Voltage Emergency Prediction: Using Signatures to Reduce Operating Margins

Vijay Janapa Reddi, Meeta S. Gupta, Glenn Holloway, Gu-Yeon Wei, Michael D. Smith, David Brooks
Harvard University
{vj, meeta, holloway, guyeon, smith, dbrooks}@eecs.harvard.edu

Abstract

Inductive noise forces microprocessor designers to sacrifice performance in order to ensure correct and reliable operation of their designs. The possibility of wide fluctuations in supply voltage means that timing margins throughout the processor must be set pessimistically to protect against worst-case droops and surges. While sensor-based reactive schemes have been proposed to deal with voltage noise, inherent sensor delays limit their effectiveness. Instead, this paper describes a voltage emergency predictor that learns the signatures *of voltage emergencies (the combinations of control flow and microarchitectural events leading up to them) and uses these signatures to prevent recurrence of the corresponding emergencies. In simulations of a representative superscalar microprocessor in which fluctuations beyond 4% of nominal voltage are treated as emergencies (an aggressive configuration), these signatures can pinpoint the likelihood of an emergency some 16 cycles ahead of time with 90% accuracy. This lead time allows machines to operate with much tighter voltage margins (4% instead of 13%) and up to 13.5% higher performance, which closely approaches the 14.2% performance improvement possible with an ideal oracle-based predictor.*

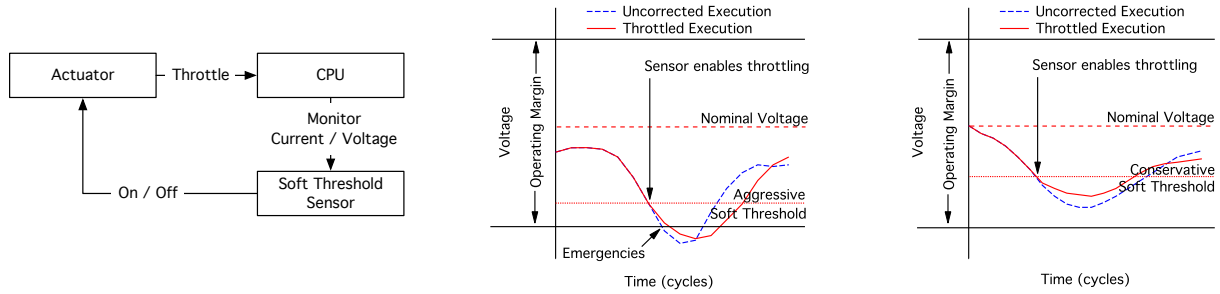
1. Introduction

The power ceiling in modern microprocessors presents a major challenge to continued performance scaling. Power-reduction techniques such as clock gating, when aggressively applied to constrain power consumption, can lead to large current swings in the microprocessor. When coupled with the non-zero impedance characteristics of power-delivery subsystem, these current swings can cause the voltage to fluctuate beyond safe operating margins. Such events, called *voltage emergencies*, have traditionally been dealt with by allocating sufficiently large timing margins. Unfortunately, on-chip voltage fluctuations and the margins they require are getting worse. A recent paper analyzing emergency-prone activity of the POWER6 microprocessor [12] shows that required timing margins translate to operating voltage margins of nearly 20% of the nominal supply voltage ($\sim$200mV for a nominal voltage of 1.1V). Such conservative operating voltage margins ensure robust operation of the system, but can severely degrade performance due to the lower operating frequencies. To reduce the gap between nominal and worst-case operating voltages, this paper proposes a *voltage emergency predictor* that identifies when emergencies are imminent and prevents their occurrence.

A voltage emergency predictor anticipates voltage emergencies using *voltage emergency signatures* and throttles machine execution to prevent them. An emergency signature is an interleaved sequence of control-flow events and microarchitectural events leading up to an emergency. A voltage emergency signature is captured when an emergency first occurs by taking a snapshot of relevant event history and storing it in the predictor. A built-in checkpoint-recovery mechanism then rolls the machine back to a known correct state and resumes execution. Subsequent occurrences of the same emergency signature cause the predictor to throttle execution and prevent the impending emergency. By doing so, the predictor enables aggressive timing margins in order to maximize performance.

The signature-based predictor outperforms previously proposed architecture-centric techniques [6, 13, 22, 23] that rely on voltage sensors to detect and react to emergencies via throttling. In these prior schemes, emergencies are detected by using a voltage sensor to monitor the supply voltage for specific soft threshold crossings, which indicate voltage margin violations are possible. Whenever the supply voltage falls below this threshold, the machine throttles execution in pursuit of emergency prevention. Unfortunately, these schemes cannot always guarantee correctness without incurring large performance penalties. Aggressively setting the soft threshold close to the operating margin limits time available to throttle and successfully prevent an emergency. Alternatively, setting the threshold too conservatively leads to unnecessary throttling that degrades performance. Not every conservative soft threshold crossing eventually crosses the lower operating voltage margin. Our predictor instead recognizes and tracks patterns of emergency-prone activity to proactively throttle execution well before an emergency can occur. Our results show high prediction accuracy is possible, which translates to performance enhancements by reducing otherwise conservative margins.

