AUDIT: Stress Testing the Automatic Way

Youngtaek Kim Lizy Kurian John
Department of Electrical & Computer Engineering
The University of Texas at Austin
Austin, TX, USA
young.kim@utexas.edu ljohn@ece.utexas.edu

Sanjay Pant[1] Srilatha Manne[2] Michael Schulte[3]
W. Lloyd Bircher[3] Madhu S. Sibi Govindan[3]
Advanced Micro Devices, Inc.
[1]Fort Collins,CO, [2]Portland,OR, and [3]Austin,TX, USA
{sanjay.pant, srilatha.manne, michael.schulte,
lloyd.bircher, sibi.govindan}@amd.com

Abstract—**Sudden variations in current (large di/dt) can lead to significant power supply voltage droops and timing errors in modern microprocessors. Several papers discuss the complexity involved with developing test programs, also known as stressmarks, to stress the system. Authors of these papers produced tools and methodologies to generate stressmarks automatically using techniques such as integer linear programming or genetic algorithms. However, nearly all of the previous work took place in the context of single-core systems, and results were collected and analyzed using cycle-level simulators.**

In this paper, we measure and analyze di/dt issues on state-of-the-art multi-core x86 systems using real hardware rather than simulators. We build on an existing single-core stressmark generation tool to develop an AUtomated DI/dT stressmark generation framework, referred to as AUDIT, to generate di/dt stressmarks quickly and effectively for multi-core systems. We showcase AUDIT's capabilities to adjust to microarchitectural and architectural changes. We also present a dithering algorithm to address thread alignment issues on multi-core processors. We compare standard benchmarks, existing di/dt stressmarks, and AUDIT-generated stressmarks executing on multi-threaded, multi-core systems with complex out-of-order pipelines. Finally, we show how stress analysis using simulators may lead to flawed insights about di/dt issues.

Keywords- di/dt; inductive noise; stressmark generation; voltage droop; power distribution network; low power; genetic algorithm; hardware measurement

I. INTRODUCTION

Reliable operation is a fundamental requirement of processor design. The processor must work correctly across a range of applications regardless of process variations, voltage variations, environmental noise, and the aging of the system. Voltage margins are introduced to compensate for potential supply voltage fluctuations in the system. Fluctuations caused during program execution must stay within allowed margins, as shown in Fig. 1. These margins need to be designed carefully to be power-efficient and prevent malfunctions from program-induced voltage fluctuations.

Specialized benchmarks, referred to as stressmarks, are used to study the susceptibility of processors to voltage fluctuations. Stressmarks may or may not be used to set the voltage margins; however, they are necessary to develop an understanding of the susceptibilities of the system being analyzed. Stressmarks are either collected from benchmarks that have produced high di/dt stresses in the past, or they are specially designed to induce voltage fluctuations in microprocessors.

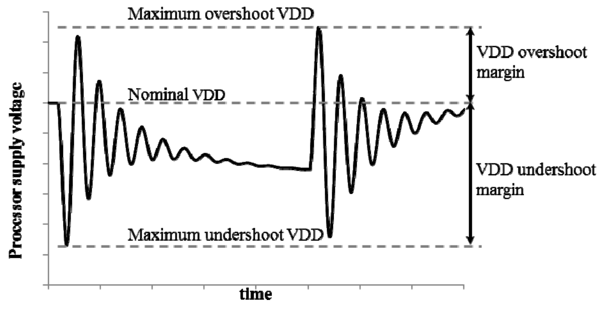

Figure 1. Voltage margins to manage fluctuations.

Stressmarks are difficult to generate manually; past work addressed the complexities involved with stressmark generation and developed tools and methodologies to generate stressmarks automatically [11][12][13]. Other work addressed mitigating droops that occur due to various architectural events [3][5][8][9][10][16][21] or leveraging the margin between typical benchmarks and stressmarks to operate with reduced voltage margins [7][15][22]. In these papers, many of the results were collected and/or analyzed using cycle-level simulators enhanced with power models. In addition, except for a few papers [6][16][23], the authors did not consider or investigate the complexities of multi-core processors with multi-threaded resources or operating system interactions.

In this paper, we address di/dt analysis and stressmark generation using hardware rather than simulators. We use state-of-the-art, multi-threaded, multi-core x86 hardware with complex out-of-order pipelines to produce a detailed analysis of voltage droops and failure points for standard benchmarks and stressmarks. We expand the single-core di/dt stressmark generation tool by Kim and John [13] to develop an AUtomated DI/dT stressmark generation framework, referred to as AUDIT, that efficiently generates stressmarks for multi-core x86 systems using simulators or hardware. We compare the effectiveness of AUDIT stressmarks against other manually generated stressmarks using state-of-the-art hardware. We also show the impact of mitigation mechanisms in hardware and AUDIT's flexibility in handling microarchitectural and architectural changes. Finally, we show how simulators fail to capture the nuances of di/dt stresses and their behavior by demonstrating that (1) droop measurements do not always correlate to failure points in real hardware, (2) OS interference can influence how loops align in multi-core systems, and (3) alignment and synchronization that occur in simulators may not be repeatable in real hardware due to natural perturbations in how the system operates.

In the rest of this paper, Section 2 discusses voltage droops, droop management techniques, and manual di/dt stressmark generation. Section 3 presents the AUDIT stressmark generation framework and the dithering algorithm for multi-core stressmark alignment. Section 4 details our experimental set-up. Section 5 presents results and analysis of droop and failure characteristics of standard benchmarks, manual stressmarks, and AUDIT stressmarks. Section 6 discusses related work, and Section 7 concludes the paper.

II. Background

The power distribution network (PDN) of a typical microprocessor consists of inductive and resistive elements on the motherboard (MB), package, and die. The parasitic resistance of the network causes a droop (IR drop) in the power supply proportional to the current drawn from the network. In addition, the inductance in the network causes undershoots and overshoots in the power supply (referred to as the di/dt droop), which depend on the rate of change of the load-current.

To mitigate the inductive noise in the power supply, decoupling capacitance (decap) is added at different locations in the PDN (Fig. 2). The series combination of parasitic inductance (L) and decap (C) results in various resonance frequencies ($1/2\pi\sqrt{LC}$) in the network, as shown in the frequency and time domains in Fig. 3. While single voltage droops caused by non-repeating di/dt events may be harmful, droops that repeat at the resonance frequency of the system grow to high amplitudes and are much more likely to cause catastrophic failures.

The prominent resonance frequencies shown in Fig. 3 are the first droop resonance due to the interaction of package and on-die inductance (L_{pkg2} + L_{die}) with on-die decap (C_{die}), the second droop resonance due to the interaction of socket and package inductance (L_{pkg1}) with package decap (C_{pkg}), and the third droop resonance due to the interaction of board inductance (L_{MB}) with decap on the board (C_{MB}). A periodically varying load can induce one or more of these resonances and cause excessive undershoots and overshoots. Although second and third droop resonances can also affect the reliability of the system, they are typically smaller in magnitude than first droop resonance [14] and are not evaluated in this work.

