Energy Efficiency Benefits of Reducing the Voltage Guardband on the Kepler GPU Architecture

Jingwen Leng Yazhou Zu Vijay Janapa Reddi

The University of Texas at Austin
{jingwen, yazhou.zu}@utexas.edu, vj@ece.utexas.edu

***Abstract*—Energy efficiency of GPU architectures has emerged as an important design criterion for both NVIDIA and AMD. In this paper, we explore the benefits of scaling a general-purpose GPU (GPGPU) core's supply voltage to the near limits of execution failure. We find that as much as 21% of NVIDIA GTX 680's core supply voltage guardband can be eliminated to achieve significant energy efficiency improvement. Measured results indicate that the energy improvements can be as high as 25% without any performance loss. The challenge, however, is to understand what impacts the minimum voltage guardband and how the guardband can be scaled without compromising correctness. We show that GPU microarchitectural activity patterns caused by different program characteristics are the *root cause(s)* of the large voltage guardband. We also demonstrate how microarchitecture-level parameters, such as clock frequency and the number of cores, impact the guardband. We hope our preliminary analysis lays the groundwork for future research.**

I. Introduction

General-purpose GPU (GPGPU) architectures are increasingly becoming mainstream general-purpose computing counterparts to the CPU. For applications with significant data parallelism, the GPU architecture can offer better performance than the CPU architecture. The GPU's throughput-driven architecture maps well to data-parallel applications as compared to the CPU's single-thread-performance focused architecture.

The cost of throughput is power consumption. Historically, the power consumption of a general purpose GPU architecture has remained higher than that of the CPU, although the performance-per-watt efficiency of the GPU may be higher. High-performance GPU architectures have maintained a typical power consumption between 200 W and 250 W, whereas many of the most competitive commodity CPU counterparts plateau at around 130 W power budget.

With the recent GPU architectures, however, we have seen a significant emphasis on lowering the GPU's power consumption. For example, NVIDIA claims that the latest Kepler architecture achieves 3× the performance per watt of their previous-generation architecture (i.e., Fermi) [1]. State-of-the-art GPU power-saving efforts strongly reflect and follow the trend of CPU power optimizations. Typical optimizations include clock and power gating, dynamic voltage and frequency scaling (DVFS), and boosted clock frequencies [1], [2].

Although there has been increasing focus on applying traditional CPU power-saving techniques to GPUs, we need to focus on new(er) opportunities for energy optimization that push the GPU to the limits of its operation.

In this paper, we demonstrate the energy-efficiency benefits of pushing the GPU architecture to the limits of its operating voltage. To combat the worst-case process, voltage, and temperature variations, traditional design methodology requires excessive supply voltage guardband, which can be as high as 20% in a production processor [3]. The guardband is predicted to grow due to the increased variations as technology scales [4]. The industry-standard practice of designing for the worst case leads to wasted energy and performance because the circuit could have operated at a lower supply voltage or a higher clock frequency in the typical case [5], [6]. This tradeoff between performance, power, and reliability has remained largely unexplored by previous works in the case of GPUs.

Using NVIDIA's GTX 680 with the Kepler architecture, we show the power benefits of reducing the processor's supply voltage at a fixed frequency to a *critical voltage* point at which the program executes correctly but fails when the voltage is reduced any further. We observe that the critical voltage depends on the workload's characteristics and can vary from 11% up to 21% of the nominal voltage. Based on the critical voltage data of different programs, we demonstrate that the $L\frac{di}{dt}$ effect is the main cause for the GPU's reducible voltage guardband (i.e., the offset between critical voltage and the nominal supply voltage).

We also show that GPU architectural features like the number of cores and GPU program characteristics, for example, being memory bounded versus compute bounded, are two important factors that influence the amount of reducible voltage guardband. Understanding such features is crucial to effectively anticipate [7], predict [8] or mitigate [9] the reducible voltage guardband.