An additional benefit is that our voltage emergency predictor does not require fine tuning based on specifics of the microarchitecture nor the power delivery subsystem, as is the case with reactive sensor-based schemes. The current



(a) (b) (c)

Figure 1: Sensor-based throttling. (a) A feedback loop is intended to detect and prevent emergencies. (b) Aggressive soft thresholds allow too little time to prevent emergencies. (c) Conservative soft thresholds trigger unnecessary throttling.

and voltage activity of a microprocessor are products of machine utilization that are specific to the workload's dynamic demands. Capturing that activity in the form of voltage emergency signatures allows the predictor to dynamically adapt to the emergency-prone behavior patterns resulting from the processor's interactions with the power delivery subsystem without having to be preconfigured to reflect the characteristics of either.

Some researchers have also proposed to use checkpoint-recovery alone to handle voltage emergencies. However, the coarse-grained checkpointing intervals of traditional checkpoint-recovery schemes (between 100 and 1000 cycles) translate to unacceptable performance penalties. Gupta *et al.* [9] have proposed a low-overhead implicit checkpointing scheme to handle voltage emergencies by buffering commits until it is confirmed that no voltage emergencies have occurred while the buffered sequence was in flight. While shown to be effective, implicit checkpointing is specialized and requires modifications to traditional microarchitectural structures. Since coarse-grained checkpoint-recovery is already available in existing production systems [1, 26] to serve multiple purposes [15, 16, 19, 25, 27, 29], we also rely on it as a fail-safe mechanism during predictor training.

In summary, the contributions of this paper are:

- *Voltage emergency prediction.* Recognizing that activity leading to voltage emergencies is a consequence of program control flow and microarchitectural events, we show that voltage emergencies are predictable with over 90% accuracy by exploiting program behavior and locality.
- *Signature-based voltage emergency reduction.* A voltage emergency predictor relies on traditional checkpoint-recovery to capture voltage emergency signatures and prevents emergencies via throttling. Its performance comes to within 5% of an oracle-based throttling scheme.
- *Efficient predictor implementation.* A Bloom filter-based voltage emergency predictor implementation is shown to achieve 11.1% improvement in performance, approaching the 14.2% possible with an oracle-based throttling scheme.

The remainder of the paper is organized as follows: Section 2 reviews the limitations of sensor-based schemes for handling voltage emergencies. Section 3 then shows how program control flow and microarchitectural events can be used to predict voltage emergencies. Section 4 describes the experimental framework we use to evaluate predictor accuracy in Section 5. Finally, Section 6 compares the performance of the predictor to other schemes and describes implementation tradeoffs.

2. Limits of Sensor-Based Approaches

Given the direct impact of voltage on circuit delay, intermittent voltage droops, past a lower operating margin, can slow down logic delay paths and lead to timing violations. Voltage spikes that exceed an upper margin can cause long-term reliability issues. Hence, modern designs impose conservative operating voltage margins to avoid these voltage emergencies and guarantee correct operation in the microprocessor. However, large margins translate to inefficient energy consumption and lower performance. This section reviews sensor-based techniques that react to and mitigate on-chip voltage emergencies.

A typical sensor-based proposal uses a tight feedback loop like that shown in Figure 1(a). The loop includes a sensor that tries to detect impending emergencies and a throttling actuator that tries to avoid them. The sensor relies on a soft current or voltage threshold as a "canary". Crossing that threshold means that voltage is approaching its lower margin, so the actuator turns on throttling until the crisis is past. Proposed throttling schemes range from frequency throttling, to pipeline freezing/firing, to issue ramping, and altering the number of accessible memory ports [6, 13, 22, 23]. The behavior of the feedback loop is determined by two parameters, the setting of the soft threshold level and the delays around the feedback loop. Unfortunately, choosing those parameters to accommodate reduced operating margins is thwarted by correctness failures and/or performance penalties.

Correctness failures. Figure 1(b) illustrates the use of a soft threshold to throttle execution and prevent an emergency. The graph shows voltage waveforms with and without sensor-based throttling (Throttled Execution and Uncorrected Execution, respectively). The solid horizontal line marked Aggressive Soft Threshold indicates the thresh-

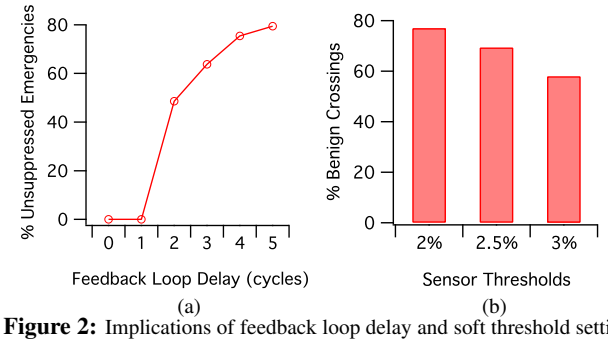


Figure 2: Implications of feedback loop delay and soft threshold settings on correctness and performance. (a) A large percentage of emergencies are not detected with sufficient time to prevent them due to feedback loop delays. (b) Even assuming a 0-cycle feedback loop delay, the number of soft threshold crossings that do not violate the minimum operating margin (i.e., benign crossings) is so large that performance suffers due to unnecessary throttling.

old at which a voltage sensor starts to take action to prevent an emergency. Setting the soft threshold aggressively (i.e., close to the lower operating margin) requires a very fast reaction by the sensor and actuation system. Failure to respond quickly enough results in a voltage emergency. In Figure 1(b), the voltage starts to recover under throttling, but not in time to avoid crossing the lower operating margin.