The first droop resonance is a strong function of package inductance (L_{pkg2}) and C_{die}, and is typically in the range of 50–200MHz. First droops can be mitigated by explicitly

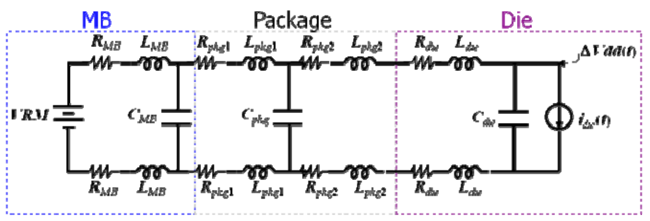

Figure 2. Simplified model of a power distribution network.

adding decap on the die [19]. However, there are limits to the feasibility of this approach due to area constraints and the leakage of the decap. Several techniques that limit the rate of change of activity in the processor are effective in suppressing first droops [8][9][10][21], but they may have a negative impact on performance.

If a single high-di/dt event occurs due to the machine executing a pattern of low-power instructions followed by a pattern of high-power instructions, there will be a droop in supply voltage, but the droop will taper off quickly, as shown on the left side of Fig. 4. However, a pattern that repeats periodically at the resonant frequency of the PDN (right side of Fig. 4) will build in amplitude and generate a larger droop than a single event, thereby increasing the risk of system failure. We cover first droop excitation and first droop resonance in our analysis.

III. Automatic Stressmark Generation

Stressmarks are common in both academia and industry. Several methods for automation have been proposed because generating stressmarks is a tedious and time-consuming process for designers [11][12][13]. Although such stressmarks may be representative of worst-case behavior, understanding this behavior is important when determining voltage and frequency margins and developing droop mitigation mechanisms to ensure reliable operation. They expose sensitivities and critical paths in the pipeline, as we will show in Section 5.

Automatic stressmark generation using a genetic algorithm (GA) was first proposed in [11] and expanded to cover di/dt resonance issues explicitly by Kim and John in [13]. This paper expands upon that work by:

- generating di/dt stressmarks for complex, multi-core x86 processors rather than single-core Alpha processors,
- utilizing real hardware rather than simulation to analyze full applications under realistic conditions,
- presenting a dithering algorithm to produce alignment across threads in a multi-core environment,
- extending AUDIT to automatically detect the resonant frequency of the system and generate both first droop excitation and first droop resonance stressmarks, and
- generating AUDIT stressmarks for varying processors, processor configurations, and operating conditions.

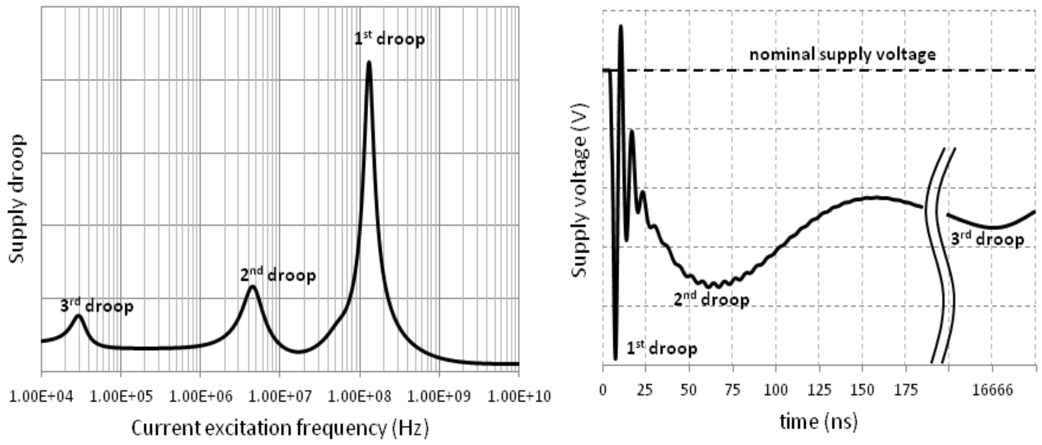

Figure 3. First, second, and third resonance droops in the frequency and time domains.

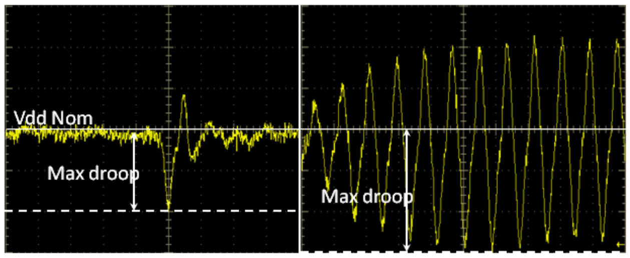

Figure 4. First droop excitation and first droop resonance generated using the AUDIT framework.

Fig. 5 shows the basic framework for AUDIT. AUDIT takes as input the instructions used to generate the stressmark and some control parameters such as the cost function and exit conditions. This information is fed to a code generator to produce a population of potential stressmarks. The initial population of stressmarks either can be generated randomly or seeded with existing benchmarks or stressmarks to improve the convergence rate.

Fig. 5 includes two possible paths for stressmark generation, simulation and hardware. With the simulation path, the voltage droops of generated instruction sequences are evaluated using a cycle-accurate simulator that produces current draw information followed by SPICE simulation. This path is most appropriate when hardware for performing di/dt stressmark generation is not available. With this approach, the assembly code instruction sequence is compiled into a simulator-friendly format (e.g., x86 binaries). The compiled code is executed on the cycle-accurate simulator and every cycle the simulator calculates the current draw of the processor based on the activity of internal modules of the processor. This methodology is similar to that used in [5][10][21]. AUDIT converts the per-cycle current profile into a current sink in HSPICE simulation using a lumped RLC model of the PDN. The HSPICE simulation produces a series of voltage droops over time from which the maximum voltage droop can be obtained.

With the hardware path, the stressmarks are run on a processor board and measurement tools capture voltage droops, power dissipation, and any other information necessary to evaluate the cost function of the stressmark. The stressmarks and their associated cost values are fed to the GA for further refinement until the exit conditions are met (e.g., the maximum voltage droop produced by AUDIT does not increase for several generations[1]). In prior research [13], we used the simulation-based path, whereas in this paper, the hardware-based path is used.

We use the same methodology as in [13] to generate the opcode sequence. However, there are some additional complexities because we are using x86 hardware and generating stressmarks for multi-core systems. First, we observe that data values used for the stressmark have a measureable impact on the final droop values, on the order of 10%. To take data values into account, we use an alternating set of values that guarantee maximum toggling between one instruction and the next executing on the same functional unit. Second, the resonance frequencies of the system can vary across different boards or even within the same board if the components of the board change (e.g., using a different processor on the same board, as is done later in the paper). Therefore, we extend AUDIT to do a sweep for the resonance frequency before attempting to generate a first-order resonant droop.

To determine the resonance frequency, AUDIT constructs a trivial stressmark consisting of a loop of high-power instructions and NOP instructions. It varies the number of cycles in the loop to determine the length that produces the worst-case droop. The number of cycles in the loop that produces the worst-case droop exercises the

[1] The cost function provided to AUDIT can vary. Although we focus on maximizing voltage droops in this paper, other, more complex cost functions such as maximizing the droop while minimizing the average power or maximizing the droop while exercising sensitive paths in the microarchitecture are also feasible and easy to implement.