Our findings show that there is great potential in improving GPU energy efficiency by controlling its reducible voltage guardband from the architecture and program viewpoint. The key challenge, however, is understanding and identifying the components that impact the reducible magnitude.

The rest of this paper is organized as follows. Sec. II studies the extent that a GPU's voltage guardband can be pushed, as well as the benefits of exploiting the voltage guardband as a knob for improving a GPU's energy efficiency. Sec. III presents our analysis on the source of the GPU's voltage guardband and how microarchitectural activities and architectural features impact the benefits. Sec. IV concludes the paper with a summary of our important findings.

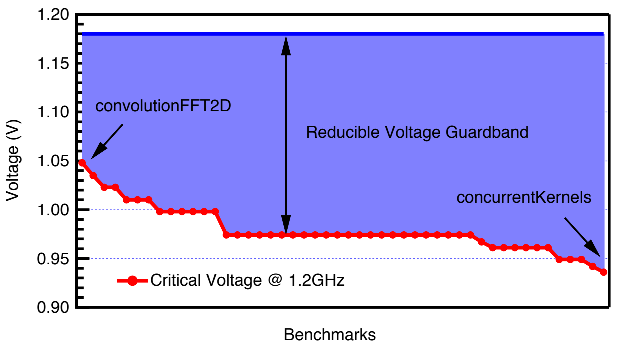

Fig. 1: Measured critical voltage for 48 programs on the GTX 680.

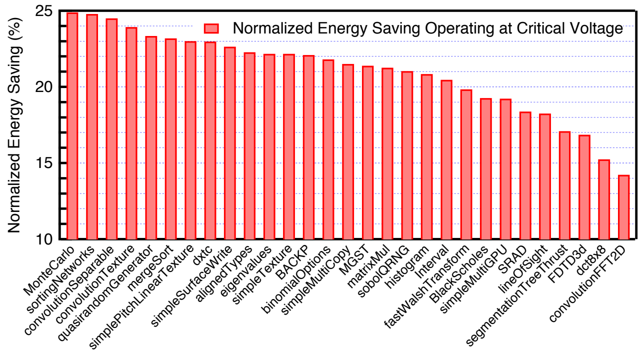

Fig. 2: Energy savings for operating the GPU at the critical voltages.

II. Pushing the Voltage Guardband

We use GTX 680, a high-end modern GPU with NVIDIA's latest Kepler architecture [1], to demonstrate that there is a large variation in the reducible voltage guardband among different programs. Pushing the guardband to the program's limit of correct execution can yield significant energy-efficiency benefits. Measurements show that we can achieve up to 25% energy reduction with this method. For all of our analysis, we use a total set of 48 programs from the NVIDIA CUDA SDK samples [10] and the Rodinia benchmark suite [11].

A. Critical Voltage Exploration

We experimentally reduce the operating voltage of each program to its *critical voltage*, an operating point at which the program executes correctly but fails when the voltage is reduced any further. The resolution with which we control the GPU's core voltage is 6 mV. As we decrease the GPU's operating voltage from its default 1.18 V at 1.2 GHz, we ensure program correctness at each step by checking if the GPU driver crashes during program execution and by validating program output against a reference run at nominal operating point. When validating program output, we restrict output data to be exactly the same as the reference run for integer workloads, and within 0.02% error range for floating-point workloads. We keep the core frequency, memory frequency, and memory voltage, and temperature constant during the experiment.

Fig. 1 shows the critical voltage for the set of programs we studied. The critical voltage varies from 0.93 V to 1.05 V, while the nominal operating point is 1.18 V at 1.2 GHz. Our measurement indicates that the critical voltage strongly varies among these programs. Nearly half of the workloads we run have a critical voltage of around 0.97 V, while some programs have a critical voltage that is above 1 V, the largest one is about 8% higher than the majority. Other programs have critical voltage that is below 0.95V. Overall, the voltage guardband is overprovisioned for the set of programs we evaluated.

