 557 kernel launches, which form a 557×28 input matrix for the linear regression. We use the same 557×28 matrix to train a neural network to predict the undervolting level. We choose a neural network with one hidden layer, which we show later yields a sufficiently accurate model. There are 28 neurons in the input layer, and ten neurons in the hidden layer, according to the rule of thumb for selecting the size of hidden layer [33]. The neural network is trained for 1,000 epochs for the convergence of model accuracy.

C. Bottom-Up Approach for V_{min} Prediction

We then evaluate a bottom-up approach for V_{min} prediction. The bottom-up approach identifies a small set of performance counters that strongly correlate with V_{min}, and constructs a simple model with a comparable accuracy of the top-down approach. The benefit of using bottom-up is the derived

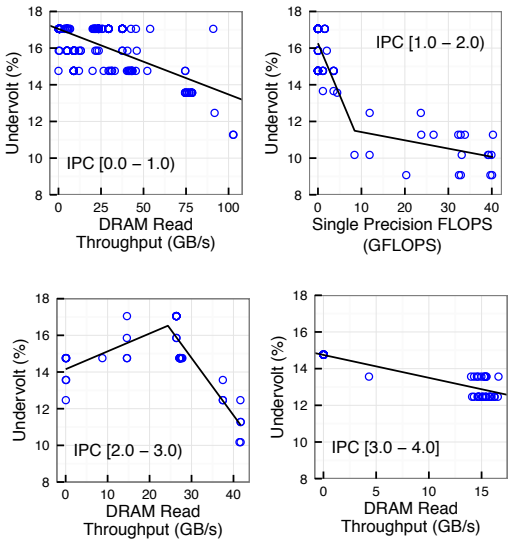
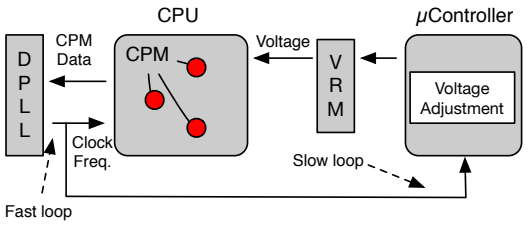
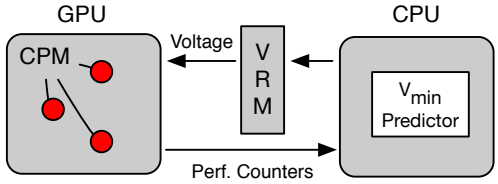


Figure 10: Piecewise linear V_{min} model using the IPC, DRAM read throughtput and single precision FLOPS.



(a) Adaptive guardbanding.



(b) Predictive guardbanding.

Figure 11: Diagram of adaptive and predictive guardbanding.

model's interpretability. For example, we find that the set of events that correlate most with V_{min} match those that our prior work has identified as the root cause of large di/dt droops [30]. The match of those events confirms the previous observation in Section IV-B that the di/dt droop during kernel execution determines the V_{min} value.

We first identify the most relevant performance counters (i.e., features). We rank the 28 features using the Boruta package [26], which iteratively sorts each feature's relevance and ranks them by their relevance to the dependent variable V_{min}. Figure 9 shows the results, wherein we find that DRAM read throughput, IPC, and single-precision floating-point multiply operations per second rank highest among all counters. The reason for them being the most relevant to V_{min} might be that they represent the general pipeline activity, which determines the voltage noise profile. In the below, we construct an accurate V_{min} model with those three performance counters and then interpret the model with the relationship of voltage noise.

We construct the V_{min} prediction model with the three most relevant counters using the following methodology. Because the peak achievable IPC in the Kepler architecture is four, we first categorize the kernels to four types by their IPC values: *[0, 1)*, *[1, 2)*, *[2, 3)* and *[3, 4]*. The heuristic we use is that the IPC is a good indicator of the pipeline stall degree. For example, kernels in the IPC *[3, 4]* region have V_{min} from 0.93 to 0.95 V. In this region, stalls are rare, and therefore the di/dt droop is small. In contrast, kernels in region *[0, 1)* have a larger V_{min} variability, from 0.9 to 0.97 V. In this region, pipeline stalls are frequent, and thus the di/dt droop may have a larger magnitude. IPC alone is not enough to capture the V_{min} variability. In each IPC region, we perform a piecewise linear regression with one breakpoint against the other two counters (DRAM read throughput and single precision FLOPS) separately. We show the results in Figure 10. We summarize the key findings below:

IPC *[0, 1)* The undervolt level decreases as DRAM read throughput increases in this region. When both the read throughput and IPC are low, the pipeline stalls are most likely due to the lack of threads (i.e., latency bound in Section V-A). In this case, the di/dt droop is small, and the undervolt level is large. Kernels with high read throughput and low IPC are memory bound. They have enough threads utilizing the memory bandwidth, which can generate a large current surge after the stall and therefore large droop.