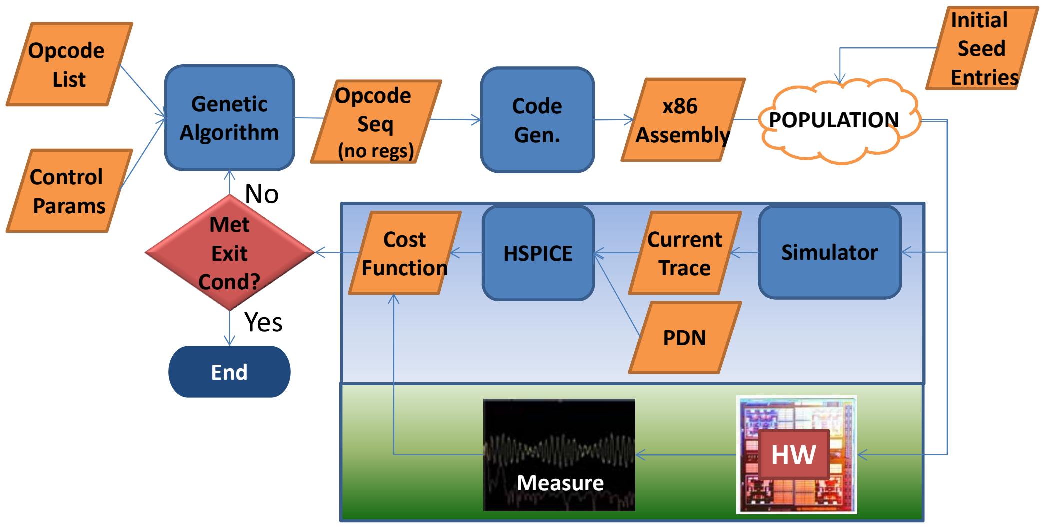

Figure 5. AUDIT framework for di/dt stressmark generation using simulators and hardware.

resonant frequency of the processor. Finally, we have to deal with the complexities of multi-core stressmark generation.

A. *Thread alignment in multi-core systems*

As noted in other papers [3][6][16][23], multiple threads running simultaneously can have a constructive or destructive impact on droops. If the threads align correctly, they produce significantly larger droops than without alignment. At first glance, thread alignment would seem to be a low-probability event in multi-core machines with complex, out-of-order cores and shared and non-shared resources. However, our analysis shows that alignment occurs relatively often when the stressmark consists of short loops due to natural perturbations in the threads caused by OS thread scheduling. We refer to this phenomenon as natural dithering.

Fig. 6 shows an example of natural dithering over the course of 100 ms when running a four-threaded resonant stressmark in which the threads are the same and consist of short loops. Each major grid point represents 10 ms and the y axis shows measured processor voltage (V_{dd}) values using a 100 megasamples/second (MS/s) sampling rate.

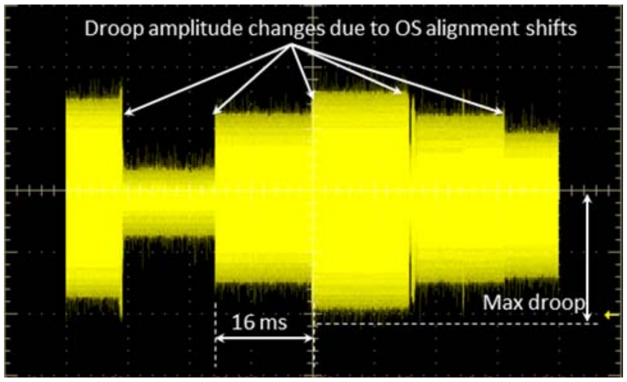

Figure 6. Scope shot of natural dithering due to OS interactions for resonant stressmark over a period of 100 ms.

Approximately every 16 ms, which corresponds to the OS timer tick on Windows systems, V_{dd} variability changes. When the threads align constructively, as is the case near the center point of the scope shot, the droop is maximized.

This data shows that small, repetitive loops occurring across multiple threads at the same time can result in significant di/dt stresses in the system due to natural dithering resulting from OS interaction. This phenomenon would have been difficult, if not impossible, to observe in a simulation environment. This type of behavior is more likely to occur in certain high-performance computing applications that consist of short, repeated loops.

Relying on OS behavior to align threads is not a reliable method to determine the worst-case droop. Hence, we propose a dithering algorithm that guarantees a worst-case droop within a fixed amount of time once OS interrupts are disabled.

B. *Dithering algorithm for guaranteed alignment*

Fig. 7 shows a periodic stress pattern with high- and low-power portions of duration H and L cycles, respectively. This waveform meets the requirements of an ideal di/dt-inducing resonant pattern described in Section 2. This periodic pattern is repeated for M cycles to produce a large resonant droop. The goal of the dithering algorithm is to guarantee that for C cores, the stressmarks running on each core align across all C cores for at least M cycles. Note that a first droop excitation is different in that it requires a low region followed by a high region where the sum of the regions is not necessarily periodic at the resonance frequency.

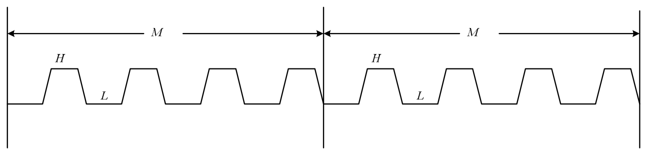

Figure 7. Periodic activity waveform for inducing power supply resonance and large voltage droops.

For a high-low sequence of length $H+L$ cycles running on C cores, the misalignment in cores 1 through $C-1$ can be represented as a $C-1$ dimensional variable $x = (x_1, x_2, \ldots, x_{C-1})$, where x_i □ $\{0,1,\ldots,L+H-1\}$. Core 0 is considered the reference core. The search space for perfect alignment of all cores is therefore $(L+H)^{(C-1)}$ possible alignments. This search space can be fully traversed in $M\times(L+H)^{(C-1)}$ cycles, where M is the number of cycles required to cause and sustain supply droop resonance.

The dithering algorithm uses the following NOP padding procedure to align the threads and achieve resonance in a processor with C cores:

- Core 0: Apply no dithering and no extra padding of NOPs. Core 0 simply executes the periodic low-high activity sequence shown in Fig. 7 repetitively.
- Core c, where $1 \leq c \leq C-1$: Apply one cycle worth of NOP padding every $M\times(L+H)^{(c-1)}$ cycles.

The maximum number of cycles to guarantee alignment is $M\times(L+H)^{(C-1)}$.

As long as the number of processors is reasonably small, the alignment algorithm works well. However, the time required for alignment becomes prohibitively large for more than four cores. For example, on a 4-GHz system with $L+H=24$ and $M=24\times40=960$, the time required to align four cores is 3.3 ms, but eight cores require 18.35 minutes. The alignment must be done for each candidate stressmark in each generation of the GA.

To expand dithering to many-core systems, we use an approximate algorithm that sets a bound on the maximum misalignment between threads. Assume that the maximum mismatch allowed among the activities of different cores is δ cycles. Then, $L+H$ is chosen such that it is a multiple of $(\delta + 1)$ and $(L+H)\times f$ is close to the resonance frequency of the PDN, where f is the operating frequency of the system.

The search space for alignment of all cores within the maximum allowed mismatch of δ cycles then becomes $[(L+H)/(\delta + 1)]^{(C-1)}$, which can be fully traversed in $M\times[(L+H)/(\delta + 1)]^{(C-1)}$ cycles. The dithering algorithm proceeds as before; however, for core c, where $1 \leq c \leq C-1$, $(\delta +1)$ cycles worth of NOP padding is applied every $M\times k^{(c-1)}$ cycles, where $k = (L+H)/(\delta + 1)$. If we use a δ of 3 in the previous example of eight cores, the maximum time required to reach alignment with the approximate algorithm shrinks from 18.35 minutes to 67 ms per candidate stressmark.