Figure 2(a) shows the sensitivity of sensor-based mechanisms to feedback loop delays by plotting the number of emergencies that go unsuppressed in our benchmark suite as a function of sensor-loop delay times. Here we assume the soft threshold to be 3% below the nominal voltage and the lower operating margin to be 4% below nominal. Feedback loop delays ranging between 0 and 5 cycles would require a nearly perfect sensor. Yet even a 2-cycle delay causes 50% of all soft threshold crossings to violate the simulated microprocessor's minimum operating margin specification. In other words, fail-safe execution is not possible at this margin using sensor-based schemes, as they cannot operate in a timely manner.

Performance penalties. To accommodate slow sensor response times and ensure that throttling effectively prevents emergencies, sensor-based schemes can use conservative soft thresholds. Lifting the soft threshold away from the lower operating margin, as illustrated by the Conservative Soft Threshold in Figure 1(c), gives the throttling system more time to prevent an emergency. But as the Uncorrected Execution waveform in Figure 1(c) shows, even in the absence of throttling, a soft threshold crossing may not be followed by an emergency. Throttling execution in such cases decreases performance without any compensating benefit. The more conservative the soft threshold setting, the greater the performance penalty. Figure 2(b) shows that this penalty can be quite large. Assuming an ideal sensor with no feedback loop delay (i.e., 0-cycle sensor delay), the percentage of benign soft threshold crossings is between 77% and 58% for soft thresholds ranging from 2% to 3%. So even if it were possible to design a feedback loop with no delay, the large performance penalties would deter architects from reducing operating margins.

Resonant versus isolated pulse emergencies. A sensor-based scheme proposed by Powell and Vijaykumar [22] reduces sensitivity to feedback loop delay by focusing on voltage emergencies that are the result of resonating patterns. While resonance-induced emergencies are dominant for some packages, recent work by Gupta *et al.* [9] illustrates that non-resonant (pulse) events are also a major source of emergencies across a range of packages. James *et al.* [12] have observed isolated (non-resonant) pulses in a POWER6 chip implementation. And Kim *et al.* show that resonant emergencies are likely to become less important than isolated pulses in future chip multi-processors with on-chip voltage regulators, as package inductance effects are decoupled from the power grid via on-chip regulators [14]. Therefore, to realize the benefits in improved energy efficiency or performance that reduced margins can enable, new solutions are needed that cope with both resonant and non-resonant voltage emergencies in future systems.

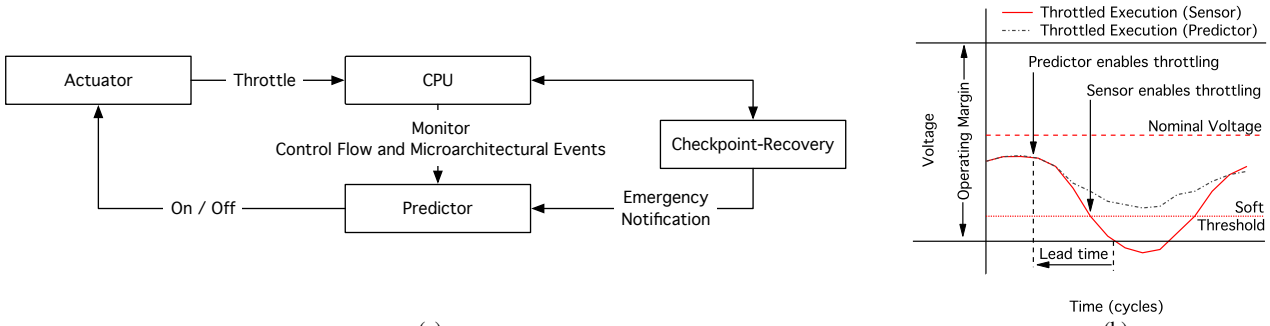
3. Prediction-Based Throttling

An effective emergency avoidance mechanism must meet two criteria: First, it must anticipate an emergency accurately to prevent performance degradation due to unnecessary throttling. Second, it must initiate the emergency avoidance mechanism with enough lead time to throttle and successfully prevent the emergency from occurring. A major goal of this work is to show that it is possible to *predict* voltage emergencies with high accuracy and sufficient lead time to throttle and prevent emergencies.

3.1. Overview

A voltage emergency predictor is a structure that learns recurring voltage emergency activity during runtime and prevents subsequent occurrences of said emergencies via execution throttling. Figure 3(a) presents a block diagram of the proposed scheme. The predictor monitors control flow and microachitectural events and keeps track of the *voltage emergency signatures* that cause voltage emergencies, identified by the checkpoint-recovery block. The predictor also actuates throttling to avoid future emergencies, but does not suffer limitations associated with sensor delays or soft thresholds. Unlike sensor-based schemes, our prediction-based approach allows the microprocessor to operate with margins much tighter than otherwise possible.