IPC *[1, 2)* The metric FLOPS correlates better with V_{min} than read throughput in this region. The undervolt level decreases rapidly as the FLOPS increases and then plateaus after 10 GFLOPS. The possible reason is that higher FLOPS causes a greater current surge and therefore a larger droop.

IPC *[2, 3)* The undervolt level first increases but then decreases as the read throughput increases. Kernels with low read throughput and medium IPC are compute bound. They have mostly dependence-induced stalls. The increased read throughput means more memory stalls, but the two types of stalls are unlikely to align with each other, and thus the noise begins to decrease [6], [30]. However, the kernel becomes memory bound after the read throughput increases above 20 GB/s. Then, the di/dt droop starts to increase similar to the kernels in the IPC region of [0, 1).

IPC *[3, 4]* As mentioned earlier, kernels in this region have a small V_{min} variability because stalls are rare, and therefore the di/dt droop is small. The undervolt level increases slightly as the read throughput increases.

VI. Predictive Guardbanding

In this section, we present the details of our dynamic program-driven voltage optimization scheme to reliably reduce the GPU timing margin. We name our scheme *predicitve guardbanding* because its fundamental novelty lies in the use of performance counters by the runtime system to predict the selection of supply voltage level. Different from prior circuit- or microarchitecture-only guardband management techniques that can impose significant hardware design complexity, our proposed scheme minimizes the hardware complexity by leveraging the cross-layer optimization.

A. Design Overview

We describe the principles and implementation of predictive guardbanding, which relies on the collaboration between the software runtime and the hardware to optimize the voltage guardband accurately and reliably.

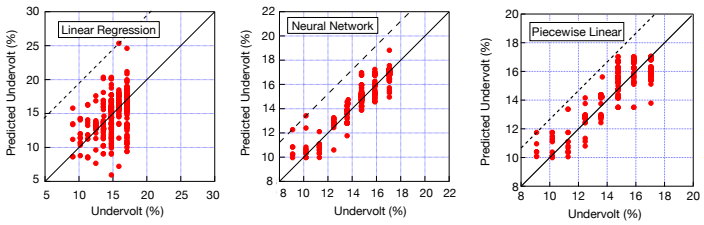
Before presenting the details, we first explain how the traditional voltage guardbanding mechanism works. The adaptive guardbanding, that is shown in Figure 11a and used in IBM Power 7 [27], relies on the critical path monitor (CPM) to detect timing margin. It uses a fast CPM-DPLL (digital phase lock loop) control loop to avoid possible timing failures: when the detected margin is low, the fast loop quickly stretches the clock. To mitigate the possible frequency loss, adaptive guardbanding also uses a slow loop to boost the voltage when the averaged clock frequency is below the target.

The adaptive guardbanding is a circuit-only solution and its design complexity is significantly higher than normal processors. In contrast, we propose the predictive guardbanding shown in Figure 11b, which is a cross-layer solution and therefore more light-weight in terms of hardware complexity. The proposed solution exploits the feasibility of kernel-level V_{min} prediction and the fact that the GPU exists as a co-processor in the system. The only modification for the GPU is the deployment of CPM for error detection as we will detail in the next subsection. All the other modifications can be implemented in the runtime software on the CPU.

We propose to use the CPU to manage the GPU's guardband at the granularity of kernel, which is the minimal control and scheduling unit by the CPU. Instead of using the complicated CPM-DPLL loop for margin detection, our scheme collects each kernel's performance counters and uses the V_{min} predictor detailed in Section II-C to reduce the operating margin. Moreover, it is unlikely that managing the guardband at a finer granularity could bring more benefits. The reason is the highly possible recurring pattern of large voltage droops during a kernel execution. Although we cannot measure it from the hardware, prior simulation-based study has shown that large voltage droops frequently occur during the kernel execution because of the throughput-optimized GPU architecture [30]. If large voltage droops happen frequently, managing the voltage with a finer granularity is unlikely to lead to more benefits.