C. *Expanding AUDIT for multi-core hardware*

The stressmark solution space for AUDIT is a function of the number of cycles in the repeated loop (the loop length), the issue width of the processor, and the number of instructions being evaluated for code generation. The combination of loop length, issue width, and the number of instructions can result in a large solution space. The loop length for first droop resonance is determined by the resonance frequency, which can result in a large solution space. For example, a 3-GHz processor with a resonance frequency of 50 MHz has a loop length of 60 cycles. Assuming a four-wide processor, this results in 240 instruction slots for AUDIT to schedule. In addition, dithering increases AUDIT's run time by requiring a sweep through a large number of alignments for each stressmark, as discussed in Section 3.B.

To converge in a reasonable time (we define reasonable time as being a few hours), we modified AUDIT to use a hierarchical generation policy. First, AUDIT separates each member of the population into a high-power (HP) and low-power (LP) region. Initially, the LP region consists of NOPs. Second, AUDIT breaks the HP region into S replicated sub-blocks of length K. For example, a 24-cycle HP loop can be composed of four (S = 4) sub-blocks of length six cycles (K = 6). The GA algorithm in AUDIT generates the instructions for each subsection, and the full stressmark -- composed of an HP region of S sub-blocks of length K and an LP region of NOPs -- is evaluated in hardware using the dithering algorithm.

At the end of the AUDIT run for the HP region, we have a stressmark that has been synthesized to produce high power for the HP region of the stressmark. We also evaluated using AUDIT to generate the LP region of the stressmark using long-latency operations with dependencies as proposed in [10]. However, for the system we evaluated, a sequence of NOPs produced comparable power values to a sequence of long-latency, dependent operations. NOPs are designed to be very low-power instructions in our experimental processor, so the rest of the evaluation uses NOPs for the LP portion of the stressmark.

We compared the hierarchical AUDIT implementation to that proposed in [13] and found sub-blocking provided faster convergence as well as better results -- 19% higher droop in less than five hours compared to a 30-hour run without hierarchical generation.

IV. EXPERIMENTAL SET-UP

The work in this paper targets the latest multi-core x86-64 processors due to their widespread use. The primary processor utilized for this study consists of four AMD Bulldozer modules with 2 MB of dedicated L2 cache per module and an 8 MB shared L3 cache [4][25].

Each Bulldozer module can execute two threads via a combination of shared and dedicated resources [2]. The front-end and floating-point logic is shared between two threads on the same module; however, the rest of the core

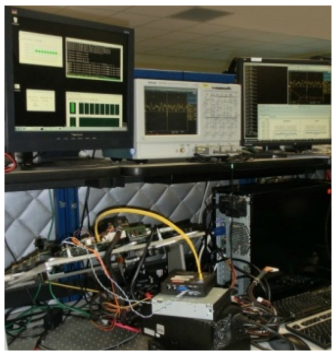
Figure 8. Experimental set-up.

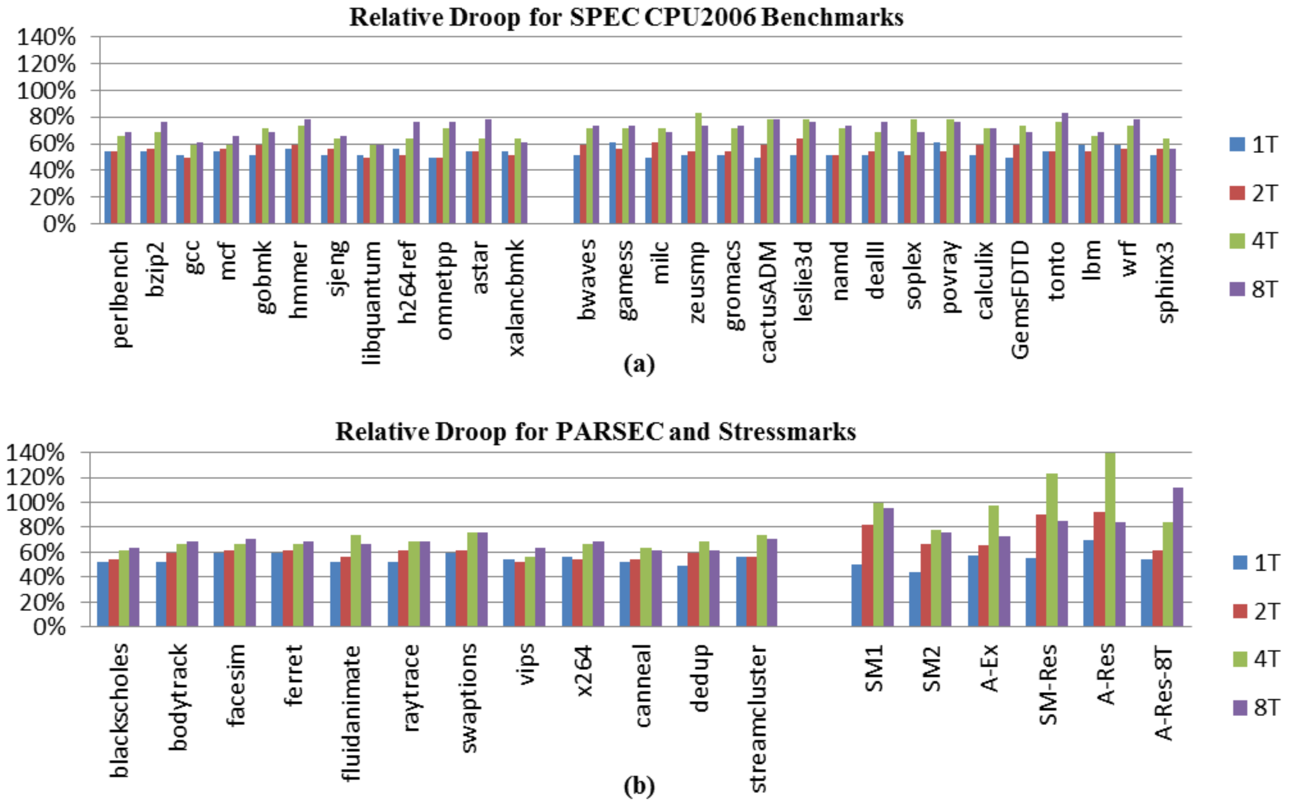

Figure 9. Hardware measurements of droop (relative to 4T *SM1*) for SPEC CPU2006, PARSEC, and stressmarks.

components (integer and retire logic, load/store unit, first-level TLB, and first-level cache) are separate. Each thread can issue four integer instructions per cycle, however, the two threads together can only issue four floating point instructions per cycle due to the sharing of the floating point units. A thread can have a maximum IPC of four. A more detailed description of the Bulldozer module and architectural features is given in [2]. In later experiments, we also replace the Bulldozer-based processor with an older-generation 45-nm AMD Phenom™ II X4 Model 925 processor to showcase AUDIT's ability to adapt to different systems and requirements.

AUDIT's code generation methodology is able to utilize all x86 instruction types, including integer, floating-point, and SIMD. General-purpose registers and 64-bit and 128-bit media registers are used for source and destination operands. Assembly code instructions are generated in NASM format and are compiled with NASM 2.09.08 [24].