A voltage emergency signature comprises an interleaved sequence of program control flow and microarchitectural events that give rise to an emergency. Voltage emergency signatures are dynamic and, as such, must be discovered at runtime. Initially, no emergency signatures are known. As the program executes, emergencies are detected as margin violations occur. Since an emergency can potentially corrupt machine state, a checkpoint-recovery mechanism is in place to recover and resume execution. While invoking the recovery mechanism, the predictor captures the signature of the emergency. Over time, the predictor collects a history of emergency-prone activity and uses this history to



(a) (b)

Figure 3: Overview of voltage emergency prediction. (a) The predictor replaces the soft threshold sensor. It relies on code and microarchitectural event activity instead of current and voltage activity to decide when to throttle. It is trained using a fail-safe checkpoint-recovery mechanism. (b) Voltage waveforms illustrate how the predictor throttles execution with sufficient lead time to prevent emergencies instead of relying on soft thresholds.

successfully prevent future emergencies via throttling. Section 3.2 presents detailed insights into emergency signatures and our reasoning for assuming a fail-safe checkpoint-recovery mechanism.

A voltage emergency predictor does not require a soft threshold. Instead, it monitors sequences of program paths and architectural events, and initiates throttling whenever an emergency-causing pattern is detected. For clarity and a brief overview, Figure 3(b) illustrates how a predictor-based scheme outperforms a sensor-based throttling scheme. As soon as the predictor observes a voltage emergency signature, it starts to throttle execution with sufficient lead time to prevent an emergency from occurring. In contrast, sensor-based throttling, corresponding to waveform Throttled Execution (Sensor) from Figure 1(b), fails to avoid the emergency with aggressive soft threshold settings. Conservative soft thresholds incur large performance penalties.

3.2. Voltage Emergency Prediction

In the following subsection, we explore the working principles underlying voltage emergency prediction using a specific, but real-life, scenario from benchmark *403.gcc*. Building upon the insights we gain from this example, we demonstrate how to capture a voltage emergency signature, which is the enabling mechanism behind a voltage emergency predictor. We then discuss factors that influence the quality of an emergency signature, such as the type and amount of information recorded.

3.2.1 Exploiting Voltage Emergency Activity Patterns

Programs are highly repetitive. Repeating code patterns give rise to repeating patterns of memory access and data flow through the processor. Gupta *et al.* show repeating sequences of processor activity have the potential to cause voltage emergencies [8]. They elaborate that microarchitectural events such as cache misses and pipeline flushes stall the pipeline. As a consequence, machine activity temporarily reduces. Upon recovering/restarting, there is a rush of activity that causes the current to spike and the voltage to drop sharply; a voltage emergency occurs when the voltage exceeds the lower operating margin. However, it is not well understood *when* such microarchitectural events are benign versus harmful. In other words, there is no guarantee that a branch misprediction or any recurring event will always cause an emergency. In this section, we show it is possible to predict the likelihood of an emergency more accurately by taking into account the context leading up to the emergency.

A microarchitectural event acting in complete isolation only sometimes causes an emergency by itself. To help illustrate when an event causes an emergency, Figure 4(a) shows pipeline activity over 880 cycles for benchmark *403.gcc* while it is executing the nested loop illustrated in Figure 4(b). Figure 4(a) illustrates pipeline flushing due to branch mispredictions using a vertical bar in the Flush subgraph. The number next to each vertical bar in the Flush graph corresponds to the basic block number in Figure 4(b) containing the mispredicted branch. Other relevant pipeline activities across different parts of our simulated microprocessor ranging from cache access, to functional unit usage, to the rate at which instructions are being dispatched, issued and committed are also shown for the same time frame. The resulting current draw and voltage activity are also shown. Lastly, Figure 4(a) shows three distinct phases A, B and C (see top of figure) and each phase terminates at an emergency (see bottom of figure).

Context. Microarchitectural events perturb machine activity significantly, but by themselves are not responsible for voltage emergencies. Pipeline flush Event 2 in Figure 4(a) is an ideal candidate for illustrating this point. Event 2 in Phase A causes a voltage droop a few cycles before Event 5 (also in Phase A), but it does not cause an emergency. The same event, however, always causes an emergency in Phase B (at the end of B). Understanding the processor activity leading up to these events explains this inconsistent behavior. The Issue, as well as other rates prior to Event 2 are different between Phase A and Phase B, so the perturbation effects of Event 2 are different between the phases. By comparison, pipeline flush Event 5 always occurs just prior to an emergency in both Phase A and Phase C. Nevertheless, our argument that activity prior to an event matters holds true. The voltage just prior to Event 5 in Phase A is rising versus falling in Phase C. The latter occurs because the voltage is already in flux due to the perturbation brought about by Event 2 in Phase B. For this reason, any scheme attempting to characterize and exploit recurring patterns must take into