We implement the predictive guardbanding on the existing GPU by leveraging the instrumentation capability of CUPTI, as described in Section II-C. For the every first invocation of a kernel, we execute the kernel with the nominal voltage and record the necessary hardware performance counters after it completes the execution. As a kernel in CUDA programs is typically launched multiple times (Figure 8), we use the recorded performance counters as the input to the V_{min} prediction model for the kernel's future invocations. When our predictive guardbanding runtime finds the recorded performance counters for a kernel, it predicts its V_{min} value and operates with the reduced margin.

In summary, the predictive guardbanding is a proactive program-driven dynamic guardbanding scheme. It uses performance counters, instead of complicated voltage margin sensors such critical path monitor (CPM) [11], to avoid the substantial calibration overhead. It adopts the V_{min} prediction model to directly determine the available timing margin, which achieves the advantage of not requiring the complex CPM-DPLL control loop in existing commercial CPU processors [27].



(a) Linear regression. (b) Neural network. (c) Piecewise linear.

Figure 12: Different V_{min} prediction model accuracy.

B. Handling V_{min} Prediction Failure

Although our model predicts V_{min} within 3% error margin for training programs, the prediction error for unseen programs, whose characteristics are very different from the training programs, could exceed the 3% error margin. This kind of corner case can result in a system failure, which makes the safety net necessary in the predictive guardbanding. In our work, we recognize the fact that the GPU is a co-processor and propose to offload the GPU error recovery to the CPU.

To ensure the full reliability coverage for all cases, we need to augment the GPU with "always-on" error detection. Hardware sensors can provide such constant monitoring capability more efficiently than software solutions. We can use existing lightweight voltage droop sensor, such as skitter circuit [15], [50], in our error detection process. Other lower-complexity error detectors, such as parity or ECC sensors [4], [5] available in large SRAM macros (e.g. caches or register files), can also provide an alert to error-prone voltage levels and can help in the exemplary design. Because these hardware detectors are limited to memory arrays, we also need the skitter circuits for the logic paths that are engaged in computation and control. The runtime system can use these sensors to detect the corner case, i.e., the larger-than-expected droop that causes a violation of the 3% error margin established from model training.

Upon detecting an error event, the GPU aborts the current kernel execution and reports it to the runtime system running on the CPU, which takes care of the error recovery. The runtime system first restores the voltage to the nominal level. Because the studied CPU-GPU system adopts the discrete memory subsystem, the error in GPU cannot corrupt the CPU memory. As such, the runtime system can restore the kernel's input memory by re-copying the relevant CPU memory. It can then launch the kernel again to recover from any timing-error-related corruption that could have occurred. The runtime system can collect this kernel's data, and add it to the offline training set to recalibrate the prediction model.

VII. Evaluation

We perform a comprehensive evaluation of the GPU predictive guardbanding system enabled by our V_{min} predictor. We

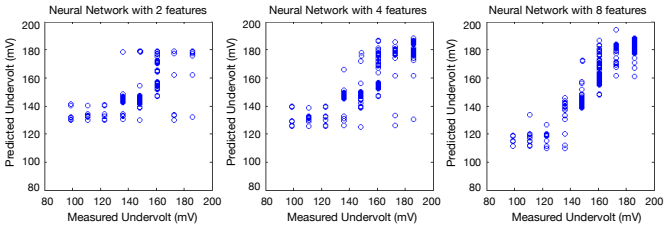


Figure 13: Number of features used in the neural network.

first evaluate the accuracy and robustness (i.e. cross-validation results) of the V_{min} prediction models studied in Section V. We then study the energy efficiency improvement potential of a conceptual GPU predictive guardbanding system, and show that with our V_{min} predictor, predictive guardbanding can achieve most, i.e., up to 80%, of the energy-efficiency benefits of the oracle case that operates at the V_{min} point.

A. V_{min} Prediction Accuracy

In this subsection, we first evaluate the accuracy of the V_{min} prediction model. We analyze the robustness of the V_{min} predictor constructed from the top-down (Section V-B) and bottom-up (Section V-C) method by design space exploration and cross-validation. In this process, we use three error metrics to evaluate the predictor's accuracy.

We use the *root mean square error* (RMSE) as the metric for prediction accuracy. We also evaluate the *maximum overprediction error*, i.e., the maximum value of measured V_{min} minus the predicted value. Reducing voltage below the actual V_{min} can cause program failure, so we need extra margin to tolerate the maximum overprediction error. Similarly, we also use the *maximum underprediction error*, i.e., the maximum value of predicted V_{min} minus the measured value. The underprediction error causes power wastage because the kernel could have run at a lower voltage level.