To quantify the amount of "wasted" guardband, we use the term *reducible voltage guardband* to denote the offset between the nominal supply voltage and the benchmark's critical voltage. In the extreme case (benchmark concurrentKernels), the reducible voltage guardband is 0.25 V (21%). For the benchmark with the highest critical voltage (benchmark convolutionFFT2D), the reducible voltage guardband is 0.13 V (11%). Because the reducible voltage guardbands vary significantly across different programs, we attribute the difference to inherent program characteristics that impact (or determine) the worst-case critical voltage.

B. Energy-Efficiency Benefits

We measure and quantify the energy-saving benefits of operating the GPU at the programs' critical voltage. For GTX 680 power measurement, we adopt the following method: The GPU card is connected to the PCIe slot through a PCIe riser card and the ATX power supply. The PCIe riser card and the ATX power supply both have power pins that deliver power to the GPU. We measure the instantaneous current and voltage to compute the power supply from each of these sources. We sense the instantaneous current draw by measuring the voltage drop that occurs across a shunt resistor. We use NI DAQ 6133 to sample voltage at a rate of 2 million samples per second.

Fig. 2 shows the energy-saving benefits of operating at the critical voltage.[1] By lowering the core supply voltage without compromising frequency, we can improve energy efficiency. On (geometric) average, the energy savings is about 21%. We achieve the largest energy savings with the MonteCarlo program. Operating at its critical voltage (0.97 V) instead of the original 1.18 V reduces GPU energy consumption by 24%. The smallest improvement is seen with convolutionFFT2D. We can reduce its energy consumption by "only" 14%.

Energy reductions are generally proportional to the reducible voltage guardband, as shown in Fig. 3. However, the relationship is not linear, because the magnitude of energy savings depends on both the reducible guardband and the program characteristics. For instance, programs that are not compute-bounded tend to exercise the memory subsystem heavily. Because we only scale core voltage, we observe smaller benefits for those memory-bound programs. For compute-bound programs, we achieve larger benefits. Therefore, it is also important to understand program behavior for optimizing the GPU's energy efficiency using the voltage guardband.

[1]Henceforth, we use a subset of the programs mentioned in the beginning of Sec. II because of power or performance counter instrumentation difficulties. Nevertheless, the subsets are large enough to faithfully represent our observations.

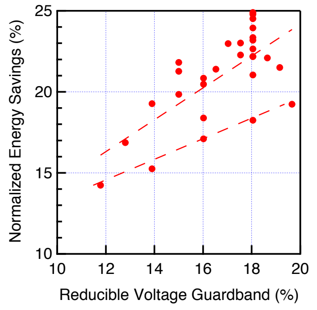

Fig. 3: Correlation between energy savings and voltage guardband of different programs.

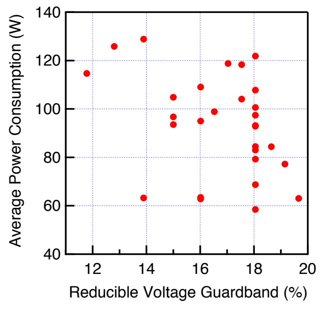

Fig. 4: No correlation between average power and measured reducible voltage guardband.

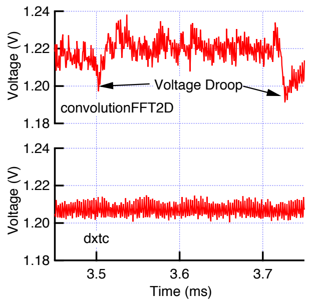

Fig. 5: Measured traces of two benchmarks, comparing IR drop and $L\frac{di}{dt}$ droops.

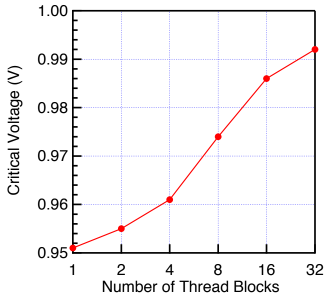

Fig. 6: Critical voltage at an increasing number of active cores when running matrixMul.