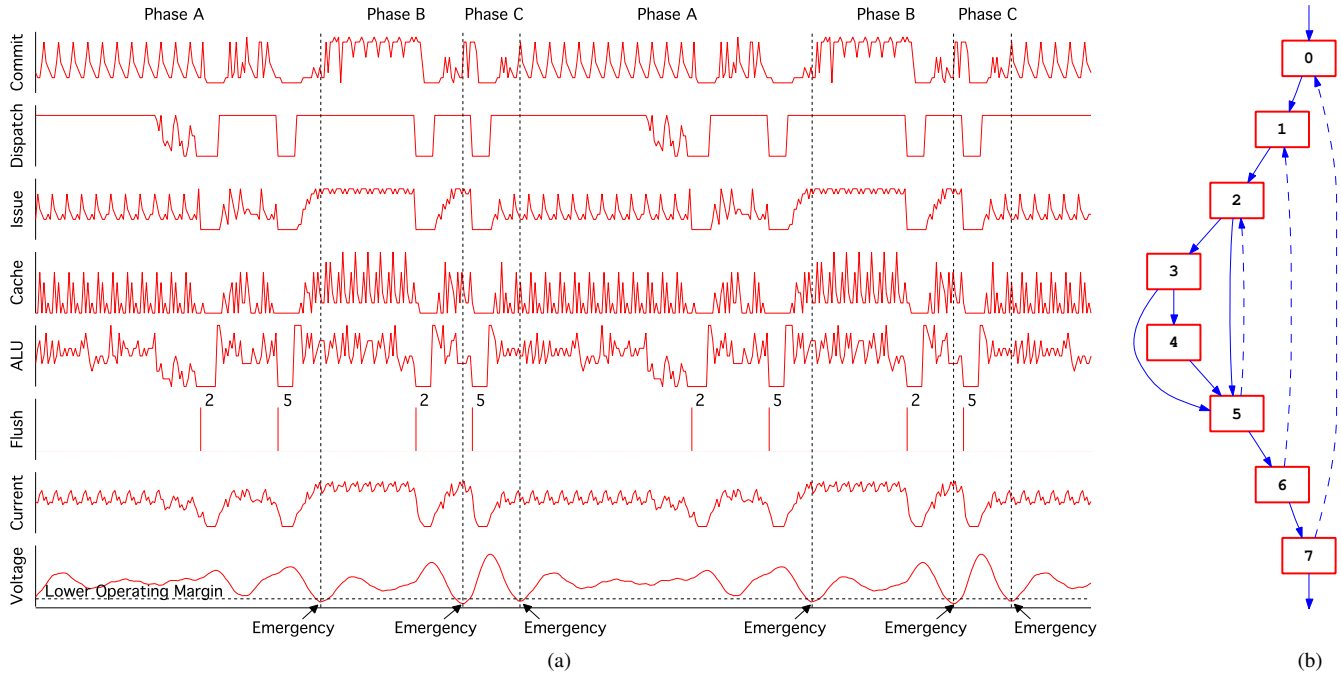


Figure 4: (a) Voltage emergencies are associated with recurring activity (phases A, B and C) over 880 cycles. The numbers next to the vertical bars in the Flush graph correspond to the basic block number in (b) containing the mispredicted branch. (b) An emergency prone nested-loop in function `init_regs` of benchmark *403.gcc*. `init_regs`'s activity snapshot is shown in (a).

account the execution context preceding an emergency.

Microarchitectural events and program control flow interleaving. Voltage emergencies are uniquely identifiable by tracking control flow instructions and microarchitectural events in order of occurrence. Rapid fluctuations in a program's control and data flow and in its level of parallel utilization of processor resources lead to changes in current flow that induce large voltage swings. For instance, the distinct current and voltage activity between phases A, B and C are the result of different control flow paths exercised by the program combined with the voltage droops induced by pipeline flush Events 2 and 5. During the early part of Phase A, the program is executing basic blocks $2 \rightarrow 3 \rightarrow 5$ (from Figure 4(b)) in a steady-state manner. The stable and repetitive Issue rate pattern during the early part of Phase A in Figure 4(a) confirms this. Slightly past the midpoint of Phase A, the program switches control flow from basic blocks $2 \rightarrow 3 \rightarrow 5$ to basic blocks $2 \rightarrow 5$. This switch triggers a pipeline flush to recover from speculatively executing incorrect code along Edge $2 \rightarrow 3$ to executing correct code along Edge $2 \rightarrow 5$. The activity on the recovery path following the pipeline flush causes the voltage to droop slightly but not enough to violate the operating margin (shown using Lower Operating Margin). After a few cycles, a misprediction on basic block 5's control instruction eventually leads to a voltage emergency. So the emergency in Phase A is because of the activity including, as well as following, basic blocks $2 \rightarrow 3 \rightarrow 5$ combined with pipeline flush Events 2 and 5. In contrast, the emergency in Phase B arises from executing basic blocks $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ followed by the single flush Event 2. Consequently, tracking control flow sequence along with pipeline flush events in order of occurrence yields two unique activity patterns representing Phase A and Phase B.