Figure 12a shows the prediction accuracy of the linear regression approach. Although it has an RMSE value of only 2.2%, it overpredicts many kernels' undervolting levels by 7%, where the maximum is 10%. Such overprediction error requires an additional voltage guardband to tolerate. Moreover, the maximum underprediction error is 10%, which means that the predicted undervolt level is 5% but the actual level is 15%. This results in a 10% guardband wastage.

Figure 12b shows prediction accuracy of the neural network model, which has an RMSE value of 0.5%. Between the two top-down models (linear regression and neural network), not only does the neural network outperform the linear regression approach in the average error, but the maximum overprediction error is only 3%. Moreover, the maximum underprediction error is only 2%, which translates to the less voltage guardband wastage. Thus, the neural network approach produces a more accurate V_{min} prediction model than the model constructed using the linear regression method, which suggests a nonlinear relationship between V_{min} and performance counters.

Figure 12c shows the accuracy of the bottom-up model, i.e. the piece-wise linear model with pruned features. As explained Section V-C, we only used three most relevant performance counters in building this model. The final model has an RMSE of 0.9%, and the maximum overprediction error and underprediction error is 2.7% and 3.3%, respectively. The accuracy of the piece-wise linear model using three key performance counters is comparable with the model produced by the neural network approach using all performance counters.

We also study the possibility of reducing the number of features in the neural network model. We construct the neural network model using 2, 4, 8, 16, and, 28 features (i.e. performance counters), which are selected according to their importance as shown in Figure 9. For example, when using 2 features, we choose DRAM read throughput and instruction per cycle owing to their greatest importance for determining the V_{min}. Figure 13 shows the accuracy increases with more features used. However, it plateaus after 16 features (not shown), which indicates the possibility of reducing feature count for the minimum overhead.

We evaluate the robustness of the bottom-up model via the 10-fold cross-validation test. We randomly partition the data into 10 sets, where 9 sets are used for training and one remaining set is used for validation. This process is repeated 10 times so that all the 10 sets are used for validation. We summarize the result as follows. We find that the modeling approach results in an almost zero bias, i.e. an even distribution between positive and negative errors. Furthermore, the approach is very insensitive to the rotation of validation set, which suggests a very low likelihood of model overfitting.

B. Energy Savings with Predictive Guardbanding

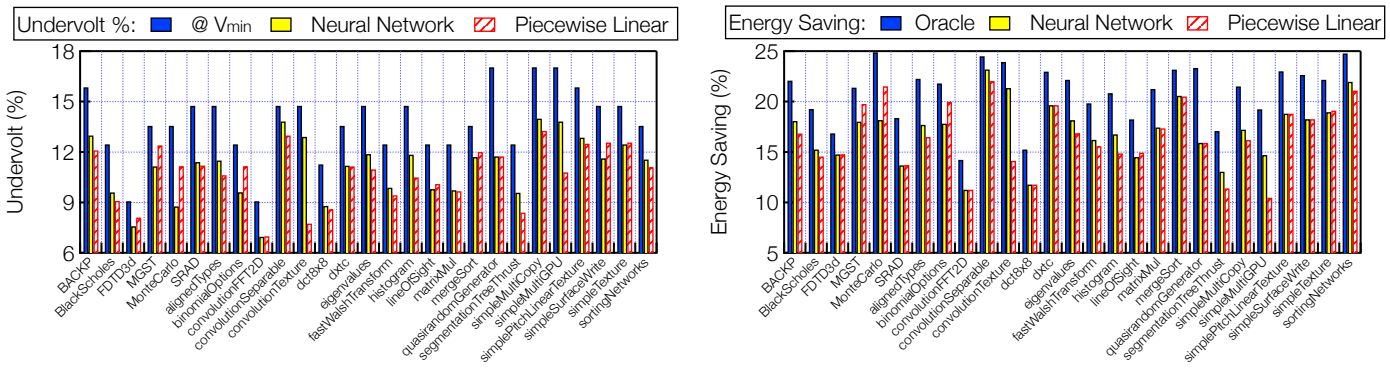
We now evaluate predictive guardbanding's energy saving benefits using the neural network and the piecewise linear prediction model. Because both models have a maximum overprediction of less than 3%, we add another 3% margin to guardband this model prediction error. Our results demonstrate that the great accuracy of V_{min} predictor is key to reclaim the possible voltage guardband.

Figure 14a compares the undervolting level at the V_{min} point and the level predicted by the two models plus the additional 3% margin. The predicted undervolting level of both models is always lower than the level at the V_{min}, which ensures that all programs execute correctly without any faults. However, the rollback mechanism for V_{min} misprediction is still required to ensure the functional correctness of other programs. The average gap between the actual and predicted undervolting level is only 2.7% for both models, which minimizes the guardband wastage.