Our experimental set-up is shown in Fig. 8. We measure voltage droops on hardware with a Tektronix TDS5104B oscilloscope and a 1.7-GHz Tektronix P6248 differential probe for triggering on large voltage droops. The probing points for the power supply voltage are attached to the package and on-die connection to enable accurate voltage droop measurements. The oscilloscope triggers and records the di/dt events at a sampling rate of 5 gigasamples/second (GS/s). We used Windows® 7 OS for SPEC CPU2006 benchmarks and stressmarks, and Red Hat Enterprise 6 for PARSEC [1] benchmarks.

V. RESULTS AND ANALYSIS

This section covers the results obtained for multi-core stressmark generation. We compare and analyze standard benchmarks, existing stressmarks, and AUDIT-generated stressmarks. For each benchmark (stressmark), we present its maximum voltage droop and analyze the processor's ability to operate under degraded voltage conditions. We also present results showing AUDIT's ability to adapt to microarchitectural and architectural changes.

A. Droop and failure analysis

Fig. 9 shows the maximum droop measured from running SPEC CPU2006 benchmarks, PARSEC multi-threaded benchmarks, and a set of existing and AUDIT-generated stressmarks in configurations of one-, two-, four-, and eight-thread runs (1T, 2T, 4T, and 8T). For SPEC CPU2006 and stressmark runs with multiple threads, the program is replicated and executed on multiple cores, similar to SPECrate. Given the shared nature of the cores in the Bulldozer module, higher voltage droops occur for a given number of threads when threads are spatially distributed across modules. The evaluation processor has four Bulldozer modules, each with two cores. Hence, for the 1T, 2T, and 4T runs, each thread is assigned to a different module. For the

8T runs, there are two threads assigned to each module. All droop results are shown relative to the 4T *SM1* stressmark, and higher numbers indicate larger droops. The values are measured with the load line of the voltage regulator module (VRM) disabled to remove any load-line droop effects [15]. Hence, the results show the droop due to di/dt stresses only.

The multi-threaded AUDIT stressmarks (*A-Ex* and *A-Res*) and the hand-generated resonant stressmark *SM-Res* use the dithering methodology described in Section 3.B to align the threads for a worst-case voltage droop. The 2T and 4T configurations use the exact algorithm, and the 8T configuration uses the approximate algorithm with a δ of 3.

Unfortunately, the dithering methodology is not easily applicable to SPEC CPU2006 benchmarks or the PARSEC suite because they do not consist of a regular, repeatable loop that can be shifted to produce alignment between the threads. Although the lack of dithering for SPEC CPU2006 results in a smaller droop than is theoretically possible with ideal alignment [23], it also reflects the reality of multi-processor execution in which the natural misalignment between threads may counteract some worst-case stress generating behavior. Prior work shows that multithreaded programs such as those in the PARSEC suite have synchronization points that could align the threads and produce opportunities for high di/dt stresses [16].

1) *Standard benchmarks*

Fig. 9 shows that, in general, the magnitudes of the voltage droops increase with the number of threads for 1T, 2T, and 4T configurations. The 8T configurations do not always follow this trend due to multi-threading in the Bulldozer module (explained further in Section 5.A.2).

As noted in Section 2, one way to generate a significant droop is to have a large change in activity from idle to full execution. For high-performance pipelines, such a change in activity occurs naturally with certain pipeline events, such as pipeline recovery after a branch misprediction stall or high execution activity after a load miss resolves [22]. These events are commonplace in complicated pipelines, and how they interact with each other in a multi-threaded scenario dictates how large a droop they produce. Destructive interference may occur between threads in a multi-core system such that when one thread is in a high-power state others are in a low-power state. Reddi et al. describe the issue of thread misalignment for SPEC CPU2006 benchmarks, examine constructive and destructive interference in a dual-core system, and discuss co-scheduling threads to reduce voltage droops [23].

The PARSEC multi-threaded benchmark suite could have alignment between threads through its use of synchronization primitives. The expectation was that we would see higher droops due to the natural alignment resulting from barrier operations in benchmarks such as *fluidanimate* and *streamcluster* as discussed in [16]. However, our results show no significant difference in droops between PARSEC and the SPEC CPU2006 suite.

To evaluate further, we designed a barrier stressmark that repeatedly synchronizes on a barrier operation and then runs the high-power virus in a 4T configuration. We expected this to result in a large voltage droop due to all cores being aligned and idle at the barrier operation followed by high activity on the cores. The resulting droop, however, was not significant. On further examination, we noticed that a natural misalignment occurs between the cores when released from a barrier. On the Bulldozer module, there is no explicit mechanism to synchronize the barrier release signal, and the signal naturally reaches each core at different times based on from where in the memory hierarchy the core receives its data. This perturbs the start of activity across the cores by enough cycles to dampen the first droop excitation resulting from the synchronization operation.

The authors in [16] examined a different x86 processor that may have different characteristics. In addition, they use fluctuations in average power estimated at intervals of 1 ms on hardware as a proxy for expected di/dt variations. This may capture third droop excitations, but not first droop excitations that occur over the course of nanoseconds. Our measurement technique is capable of identifying the high-frequency first droop variations in voltage. Hence, the worst-case droops in PARSEC are most likely the result of the same microarchitectural events that align across multiple threads in the SPEC CPU2006 suite. The authors in [16] also note that barriers are not the only cause of high power swings and point to microarchitectural events such as long-latency cache misses followed by bursts of activity as other potential inducers of high droop. Furthermore, the effect in [16] is pronounced for cases with 32 threads, whereas our experiments did not include such configurations.

2) *Stressmarks*

Fig. 9(b) also shows the results for various stressmarks, either manually collected or hand-generated (*SM1*, *SM2*, and *SM-Res*) or automatically generated by AUDIT (*A-Ex* and *A-Res*). *A-Ex* is a first-droop excitation stressmark, and *A-Res* is a first-droop resonant stressmark. *SM1* and *SM2* contain both single-droop and resonant excitations, and *SM-Res* is a hand-generated resonant stressmark. The manual stressmarks are the result either of past di/dt issues or a non-trivial design effort (on the order of a week per stressmark) from a highly skilled engineer with detailed knowledge of the pipeline architecture. The goal of AUDIT is to generate similar or better stressmarks without detailed knowledge of the pipeline in question.

To produce the *A-Ex* and *A-Res* stressmarks in Fig. 9(b), we instructed AUDIT to generate a homogeneous stressmark with four identical threads, one assigned to each module. For the resonant stressmark, we use a high-power sub-block of length six cycles and repeat as many times as necessary to produce the high-power region. The low-power region of the stressmark, for the reasons noted in Section 3.C, consists of NOPs. The stressmark generation takes less than five hours to complete without human intervention.