III. Characterizing Factors that Impact the Reducible Voltage Guardband

It is important to understand what constrains the extent to which the supply voltage of a GPU can be reduced and how architectural parameters and program characteristics interact with each other and impact a program's critical voltage. We present a measurement-based analysis that serves as the basis for understanding voltage noise in GPUs from this approach. We start by showing that the magnitude of the critical voltage is affected by the $L\frac{di}{dt}$ noise rather than the IR drop of a GPU. Next, we demonstrate that the number of active GPU cores impacts the critical voltage. We also show that increasing the clock frequency can detrimentally affect the critical voltage. Finally, we explain how the critical voltage can be associated with memory versus compute-bound program characteristics.

A. $L\frac{di}{dt}$ Noise

To understand why the programs have different reducible voltage guardbands, we must understand whether the reducible voltage guardband is caused by the IR drop or $L\frac{di}{dt}$ effect [4]. The *static* IR drop is the voltage drop resulting from the resistive component of the power delivery network when the processor consumes high power. $L\frac{di}{dt}$ is a *dynamic* event, resulting from the inductive and capacitive components when microarchitectural activity causes power fluctuations. Reducing the IR drop requires us to lower the GPU's peak power consumption, and therefore may negatively impact the GPU's performance. Because $L\frac{di}{dt}$ is typically a rare transient effect, prior CPU works have shown that optimizing it can significantly boost performance [12]. Alternatively, it can also be used to reduce energy consumption for a fixed frequency.

We find that the majority of the voltage guardband is needed for the inductive voltage noise (i.e., di/dt voltage droop). When the static IR drop is the main cause, the reducible voltage guardband would have a strong correlation with the average power consumption. Fig. 4 shows the relationship between the reducible voltage guardband and the average power consumption. It shows that these two are *not* correlated.

To confirm our analysis, we also measure the GPU processor's voltage trace at the package level using the DAQ. Fig. 5 shows the snapshot of the measured voltage traces for convolutionFFT2D and dxtc. Both programs have a similar power draw of around 115 watts (not shown). However, convolutionFFT2D has a lower reducible voltage guardband than dxtc. Their reducible voltage guardbands are 0.13 V and 0.2 V, respectively. The top graph in Fig. 5 shows the transient voltage droops for convolutionFFT2D. The bottom graph in Fig. 5 shows the measured trace for dxtc, which is more stable.

The trends seen with convolutionFFT2D and dxtc are representative of the other programs. Therefore, we conclude that the inductive voltage droop caused by the GPU processor's current draw variation is the major cause of the lower reducible voltage guardband in some benchmarks. The processor's current draw can vary in accordance with both microarchitectural activity and program characteristics. For instance, microarchitecture stalls can cause voltage droops [12].

B. Number of Cores

Prior work with multicore CPUs demonstrated that the number of active cores could detrimentally impact the reducible voltage guardband due to the nature of "constructive voltage noise interference" [4]. This sort of analysis is yet to be studied in GPUs, which use many simply in-order cores that are significantly less power hungry than traditional out-of-order superscalar processors.

We study the effect of active GPU cores on the critical voltage by conducting an experiment using matrixMul. We use matrixMul because it uniformly exercises all SIMD execution lanes in the GPU without introducing complex behavior (e.g., control divergence). Because we cannot directly control the number of active cores in the GPU, we vary the number of CUDA thread blocks used by the program. It lets us indirectly control the number of active cores.

Fig. 6 shows the critical voltage changes as the number of thread blocks increase. When only one thread block is active, the critical voltage is as low as 0.95 V. When 32 thread blocks are used, the critical voltage increases to 0.99 V. This result implies the guardband would increase as more cores are used.