Figure 14b compares the energy savings running at the level predicted by the two models with the oracle case. The energy savings range from 11.3% to 23.2% using the neural network model and 10.5% to 22% using the piecewise linear model, as opposed to 14% to 25% in the oracle case. The average savings are 16.9% and 16.3% using the two models and 21% in the oracle case. Using both models achieves over 80% of the energy-savings benefits of the oracle case.

C. Total Cost of Ownership Improvement

In the end, we evaluate the scenario of deploying GPUs with predictive guardbanding into a datacenter. Modern datacenters



(a) Undervolt level at measured and predicted V_{min}.

(b) Energy savings with oracle and predictive scheme.

Figure 14: (a) undervolt level and (b) energy savings at measured and predicted V_{min}.

have begun to adopt GPU even multi-GPU servers due to the increasing prevalence of large-scale machine learning applications [14]. As GPUs typically consume more power than CPUs, the energy efficiency of GPUs are critical for datacenter cost. We study how V_{min} prediction and kernel-level voltage guardband management can improve the cost to maintain a server equipped with GPUs.

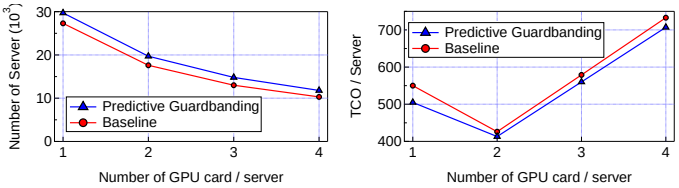
Table II summarizes the estimated datacenter cost. The figures are primarily drawn from public sources [3], [19]. As a comparison, we evaluate the maintenance cost of a GPU server over its lifetime with and without predictive guardbanding. In this study, a high-end NVIDIA Tesla K-80 GPU designed for datacenter is used. We approximate the price of one K-80 GPU to be around $4000. We assume the undervolting and energy saving benefits of the V_{min} predictors evaluated on GTX 680 apply to this GPU architecture as it is the case among the GPU architectures evaluated in our study. For each GPU card, we estimate that using V_{min} prediction can save 15% card power.

The power reduction of GPU through predictive guardbanding translates to about 12% power reduction for a four-GPU server. This enables more servers to be accommodated in a datacenter under a particular IT power budget, which better amortizes facility investment or CapEx over the long run. Table III shows the monthly maintenance cost of the entire datacenter with and without predictive guardbanding. As the predictive guardbanding leads to more servers, the server cost increases, which is amortized over its three-year lifetime. Power cost is the same because the facility's power budget is the fixed. We calculate power cost under the premise that the datacenter's normal load level is 80% of its peak. Capital expense is amortized over the 15-year lifetime.

In total, the datacenter cost for maintenance and operation is higher with predictive guardbanding because of the greater server count. However, on a per-server basis, the cost of ownership drops from $731 to $705 per month, because more

IT Power Budget (MW)	15	Power Conversion Ratio	0.6	Electricity Fee ($/kWh)	0.07
CapEx ($)	200 M	Facility Lifetime (yrs)	15	Server Lifetime (yrs)	3
CPU Server Power (W)	250	CPU Server Cost ($)	3000	PCIe slots	4
GPU Power (W)	300	GPU Cost ($)	4000	Average load level	0.8

Table II: Estimation of cost of a datacenter with GPU servers.



(a) Number of servers.

(b) TCO per server per month.

Figure 15: Datacenter TCO and compute capability.

servers amortize capital and other fixed expenses over the long run. In other words, for the same space, predictive guardbanding enables the datacenter to house more GPU servers and to provide higher compute capability, which is a more economic strategy. The cost of maintaining one GPU server is high compared conventional CPU-only server [49] because the GPU server we model can insert four high-end GPU cards, which amounts to more than 10,000$ purchase cost. Overall, by reducing GPU power with V_{min} prediction, the datacenter can reduce per-server maintenance cost by 3.7% and provide 15% more compute capability.

Figure 15 expands our TCO analysis to several different server configurations. We evaluate a CPU-GPU balanced configuration where each server has one GPU card, to a GPU-centric configuration where each server has four cards. In all cases, our solution consistently makes the datacenter accommodate more servers, and consequently more GPUs to compute. The higher server number also amortizes TCO more economically, which drops per server maintenance cost. We find that using two GPU cards per server achieves a sweet spot for cost per server. The reason is that the GPU purchase cost and the server power achieves a balance at this point.