With the exception of *SM2*, all stressmarks produce significantly greater droops than the standard benchmarks. As will be shown in Section 5.A.4, *SM2* is still a viable stressmark because it exercises the sensitive paths on the processor. The two resonant stressmarks (*SM-Res* and *A-Res*) produce significantly larger droops than all other stressmarks

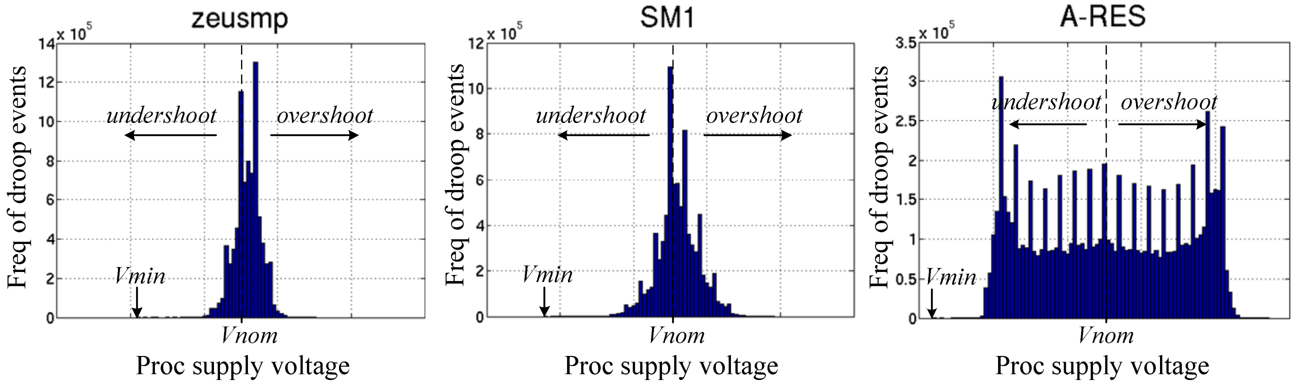

Figure 10. Frequency of droop events.

for the reasons described in Section 2. Both AUDIT-generated stressmarks (*A-Ex* and *A-Res*) produce droops that are either comparable or greater than that of the existing stressmarks. This highlights AUDIT's ability to produce results that are comparable to well-engineered stressmarks that require significantly more effort and knowledge to generate.

The stressmarks produce larger droops for the 4T case than for the 8T case. All stressmarks contain some amount of floating-point instructions, and the FPU is shared between the two threads in each Bulldozer module in the 8T runs. This results in interference between the threads; this shifts the loop lengths, making it difficult to align the first droop excitation across the threads or to oscillate at the resonant frequency. The same interference may not exist in the standard benchmarks depending on the density of floating-point operations and how the threads align.

The *A-Ex* and *A-Res* stressmarks are generated using four homogeneous threads assigned one to each module. Hence, the GA in AUDIT is not trained to deal with the shared FPU in the 8T run. To test our hypothesis, we ran AUDIT again to use eight homogeneous threads, with two threads per module, to generate a new stressmark (*A-Res-8T*). The resulting data is shown in Fig. 9(b). The 8T results for *A-Res-8T* are significantly better than the 8T results for *A-Res* or *SM-Res*. However, the 1T, 2T, and 4T results suffer for the same reason that the 8T run suffers for the other stressmarks -- because the characteristics assumed for the stressmark generation are not valid in some of the multi-threaded configurations. These results show that (1) system characteristics (such as shared resources) must be considered when generating stressmarks, (2) one type of stressmark may not apply to all configurations in a multi-core system, and (3) AUDIT is robust and flexible enough to find patterns that can exercise the characteristics of the system being evaluated with minimal manual intervention.

3) *Droop probability*

Fig. 9 shows the worst-case droop for the benchmarks and stressmarks. However, it does not show how often the droop occurs. The more frequently a large voltage droop occurs, the more likely it is to result in a catastrophic failure. Not only does first droop resonance produce larger droops than first droop excitation (see Fig. 4), it also produces more such events.

In Fig. 10, we use our hardware measurement tools to produce a histogram of voltage droops for *zeusmp*, *SM1*, and *A-Res*. Each plot contains 8 million samples. The x axis shows the measured V_{dd} and the x axis range is the same for all figures. The y axis shows the number of samples for the given V_{dd}. Values to the left (right) of center indicate voltage droops (overshoots).

The *zeusmp* benchmark has the least variation in voltage, as expected from the results in Fig. 9. *SM1* has a larger range of measured V_{dd}, yet the largest number of samples is centered at the nominal V_{dd} with a sharp reduction for lower voltages. There are spikes along the way, most likely due to code regions with resonant behavior, but the application has a long tail for both droops and overshoots. The resonance stressmark has the opposite characteristic with the highest number of events occurring near the worst-case droop values. Both stressmarks have a tail of low-probability droop events, but what dictates the failure point of these benchmarks is the higher-probability droop events near the tail. With hardware measurement, we can evaluate these characteristics across the entire run of the program, which is not possible in simulation. In the next section, we evaluate how these droop characteristics translate into failure points for each application.

4) *Droop vs. voltage at failure*

The size of the maximum voltage droop is an indirect indicator of the voltage operating margin of the program. The ultimate test is to determine the point at which failure occurs for each configuration. In the next experiment, running 4T configurations for two standard benchmarks with the largest droop (*swaptions* and *zeusmp*) and the stressmarks, we reduce the operating voltage in decrements of 12.5 mV until failure occurs. The higher the voltage at failure, the better the program is at stressing the system.

Table 1 shows the results relative to *A-Res*, which fails at the highest voltage (V_F). The other resonant stressmark, *SM-Res*, fails at a value 12 mV lower. The next to fail are the other stressmarks, with *zeusmp* and *swaptions* failing last as V_{dd} is reduced.

As discussed earlier, the largest droops in the standard benchmarks are the result of a first droop excitation that tapers off quickly, as shown in the left side of Fig. 4. Hence, they may or may not cause system failure depending on whether the droop occurs when critical paths are being exercised. *A-Ex* also generates a first droop excitation, but it is large enough to cause a failure at higher voltages. *SM1* and *SM2* have both first droop excitation and first droop resonance, and they fail at a higher voltage than the standard benchmarks. This is expected for *SM1* due to its large droop. *SM2*, however, has a droop that is comparable to the standard benchmarks yet is more sensitive to the voltage levels. This is because *SM2*, unlike the benchmarks, is designed to exercise sensitive paths in the architecture.

What these results show is that the voltage droop is one indicator of potential failure, but not the only one. This insight would be difficult to gather from a cycle-accurate simulator that does not detect droop-induced system failures, and benchmarks such as *SM2* would be discarded as potential stressmarks.

As currently implemented, AUDIT's cost function for selecting successful populations is based on the measured droop in the system. However, it is trivial to adjust the cost function to reward the use of certain types of instructions that exercise critical paths if they are known. The key is that AUDIT is agile enough to manage these changes with little effort.

TABLE I. VOLTAGE AT FAILURE RELATIVE TO A-RES 4T FAILURE POINT.

A-Res	SM-Res	SM1	A-Ex	SM2	zeusmp	swaptions
V_F	V_F – 12 mV	V_F – 62 mV	V_F – 75 mV	V_F – 87 mV	V_F – 125 mV	V_F – 125 mV

5) *AUDIT loop analysis*

To determine how AUDIT is able to produce large droops, we analyzed the main loop of the resonant stressmarks *SM-Res* and *A-Res*. *SM-Res* is hand-designed and regular in using floating-point and SIMD instructions during the high-power phase of the loop. *A-Res* uses a combination of integer and floating-point operations and high- and low-power instructions, including some NOPs in the high-power phase.

By mixing integer and floating-point operations, it is able to exercise multiple schedulers and execution clusters in the pipeline. What is more difficult to assess is why sprinkling NOPs in the code increases the droop.