Granted that matrixMul is a relatively simple application compared to other programs with complex control flow, the observation points to an optimization trade-off for energy efficiency between the number of cores, the GPU critical voltage, and energy efficiency that remains open for exploration.

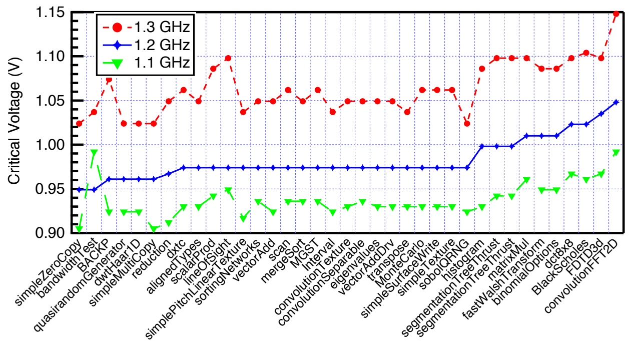

Fig. 7: Critical voltages for operating at different frequencies.

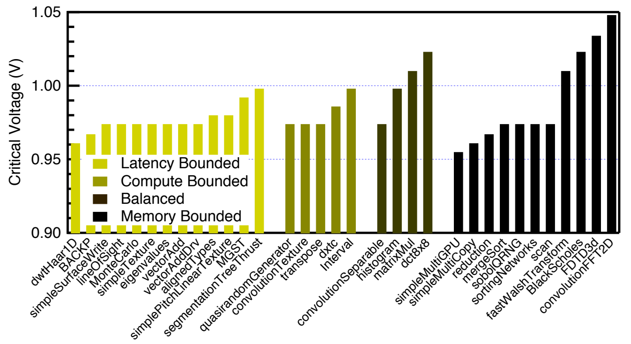

Fig. 8: Critical voltages for the different program types.

C. Clock Frequencies

In the pursuit for low power and high performance, GPUs are employing dynamic voltage and frequency control to lower voltage and achieve power savings when performance is not needed. For example, NVIDIA's GPUBoost dynamically increases the GPU's clock frequency until it hits a predetermined temperature to deliver performance [1].

We find the possibility for an interesting trade-off between the critical voltage and the processor's operating clock frequency. We discover the strong likelihood that we may need to consider the critical voltage when changing frequencies, because a small clock-frequency increase may necessitate a relatively large critical-voltage increase, and this could void the benefits of reducing the voltage guardband and/or the boosted clock frequency.

We measure the critical voltage for the programs under three frequency settings: 1.1 GHz, 1.2 GHz and 1.3 GHz. Fig. 7 shows our results. From our measurements, we make three important observations: First, programs generally need a higher critical voltage at higher clock frequencies due to short cycle time's impact on $L\frac{di}{dt}$ noise. At a higher clock rate, stalls and their impact on voltage droop become more pronounced because current increases and the time duration during which current changes gets smaller.

Second, for a fixed increase in clock frequency (e.g., 100 MHz step), the critical voltage increases superlinearly for nearly all programs. For example, the critical voltage of benchmark convolutionFFT2D increases from 1 V to 1.05 V when frequency increases from 1.1 GHz to 1.2 GHz. When the frequency is increased further to 1.3 GHz, the critical voltage increases by a larger amount to 1.15 V. The trend applies generally to almost all the programs we consider.

Third, the exact magnitude of the increase can vary across the programs. For some programs, such as BACKP, scalarProd and lineOfSight, the critical voltage can increase much larger than the other benchmarks when frequency changes. When frequency increases from 1.2 GHz to 1.3 GHz, the critical voltage increases sharply for both BACKP and scalarProd. However, in the case of sobolQRNG, the increase is smaller. To understand these differences, we need to examine the programs and understand their inherent workload characteristics.