	Baseline GPU Server	With Predictive Guardbanding
Server ($M)	5.46	6.23
CapEx ($M)	1.1	1.1
Power ($M)	1.0	1.0
Total $M	7.56	8.33
Server count (10^3)	10.34	11.81
TCO/Server ($)	731	705

Table III: Monthly maintenance cost of a GPU server.

VIII. Related Work

Our work is one of the first to perform a comprehensive measurement-based study of voltage guardband in GPUs [28]. We compare and contrast our work with prior work on voltage guardband both in CPU and GPU domains.

Voltage Guardband Characterization In our work, we conduct V_{min} test on a class of off-the-shelf GPU cards to characterize their voltage guardband. In the V_{min} test, we measure each program's V_{min} and study its error behavior and probability when the voltage goes below V_{min}. Our measurement results unveil a large voltage guardband potential and a program-dependent guardband behavior that has not been identified by any prior work. We also show that the di/dt droop during the kernel execution is the root-cause analysis for the program-dependent guardband behavior.

Many prior arts adopt the simulation approach to study voltage noise in the single-core [23], [44], [45] and multi-core [18], [34] CPUs. Researchers also conduct measurement-based study of voltage noise in CPUs [6], [24], [48]. In contrast, our work examines the problem in GPUs whose architecture is fundamentally different from CPUs.

There have also been prior studies that target voltage noise in GPUs. However, they all focus on modeling of voltage noise [31] and characterization of voltage noise via simulation [30], [52], [53]. In contrast, our work is the first to perform a comprehensive measurement study of voltage guardband using multiple off-the-shelf GPU cards.

Voltage Guardband Optimization We compare our voltage guardband optimization technique with prior works as shown in Table IV. Prior work on optimizing the voltage guardband can be categorized into two types. The first category reduces the voltage guardband while the processors continue to function correctly [27], whereas the second category tolerates timing speculation errors with the aid of an error detection and recovery mechanism [13]. Similar to the former scenario, our V_{min} test assumes that no error occurs at the V_{min} point. However, our work's emphasis is to characterize the voltage guardband and build a fundamental understanding of its essential characteristics in the context of GPU architecture.

Our work proposes a novel guardband management scheme, which reduces the supply voltage according to the program's performance characteristics. The proposed scheme, predictive guardbanding, is a cross-layer approach where the software takes the role of margin detection and error recovery while the hardware is responsible for error detection. Compared to the prior efforts including Razor [13] and adaptive guardbanding [27] shown in Table IV that are mostly hardware only, our scheme is simpler and more hardware cost-effective.

Both prior and our work rely on hardware sensors to reduce the operating margin for energy saving. The adaptive guardbanding in POWER7+ requires critical path monitor (CPM) [12] while Razor uses shadow latches [13], and another work uses ECC signal [4]. To the best of our knowledge, our work is the first to use performance counter for guardband management. Voltage sensors like CPM used in prior work incur significant calibration overhead as they need to measure a range of margin accurately. In contrast, our work only requires sensors to detect the event of voltage dropping a threshold instead of the entire margin range, because the software performs the margin detection (via prediction). Lightweight sensors such as skitter circuit [15], [50] suit our system.

	Razor [13]	Adaptive guardbanding [27]	Predictive guardbanding
Margin Detection	N/A	CPM	V_{min} prediction model
Error Detection	Shadow flip flop	CPM	Skitter circuit
Error Recovery	Branch misprediction	CPM-DPLL control loop	Relaunch kernel

Table IV: Comparison with prior work.

IX. Conclusion

In this work, we propose and evaluate a novel dynamic guardbanding scheme called predictive guardbanding, which can achieve energy-reduction benefits by lowering the GPU's processor voltage without inducing errors. The proposed scheme works in program-driven fashion as our comprehensive V_{min} measurement study unveils a strong program-dependent V_{min} behavior, which is also the fundamental challenge for the guardband optimization. We perform root cause analysis and identify that the di/dt droop during the kernel execution is the largest determinant of V_{min}. We further characterize the impact of program characteristics on the V_{min}, and show that we can predict the V_{min} of each individual kernel using performance counter information. The predictive guardbanding follows the principle of cross-layer optimization and uses the derived V_{min} prediction model to fulfill the energy savings potential by operating the GPU at the predicted V_{min} point.

Acknowledgment

This work is sponsored in part by Defense Advanced Research Projects Agency (DARPA), Microsystems Technology Office (MTO), under contract HR0011-13-C-0022, National Science Foundation (NSF) grant CCF-1218474, Semiconductor Research Corporation (SRC), and National Natural Science Foundation of China (NSFC) grant 61702328. The views expressed are those of the authors and do not reflect the official policy or position of the NSFC, the Department of Defense, the NSF, the SRC or the U.S. Government. This document is: Approved for Public Release, Distribution Unlimited.