To further understand the effect of the NOPs, we replaced the NOPs in the high-power region with independent, integer ADD operations and measured the resulting droop. If the pipeline flow remains the same, the ADDs should produce a higher droop than the NOPs since they are a higher power operation than NOPs. The modified *A-Res* stressmark generated a smaller droop (by 40 mV) than the original stressmark. In addition, the frequency of the di/dt pattern shifted lower than the ideal resonant frequency, indicating that the duration of the loop increased due to the inclusion of the ADD operations. Unlike ADDs, NOPs consume fetch and decode resources but do not affect other structures in the pipeline such as the schedulers, physical registers, or result busses. The use of the NOPs enabled the stressmark to attain resonance. Although the pipeline and the modified *A-Res* stressmark are constructed to attain a throughput of four instructions per cycle, resource hazards such as physical register availability, decode width capabilities, token-based scheduling restrictions, and result bus utilization impact the final outcome. AUDIT with its GA-based algorithm was able to construct a stressmark that worked around the pipeline hazards to produce a large droop. AUDIT's ability to accommodate pipeline restrictions is examined further in Section 5.B.

One valid concern is that AUDIT stressmarks are unrealistic because the droops generated by them are much worse than normal benchmarks or other stressmarks. As noted earlier, instead of using the stressmarks to set voltage margins, they can be used to understand the bounds of the problem and sensitivities of the pipeline. For example, the *A-Res* stressmark shows that it is possible to generate large droops by selecting both the floating-point and integer execution clusters in the pipeline rather than just focusing on the floating-point pipeline. Additionally, as will be shown in Section 5.B, when one di/dt stress path is blocked through droop mitigation mechanisms, AUDIT can find other high-stress paths in the pipeline. As noted by Patel [20], there are many sensitive paths on cores that can lead to catastrophic failures when the system is stressed by reduced noise margins, and it is imperative that we have the tools necessary to identify these paths.

6) *Observations and summary*

There are some key observations from the results and discussion so far:

- Benchmarks do not produce the same levels of first-order droop as the stressmarks. On our processor, this is true even for benchmarks that have global synchronization resulting from barriers.
- There are many different ways to construct a stressmark in a multi-core system depending on what structures and types of configurations one is trying to exercise. Therefore, a stressmark that works well for one configuration (such as *A-Res* for 4T runs) may not produce the best results for other configurations. AUDIT's flexibility and ease of use can be leveraged to develop a suite of stressmarks that can effectively exercise all significant usage scenarios in the system.
- The measured droop is not the only indicator of sensitivity to failure. The paths exercised by the stressmark and the number of times the droop event occurs also have an impact on overall program susceptibility.

- AUDIT is able to match or exceed the droops produced by benchmarks and other stressmarks by exercising a richer set of paths in the pipeline.

The rest of the results section presents ways in which AUDIT can accommodate microarchitectural and architectural changes.

B. *Impact of FPU throttling*

Floating-point and SIMD instructions are generally the highest-power instructions available in the execution pipeline and they are used extensively in the high-power portion of the stressmark. A number of papers have noted that hardware and software architectural throttling schemes reduce di/dt stresses by limiting the rate of change in the execution of high-power instructions [3][5][7][9][10][18][21][22][23]. We utilize a FPU throttling scheme that statically limits the maximum number of FPU instructions executed in a cycle.

We measured the droop on some of the stressmarks with FPU throttling enabled to determine the maximum droop and maximum voltage at failure. The results are shown in Table 2. As before, all droop data are relative to the 4T *SM1* stressmark with FPU throttling disabled. FPU throttling is highly effective for *A-Res* and *SM-Res*, but less so for *SM1*. *SM1* is composed of multiple high-stress code sequences, and FPU throttling does not affect all stress paths in *SM1*. Although the results vary, the droop and voltage at failure improve with FPU throttling. These results show that FPU throttling functions as expected by limiting di/dt stresses; however, the results so far do not show whether AUDIT can find another stressmark that can produce a significant droop with FPU throttling enabled.

TABLE II. IMPACT OF FPU THROTTLING ON RELATIVE DROOP (RELATIVE TO 4T SM1) AND FAILURE POINT (RELATIVE TO 4T A-RES).

	Stressmark	Rel. Droop	Failure Point
No Throttling	SM1	1	V_F – 62 mV
	A-Res	1.39	V_F
	SM-Res	1.25	V_F – 12 mV
FPU Throttling	SM1	0.93	V_F – 75 mV
	A-Res	0.86	V_F – 100 mV
	SM-Res	0.78	V_F – 113 mV
	A-Res-Th	0.98	V_F – 75 mV

We used AUDIT to generate a new stressmark (*A-Res-Th*) to determine if there are other opportunities to generate a large droop in conjunction with FPU throttling. We repeated the AUDIT stressmark generation using four threads, but with FPU throttling enabled. Table 2 shows the droop and failure levels for the new stressmark *A-Res-Th*. AUDIT was able to generate a stressmark that works around the FPU throttling restrictions to increase the size of the droop. However, it is not able to match the droops seen without FPU throttling because it is now limited to using fewer high-power floating-point and SIMD operations. With FPU throttling enabled, *A-Res-Th* exceeds the *SM1* stressmark for droop and matches it for sensitivity to voltage. It also highlights another stress path through the processor for engineers to evaluate.

The results show the experimental FPU throttling scheme works well for reducing voltage droops in the system, and AUDIT, in a relatively short time (~5 hours) has identified another path that can still produce significant voltage droops with FPU throttling enabled.

C. *AUDIT on a different processor*

To present AUDIT's ability to adjust to microarchitecture and system changes, we replaced the Bulldozer-based processor in the experimental system with an older-generation 45-nm AMD Phenom II X4 Model 925. The rest of the board remained unchanged. Each core in the AMD Phenom processor has local L1 and L2 caches, no multi-threading, and less variation between high- and low-power regions because it does not manage power as aggressively as the Bulldozer-based system. We generated new resonant stressmarks for the AMD Phenom processor using AUDIT and the results are shown in Table 3. We were unable to run *SM1* on the older processor due to incompatible instructions. As with the Bulldozer-based system, AUDIT was able to generate stressmarks that were comparable or better than hand-tuned stressmarks, highlighting the capabilities of automatic stressmark generation tools such as AUDIT.

TABLE III. DROOP AND FAILURE RESULTS FOR A 45-NM AMD PHENOM II PROCESSOR. DROOP AND FAILURE POINT ARE SHOWN RELATIVE TO SM2.

	zeusmp	SM2	A-Res
Rel. Droop	0.82	1	1.10
Failure Point	V_F – 50 mV	V_F	V_F

VI. PREVIOUS WORK

There has been some previous work on hardware analysis of production systems. In [23], Reddi et al. measured and analyzed droops on a two-core Intel system and discussed constructive and destructive interference between processors and the difference in droops between average and worst-case scenarios. This information was used to design a noise-aware thread scheduler to mitigate some of the di/dt stresses in the system. To date, the work by Reddi is the most detailed hardware analysis of droops.

We expand on that work by analyzing a more complex system with multi-threading and up to eight logical processors. In addition, we show that constructive interference occurs more often than expected due to OS effects, and use this knowledge to design effective stressmarks.

More recently, Miller et al. examined voltage emergencies in multi-core processors [16] with increasing numbers of cores, and showed how global synchronization points create large stresses in the system. This work used power variability as a proxy for di/dt stresses and examined the hardware at a coarse granularity of 1-ms intervals. In our work, we use true voltage droop measurements and fine-grained sampling to detect first-order droops and discuss droop values as well as voltage failure points in hardware.