Recurrence and stability. Voltage emergencies, like program phases, are repetitive over a program's lifetime, which make them predictable. Consider the three phases illustrated in Figure 4(a). The phases are recurring because execution sequence flows through phases $A \rightarrow B \rightarrow C$ and back to Phase A. A subsequent occurrence of the same phase leads to yet another emergency. For instance, Event 2 always causes an emergency as execution flows through phases $B \rightarrow C$, but not through phases $A \rightarrow B$. Thus, a pattern of voltage emergency occurrence emerges. Identifying and exploiting such recurring activity is the basis for predicting voltage emergencies in terms of program behavior, as well as microarchitectural behavior.

3.2.2 Capturing Voltage Emergency Signatures

In this section, we describe the hardware necessary to capture program control flow and microarchitectural event interleaving.

Emergency detection. Capturing a voltage emergency signature, with our scheme, requires an emergency to occur at least once. So we require a mechanism to monitor operating margin violations. We rely on a voltage sensor. Our scheme is not time-sensitive to sensor delay because the predictor does not react to sensing a soft threshold crossing to throttle. The sensor is used to signal that an emergency has occurred and the system ought to take appropriate actions.

Checkpoint-recovery. Processor state is potentially corrupted as emergencies occur, since voltage emergencies induce timing faults. So we rely on a fail-safe checkpoint-recovery mechanism to recover from emergencies. The fail-safe mechanism initiates a recovery whenever the sensor detects an emergency, and in that process also captures a voltage emergency signature. Checkpoints can be taken at varying intervals (e.g., 10-1000 cycles). We assume a 100-cycle rollback penalty.

Coarse-grained checkpoint-recovery is already shipping in today's production systems [1, 26], and researchers are proposing a broad range of novel applications that use traditional checkpoint-recovery [15,16,19,25,27,29]. With ever-increasing applications of this fail-safe mechanism, we believe checkpoint-recovery will become part of future mainstream processors. However, checkpoint-recovery alone as a solution for handling voltage emergencies is unacceptable due to performance penalties (as shown in Section 6.1).

Event history register. The predictor relies on a shift register to capture the interleaved sequence of control flow instructions and architectural events that give rise to an emergency. A signature is a snapshot of the event history register. The interleaving of events in the event history register is important for capturing the dynamic current and voltage activity resulting from program interactions with the underlying microarchitecture (as described in Section 3.2.1). The purpose of tracking the instruction stream is to capture the dynamic path of a program. Consequently, control flow instructions are ideal candidates for tracking a program's dynamic execution path.

Event history tracking is a well-studied topic in the area of branch prediction. Our contribution is unique in that we can identify the information flow that precisely captures activity prone to voltage emergencies.

Figure 5 illustrates example snapshots of the emergencies shown in Figure 4(a) across phases A, B and C. The updates into a 4-entry wide event history register are shown over time. At the point of the emergency in Phase B, the history register contains the following (from oldest to most recent): two control flow instruction addresses (illustrated as BR) and an event encoding for the pipeline flush (illustrated as 2), followed by another branch. It is important to never clear the event history register after capturing a snapshot to maintain a rolling window of contextual information. For example, the oldest BR in Signature C overlaps with the most recent entry in Signature B.

Since voltage emergencies contribute to timing faults, all predictor logic and checkpoint-recovery hardware must be carefully designed with sufficiently conservative timing margins. As these structures are not timing critical, there are no performance implications. Any state corruption in the predictor logic only leads to incorrect predictions, and will therefore only affect the performance of the system due to unnecessary throttling, but it will not violate correctness guarantees.

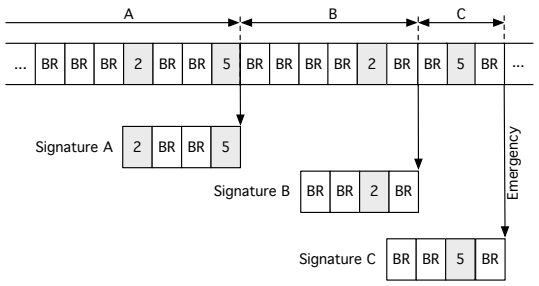


Figure 5: Overview of voltage emergency signatures. Taking snapshots of a 4-entry event history register for emergencies illustrated in Figure 4(a) across phases A, B and C.

3.2.3 Voltage Emergency Signature Semantics

The function of a voltage emergency signature is to precisely indicate whether a pattern of control flow and microarchitectural event activity will give rise to an emergency. To evaluate the effectiveness of different flavors of signatures, we define predictor accuracy as the fraction of predicted emergencies that become actual emergencies.

Contents. Information tracking in the event history register must correspond to parts of the execution engine that experience large current draws, as well as dramatic spikes in current activity. The event history register can collect the control flow trace at different points in a superscalar processor: in-order fetch and decode, out-of-order issue, and in-order commit. Each of these points contribute different amounts of information pertaining to an emergency. For instance, tracking execution in program order fails to capture any information regarding the impact of speculation on voltage emergencies. Tracking information at the in-order fetch and decode sequence captures the speculative path, but it does not capture the out-of-order superscalar issuing of instructions.

The accuracies of different signature types are illustrated in Figure 6(a) (assuming a signature size of 32 entries, which will be discussed next). Tracking committed control flow sequences in the event history register gives an accuracy of only 40%. If the history register tracks information at the decode stage, an accuracy of 72% is possible because the decode stage captures the speculative control flow path. Accuracy improves further by 12%, from 72% to 84%, if the history register tracks control flow at the issue stage, since we can now capture interactions more precisely at the level of hardware instruction scheduling and code executed along a speculative path.