References

[1] M. Afterburner, "MSI Afterburner," http://goo.gl/fs2pti, 2016.
[2] S. Anthony, "IBM and NVIDIA to build 100 petaflop+ supercomputers," https://goo.gl/vjhpCt, 2014.
[3] Aspen Server Cost, "GPU Server Cost," https://www.aspsys.com/servers/category/410/, 2015.
[4] A. Bacha and R. Teodorescu, "Dynamic reduction of voltage margins by leveraging on-chip ECC in itanium II processors," in *International Symposium on Computer Architecture*, 2013.
[5] ——, "Using ECC feedback to guide voltage speculation in low-voltage processors," in *International Symposium on Microarchitecture*, 2014, pp. 306–318.
[6] R. Bertran, A. Buyuktosunoglu, P. Bose, T. J. Slegel, G. Salem, S. M. Carey, R. F. Rizzolo, and T. Strach, "Voltage noise in multi-core processors: Empirical characterization and optimization opportunities," in *International Symposium on Microarchitecture*, 2014, pp. 368–380.
[7] S. Bhardwaj, S. B. K. Vrudhula, P. Ghanta, and Y. Cao, "Modeling of intra-die process variations for accurate analysis and optimization of nano-scale circuits," in *Design Automation Conference*, 2006, pp. 791–796.
[8] M. Burtscher, R. Nasre, and K. Pingali, "A quantitative study of irregular programs on GPUs," in *International Symposium on Workload Characterization*, 2012, pp. 141–151.

[9] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *International Symposium on Workload Characterization*, 2009, pp. 44–54.

[10] C. Constantinescu, I. Parulkar, R. Harper, and S. Michalak, "Silent data corruption - myth or reality?" in *International Conference on Dependable Systems and Networks*, 2008, pp. 108–109.

[11] A. Drake, M. Floyd, R. Willaman, D. Hathaway, J. Hernandez, C. Soja, M. Tiner, G. Carpenter, and R. Senger, "Single-cycle, pulse-shaped critical path monitor in the POWER7 microprocessor," in *International Symposium on Low Power Electronics and Design*, 2013.

[12] A. J. Drake, R. M. Senger, H. Deogun, G. D. Carpenter, S. Ghiasi, T. Nguyen, N. K. James, M. S. Floyd, and V. Pokala, "A distributed critical-path timing monitor for a 65nm high-performance microprocessor," in *International Solid-State Circuits Conference*, 2007.

[13] D. Ernst, N. S. Kim, S. Das, S. Pant, R. R. Rao, T. Pham, C. H. Ziesler, D. Blaauw, T. M. Austin, K. Flautner, and T. N. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *International Symposium on Microarchitecture*, 2003.

[14] Facebook AI Hardware, "Facebook to open-source AI hardware design," https://goo.gl/iDJIDN, 2015.

[15] R. L. Franch, P. Restle, J. K. Norman, W. V. Huott, J. Friedrich, R. Dixon, S. Weitzel, K. van Goor, and G. Salem, "On-chip timing uncertainty measurements on IBM microprocessors," in *International Test Conference*, 2008, pp. 1–7.

[16] E. Grochowski, D. Ayers, and V. Tiwari, "Microarchitectural simulation and control of di/dt-induced power supply voltage variation," in *International Symposium on High-Performance Computer Architecture*, 2002, pp. 7–16.

[17] M. S. Gupta, J. A. Rivers, P. Bose, G.-Y. Wei, and D. Brooks, "Tribeca: Design for pvt variations with local recovery and fine-grained adaptation," in *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 42. New York, NY, USA: ACM, 2009, pp. 435–446.

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

[19] J. Hamilton, "Cost of power in large-scale data centers," http://perspectives.mvdirona.com/2008/11/cost-of-power-in-large-scale-data-centers/, 2008.

[20] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[21] Janak Patel, "CMOS process variations: A critical operation point hypothesis." http://goo.gl/K0yWkf, 2008.

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, 2014.

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

[24] Y. Kim, L. K. John, S. Pant, S. Manne, M. J. Schulte, W. L. Bircher, and M. S. S. Govindan, "AUDIT: stress testing the automatic way," in *International Symposium on Microarchitecture*, 2012.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the International Conference on Neural Information Processing Systems*, 2012.