D. Program Characterization

We demonstrate that the characteristics of a program impact its critical voltage. Specifically, we show the extent to which memory- versus compute-bound programs affect the critical voltage. We find that memory-bounded programs typically have a higher critical voltage, which is possibly caused by stalling behavior even though GPU architectures are aggressively designed to mask memory stalls via massive multithreading. A typical GPU can support over 20,000 threads.

We categorize the programs into four different types using the NVIDIA visual profiler [13]. The four types of programs we study are groups as such: memory bounded, whose execution time is bounded by memory bandwidth; compute bounded, whose execution time is bounded by the GPU's computational capabilities; latency bounded, which do not have enough threads to run on the GPU hardware and thus have very low utilization of both compute units and main memory bandwidth; and balanced, which is the ideal program to run on a GPU because it has a high utilization rate on both compute units and memory bandwidth.

Fig. 8 shows the critical voltage for the different program types. Memory-bounded programs tend to have a higher critical voltage (i.e., larger voltage droops). Balanced programs show moderate critical voltage. Compute- and latency-bounded programs tend to have lower critical voltage (i.e., smaller voltage droops).

Prior work in the CPU domain has shown that two conditions are required for large voltage droops to occur: regular microarchitecture stalls, and synchronized stalls among multiple cores. Both of these conditions can explain why the memory-bounded kernels show a large droop. Memory-bounded kernels have stall behavior that is caused by the memory subsystem, and these kinds of stalls tend to synchronize because of contention at the memory subsystem level. Although latency-bounded programs also have stall behaviors, the stalls are likely not aligned due to the lack of contention among common resources. Compute-bounded programs either have stable power draw or unsynchronized stalls.

In the future, it may be worthwhile to explore GPU kernel-level characteristics. It may also be worthwhile to understand how explicit program characteristics such as barrier synchronizations, etc. impact the reducible guardband magnitude.

IV. Conclusion

We demonstrated that we can achieve energy-reduction benefits as high as 25% by pushing the Kepler GPU's core supply voltage to its limit. The challenge for leveraging this opportunity lies in understanding what impacts the reducible voltage guardband. We find that voltage guardband of GPUs is mainly caused by $L\frac{di}{dt}$ noise, and the critical voltage depends on workload characteristics. We also show how microarchitecture-level parameters, such as the number of active cores and core frequency, impact the reducible voltage guardband. We believe that there is a large potential for this work, and it encourages us to further understand the GPU voltage guardband's interactions with architecture-level parameters as well as GPU programs' characteristics.

References

[1] NVIDIA Corporation, "NVIDIA CUDA Programming Guide," 2011.

[2] 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 *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2013.

[3] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie, "Comparison of Split-Versus Connected-Core Supplies in the POWER6 Microprocessor," in *International Solid-State Circuits Conference (ISSCC)*, 2007.

[4] V. Reddi, S. Kanev, W. Kim, S. Campanoni, M. Smith, G.-Y. Wei, and D. Brooks, "Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-Guided Thread Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2010.

[5] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter, "Active Management of Timing Guardband to Save Energy in POWER7," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2011.

[6] V. J. Reddi, M. Gupta, G. Holloway, M. D. Smith, G.-Y. Wei, and D. Brooks, "Predicting voltage droops using recurring program and microarchitectural event activity," *IEEE micro Top picks*, 2010.

[7] M. D. Powell and T. Vijaykumar, "Pipeline damping: a microarchitectural technique to reduce inductive noise in supply voltage," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003.

[8] V. Reddi, M. Gupta, G. Holloway, G.-Y. Wei, M. Smith, and D. Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2009.

[9] M. S. Gupta *et al.*, "An event-guided approach to handling inductive noise in processors," in *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*, 2009.

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

[11] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2009.

[12] M. S. Gupta, K. K. Rangan, M. D. Smith, G.-Y. Wei, and D. Brooks, "Towards a software approach to mitigate voltage emergencies," in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

[13] NVIDIA Corporation, "NVIDIA Visual Profiler," 2013.