Interleaving microarchitectural events with program control improves accuracy even further, as processor events provide additional information about swings in the supply voltage. For instance, pipeline flushes cause a sharp change in current draw as the machine comes to a near halt before recovering on the correct execution path (as observed in Figure 4(a) immediately following pipeline flush events). The last two bars of Figure 6(a) show accuracy improvements from adding microarchitectural event activity to the event history register. The second to last bar represents the ef-

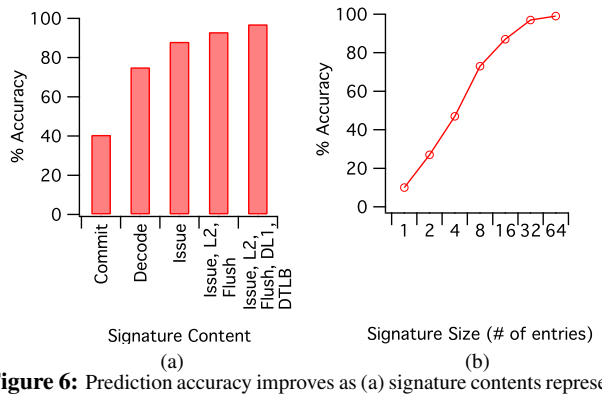


Figure 6: Prediction accuracy improves as (a) signature contents represent machine activity more closely and as (b) the number of entries per signature increases.

fect of capturing events that have the potential to induce large voltage swings—pipeline flushes and secondary (L2) cache misses. An improvement of five percentage points is achieved by taking flushes and L2 misses into account (i.e., total accuracy of 89%). Capturing the more frequently occurring events like DTLB and DL1 misses contributes additional improvements of ∼4%. Microarchitecture perturbations resulting from instruction cache activity (i.e., IL1 and ITLB) are negligible and do not lead to an improvement in accuracy.

From here on, we assume the event history register resides at the issue stage of the pipeline and captures microarchitectural-event activity. More formally, the event history register is updated whenever a control flow instruction is executed, along with Level 1 and Level 2 cache and TLB misses. Lastly, pipeline flushes are also events recorded in the event history register.

Size. Accuracy depends not only on recording the right interleaving of events, but also on balancing the amount of information the event history register keeps. Accuracy improves as the length of history register increases. However, it can be detrimental to increase the number of register entries beyond a certain count. Large numbers of entries in a signature can cause unnecessary differentiation between similar signatures—signatures whose most recent entries are identical and whose older entries are different, but not significantly so. The predictor would have to track more unique signatures per emergency because of this differentiation.

Figure 6(b) shows prediction accuracy improves as signature size increases. Accuracy is only 13% on average for a signature containing only 1 entry, which supports the discussion presented in Section 3.2.1 that voltage emergencies do not solely depend upon the last executed branch or a single microarchitectural event. It is the history of activity that determines the likelihood of a recurring emergency. Prediction accuracy begins to saturate once signature size reaches 16, and peaks at 99% for a signature size of 64 entries.

Signature encoding. Hardware implementations are resource constrained. So the number of bits representing a signature in a realistic hardware implementation matters. To avoid large overheads, we use a 3-bit encoding per entry in the event history register. But encoding causes aliasing between signatures. Therefore, we extend an encoded signature to also contain the program counter for the most recently taken branch—the *anchor PC*. Anchor PC's have the added benefit of implicitly providing the complete path information leading up to the most recent event in the history register. The 3-bit encoding compactly captures all of the relevant information consisting of different processor events, and takes into account the edge taken by each branch (i.e., fall-through paths are encoded as 000 versus 001 for taken edges). The compact representation described above results in a total signature length of 16 bytes (4 bytes for the anchor PC and 12 bytes for a signature size of 32 entries with 3 bits per entry).

Signature compaction. We can further reduce hardware overheads by folding multiple signatures corresponding to a specific anchor PC into a single representative signature. We use a weighted similarity metric based on Manhattan distance to determine how much compaction is possible for a set of signatures corresponding to a particular benchmark. Let x and y be k-element signatures associated with the same instruction address. We define the similarity of x and y to be

$$s = \frac{2}{k(k+1)} \sum_{i=1}^{k} \begin{cases} i & \text{if } x_i = y_i \\ 0 & \text{otherwise} \end{cases}$$

If the signatures are identical, s is one. If no two corresponding elements are the same, it is zero. The later elements in x and y correspond to later events in time. They are more heavily weighted in s, because they are more significant for emergency prediction. Other measures of similarity might yield better compaction, but they would be more expensive to compute in hardware. For a given instruction address, we partition the signatures into maximal sets in which each signature x is related to one or more other signatures y with similarity of 0.9 or greater. The resulting partition is then used instead of the original signature set.

The number of recurring signatures per benchmark varies significantly. Benchmark *403.gcc* has nearly 87000 signatures that repeatedly give rise to emergencies. At the other end of the spectrum is benchmark *462.libquantum* with only 39 signatures. Applying signature compaction on *403.gcc* reduces the number of signatures to 29000, thereby achieving a ∼67% reduction. Overall, compaction reduces the number of signatures by over 61% and the biggest winners are benchmarks that exhibit a large number of signatures.