[26] M. Kursa and W. Rudnicki, "Feature Selection with the Boruta Package," *Journal of Statistical Software*, vol. 36, no. 11, 2010.

[27] C. Lefurgy, A. J. Drake, M. S. Floyd, M. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active management of timing guardband to save energy in POWER7," in *International Symposium on Microarchitecture*, 2011, pp. 1–11.

[28] 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 *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015.

[29] J. Leng, T. H. Hetherington, A. ElTantawy, S. Z. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: enabling energy optimizations in GPGPUs," in *International Symposium on Computer Architecture*, 2013, pp. 487–498.

[30] J. Leng, Y. Zu, and V. J. 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*, 2015, pp. 161–173.

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

[32] P. Lu, K. A. Jenkins, T. Webel, O. Marquardt, and B. Schubert, "Long-term NBTI degradation under real-use conditions in IBM microprocessors," *Microelectronics Reliability*, vol. 54, no. 11, 2014.

[33] Mic, "Selecting the number of neurons in the hidden layer of a neural network," http://goo.gl/frjVt7.

[34] 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*, 2012, pp. 249–260.

[35] National Instruments DAQ 6133, "NI DAQ 6133," http://goo.gl/ez2mof.

[36] NVIDIA CUDA API, "CUDA Runtime API," http://goo.gl/G27upA.

[37] NVIDIA CUDA Profiling, "CUDA Profiling Tools Interface," http://goo.gl/nbAVCf, 2015.

[38] NVIDIA CUDA SDK, "CUDA C/C++ SDK CODE Samples," 2011.

[39] NVIDIA Fermi, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi," http://goo.gl/zmoJkZ, 2009.

[40] NVIDIA Kepler, "GTX 680 Kepler Whitepaper - GeForce," http://goo.gl/fyg2z1, 2012.

[41] NVIDIA Visual Profiler, "NVIDIA Visual Profiler User Guide," http://goo.gl/gefn6p, 2015.

[42] K. Okada and H. Onodera, "Statistical parameter extraction for intra- and inter-chip variabilities of metal–oxide–semiconductor field-effect transistor characteristics," *Japanese journal of applied physics*, vol. 44, no. 1R, p. 131, 2005.

[43] M. Orshansky, L. Milor, and C. Hu, "Characterization of spatial intrafield gate CD variability, its impact on circuit performance, and spatial mask-level correction," *IEEE Transactions on Semiconductor Manufacturing*, vol. 17, no. 1, pp. 2 – 11, 2004.

[44] M. D. Powell and T. N. Vijaykumar, "Pipeline damping: A microarchitectural technique to reduce inductive noise in supply voltage," in *International Symposium on Computer Architecture*, 2003, pp. 72–83.

[45] ——, "Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise," in *International Symposium on Low Power Electronics and Design*, 2003, pp. 223–228.

[46] V. J. Reddi, M. S. Gupta, G. H. Holloway, M. D. Smith, G. Wei, and D. M. Brooks, "Predicting voltage droops using recurring program and microarchitectural event activity," *IEEE Micro*, vol. 30, no. 1, p. 110, 2010.

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

[48] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G. Wei, and D. M. Brooks, "Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling," in *International Symposium on Microarchitecture*, 2010, pp. 77–88.

[49] V. J. Reddi, B. C. Lee, T. Chilimbi, and K. Vaid, "Mobile processors for energy-efficient web search," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 3, p. 9, 2011.

[50] P. J. Restle, R. L. Franch, N. K. James, W. V. Huott, T. M. Skergan, S. C. Wilson, N. S. Schwartz, and J. G. Clabes, "Timing uncertainty measurements on the power5 microprocessor," in *International Solid-State Circuits Conference*, Feb 2004, pp. 354–355 Vol.1.

[51] S. Roy and A. Asenov, "Where Do the Dopants Go?" *Science*, vol. 309, no. 5733, pp. 388 – 390, 2005.

[52] R. Thomas, K. Barber, N. Sedaghati, L. Zhou, and R. Teodorescu, "Core tunneling: Variation-aware voltage noise mitigation in GPUs," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 151–162.

[53] R. Thomas, N. Sedaghati, and R. Teodorescu, "EmerGPU: Understanding and mitigating resonance-induced voltage noise in GPU architectures," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, April 2016, pp. 79 – 89.

[54] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Addison-Wesley Publishing Company, 2010.

[55] Y. Zu, W. Huang, I. Paul, and V. J. Reddi, "Ti-states: Processor power management in the temperature inversion region," in *International Symposium on Microarchitecture (MICRO)*. IEEE, 2016.