The second major contribution of our work is automatic stressmark generation using real hardware. Joseph, Brooks, and Martonosi presented a hand-coded di/dt stressmark [10]. Their basic idea was to create a sequence in which a high-current instruction follows a low-current instruction. The high-current component typically consisted of a memory load/store instruction and the low-current component consisted of a divide instruction followed by a dependent instruction, resulting in a long pipeline stall. However, their di/dt stressmark was manually crafted for a specific microarchitecture based on the knowledge of the current draw of various instructions. Furthermore, they focused only on memory-intensive behavior such as loads and stores and increased current draw by accessing L1 and L2 data caches. In contrast, our approach does not require microarchitectural knowledge and relies on measured voltage droops in a closed-loop measurement infrastructure.

Ketkar and Chiprout proposed a di/dt stressmark generation methodology using integer linear programming (ILP) [12]. They extracted current draw for certain instructions from a register transfer language (RTL) model for the hardware. Linear programming with constraints was used to maximize voltage droop. However, they focused only on the ALU. It is difficult to make ILP relationships of instructions for all the pipeline stages and the caches; hence, it is difficult to apply their technique to an entire processor, especially one with out-of-order processing, multiple cores, and complex shared resource structures.

Joshi et al. [11] presented a methodology for generating maximum-power viruses and mentioned in passing that high-power and low-power instruction sequences from two different power optimizations can be interleaved to generate a di/dt stressmark. This was only a suggestion, without implementation details or results. Also, they did not talk about the importance of repeating the sequence at the PDN's resonant frequency. Neither di/dt effects nor voltage droops were the focus of Joshi's work.

Kim and John [13] presented a methodology for automatic generation of a di/dt stressmark for a single-core processor. However, no methodology was presented for multi-core or multi-threaded processors. Furthermore, that was a simulation-based study using SimpleScalar with the Alpha instruction set; they did not use hardware to generate or validate the stressmarks. As discussed in Section 3, we have expanded on their work by introducing dithering and sub-blocking to produce better stressmarks, analyzing the results on hardware, comparing AUDIT's performance to existing stressmarks, and showing AUDIT's flexibility in handling microarchitectural and architectural constraints.

A significant number of other studies have focused on preventing, reducing, or recovering from di/dt effects or voltage droops [3][5][6][7][8][9][10][15][17][18][21][22][23]. However, none of these focus on automatically generating stressmarks for di/dt.

VII. Conclusion

This paper addressed the issue of di/dt stress generation and analysis in state-of-the-art multi-core x86 processors. We discussed the complexities involved with designing a stressmark, especially for multi-core systems. We presented the dithering algorithm to aid in multi-core stressmark generation and evaluated an automatic stressmark generation tool, AUDIT, that uses sub-blocking and dithering to produce stressmarks using hardware. We showed how AUDIT stressmarks compare to existing stressmarks and standard benchmarks, and presented results showing AUDIT's ability to adjust to different processor characteristics such as shared resources, FPU throttling, and different processors.

Acknowledgment

Lizy John and Youngtaek Kim are partially supported by NSF grant 1117895 and by AMD unrestricted research funding. Views and opinions expressed in this paper are that of the authors and not that of the National Science Foundation. This work was conducted when Youngtaek Kim was an intern at AMD Research. The authors thank Tom Snodgrass for helping set-up and run all the laboratory equipment, and thank other AMDers for giving great help and invaluable comments during this research.

References

[1] C. Bienia, Benchmarking Modern Multiprocessors, Ph.D. Thesis. Princeton University, 2011.

[2] M. Butler, L. Barnes, D. Sarma, and B. Gelinas, "Bulldozer: an approach to multithreaded compute performance," *IEEE Micro*, vol. 31, no. 2, pp. 6-15, 2011.

[3] W. El-Essawy and D. Albonesi, "Mitigating inductive noise in SMT processors," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[4] T. Fischer et al., "Design solutions for the Bulldozer 32nm SOI 2-core processor module in an 8-core CPU," in *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2011.

[5] E. Grochowski, D. Ayers, and V. Tiwari, "Microarchitectural simulation and control of di/dt-induced power supply voltage variation," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-8)*, 2002.

[6] M. Gupta, J. Oatley, R. Joseph, G. Wei, and D. Brooks, "Understanding voltage variations in chip multiprocessors using a distributed power-delivery network," in *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2007.

[7] M. Gupta, K. Rangan, M. Smith, G. Wei, and D. Brooks, "DeCoR: a delayed commit and rollback mechanism for handling inductive noise in processors," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-14)*, 2008.

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

[9] K. Hazelwood and D. Brooks, "Eliminating voltage emergencies via microarchitectural voltage control feedback and dynamic optimization," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[10] R. Joseph, D. Brooks, and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-9)*, 2003.

[11] A. Joshi, L. Eeckhout, L. John, and C. Isen, "Automated microprocessor stressmark generation," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-14)*, 2008.

[12] M. Ketkar and E. Chiprout, "A microarchitecture-based framework for pre- and post-silicon power delivery analysis," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, 2009.

[13] Y. Kim and L. John, "Automated di/dt stressmark generation for microprocessor power delivery networks," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 2011.

[14] N. Kurd, J. Douglas, P. Mosalikanti, and R. Kumar, "Next generation Intel® Core micro-architecture (Nehalem) clocking architecture," *IEEE Journal of Solid-state Circuits*, vol. 44, iss. 4, pp. 1121-1129, 2009.

[15] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tiemo, and J. Carter, "Active management of timing guardband to save energy in POWER7," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*, 2011.

[16] T. Miller, R. Thomas, X. Pan, and R. Teodorescu, "VRSync: characterizing and eliminating synchronization-induced voltage emergencies in many-core processors," in *Proceedings of the 39th International Symposium on Computer Architecture (ISCA-39)*, 2012.

[17] F. Mohamood, M. Healy, S. Lim and H. Lee, "A floorplan-aware dynamic inductive noise controller for reliable 2D and 3D microprocessors," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)*, 2006.

[18] M. Pant, P. Pant, D. Willis, and V. Tiwari, "Architectural solution for the inductive noise problem due to clock-gating," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 1999.

[19] S. Pant and E. Chiprout, "Power grid physics and implications for CAD," in *Proceedings of the 43rd Annual Design Automation Conference (DAC-43)*, 2006.

[20] J. Patel, "CMOS process variations: a critical operation point hypothesis," *Online Presentation*, 2008.

[21] M. Powell and T. Vijaykumar, "Exploiting resonant behavior to reduce inductive noise," in *Proceedings of the 31st International Symposium on Computer Architecture (ISCA-31)*, 2004.

[22] V. Reddi, M. Gupta, G. Halloway, G. Wei, M. Smith, and D. Brooks, "Voltage emergency prediction: using signatures to reduce operating margins," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-15)*, 2009.

[23] V. Reddi, S. Kanev, W. Kim, S. Campanoni, M. Smith, G. Wei, and D. Brooks. "Voltage noise in production processors," *IEEE Micro,* vol. 31, no. 1, pp. 20-28, 2011.

[24] The NASM Development Team, "NASM - The Netwide Assembler," Available: http://www.nasm.us.

[25] D. Weiss et al., "An 8MB Level-3 cache in 32nm SOI with column-select aliasing," in *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2011.