4. Experimental Framework

The vehicle for all concepts and data presented in this paper is the $x86$ SimpleScalar infrastructure. Table 1 lists the configuration parameters used to initialize SimpleScalar for our baseline microprocessor design, which we refer to as Arch 1. The workload set is comprised of benchmarks from the SPEC CPU2006 suite. All but a few were simulated for 100 million instructions across their different inputs using

Clock Rate	3.0 GHz	RAS	64 Entries
Inst. Window	128-ROB, 64-LSQ	Branch Penalty	10 cycles
Functional Units	8 Int ALU, 4 FP ALU, 2 Int Mul/Div, 2 FP Mul/Div	Branch Predictor BTB	64-KB bimodal gshare/chooser 1K Entries
Fetch Width	8 Instructions	Decode Width	8 Instructions
L1 D-Cache	64 KB 2-way	L1 I-Cache	64 KB 2-way
L2 I/D-Cache	2MB 4-way, 16 cycle latency	Main Memory	300 cycle latency

Table 1: Baseline architecture (Arch 1) parameters for SimpleScalar.

Package	Peak Impedance (mOhm)	Current (A)	Quality Factor	Resonance Cycles	Comment
Pkg 1	5	16–50	3	30	Pentium 4 [2]
Pkg 2	2	30–70	2	60	Used in [13]
Pkg 3	17	16–50	6	30	Worst package

Table 2: Characteristics of the packages evaluated.

the phase most heavily weighted by Simpoint [21].[1] The benchmarks were compiled at optimization level *-O3* using the *GNU GCC 3.4* compiler toolchain.

To get a detailed cycle-accurate current profile, we incorporate a modified version of Wattch [4] into our SimpleScalar simulator. Simulated current profiles are convolved with an impulse response of the power delivery subsystem to obtain voltage variations. Other studies [13, 23] use this second-order model as well.

Operating margin. For the purpose of quantitative comparisons and evaluation, a maximum swing of 4% is allowed between nominal supply voltage and the lower operating voltage margin, beyond which a voltage emergency occurs. However, the work in this paper is independent of a specific margin and the major findings of the paper remain unchanged across different margin settings.

Power delivery model. We evaluate three different packages. Quality factor (Q) is the ratio of the resonant frequency to the rate at which the package dissipates its energy. A larger Q gives rise to larger voltage swings for currents oscillating within the resonance band of frequencies. Applications with current fluctuations in the resonance band therefore suffer more from inductive noise with a high-Q package. The packages are labeled Pkg 1, Pkg 2 and Pkg 3. Details pertaining to the packages are shown in Table 2. Our baseline package is Pkg 1, which closely resembles characteristics of the Pentium 4 package [11]. Package Pkg 2 is representative of the package used in an earlier study [13], and its parameters are based on the Alpha 21264/21364 package. For comparisons, we include Package Pkg 3, which represents a bad package with very large quality factor.

Single-core vs. multi-core and multi-threaded architectures. We limit our evaluations in this paper to a single-core platform with an off-chip power delivery subsystem. Much of prior work is also within the context of single-core platforms, which allows comparative analysis of our scheme to others. Kim *et al.* and Gupta *et al.* have shown that voltage emergencies are problematic for multi-core platforms as well [7, 14]. The authors demonstrate that synchronous/in-phase operation of cores or chip-wide resonant behavior can cause voltage emergencies, and so can per-core power domains. We believe it is possible to extend our work to capture inter-core activity leading to emergencies by tracking additional events such as cache coherence messages and inter-thread synchronization primitives. And in the case of a multi-threaded architecture, it is possible to easily adapt the emergency capturing mechanism to be a part of the hardware's thread context. Building a predictor for a multi-core and multi-threaded architecture is beyond the scope of this paper and requires further investigations.

[1] *445.gobmk* input `13x13`, *456.hmmer*, *471.omnetpp*, *473.astar*, *434.zeusmp*, *453.povray* and *470.lbm* are omitted because SimpleScalar's $x86$ decoder does not support instruction encodings used by these benchmarks.

5. Predictor Accuracy Evaluation

A signature-based emergency predictor, in contrast to a sensor-based scheme, is broadly applicable across different combinations of microprocessor designs and power delivery subsystems with no need for fine-tuning, catering for the worst-case, or relying on soft thresholds. In this section, we demonstrate the robustness of signature-based prediction across different machine configurations assuming a signature size of 32 entries. We also demonstrate an ability to predict emergencies 16 cycles ahead of time with 90% accuracy.

Workloads. Applications exhibit different characteristics that drive the machine into different levels of activity and, therefore, varying rates of current draw. Figure 7(a) plots prediction accuracy across the spectrum of benchmarks from CPU2006. For benchmarks with multiple inputs, we present the average prediction accuracy across different inputs. The signatures enable high prediction accuracy with an average of 93% and a median of 94%. Voltage emergency signatures are able to handle a range of benchmarks from control-flow-intensive benchmarks like *403.gcc* and *400.perlbench* to memory-intensive benchmarks like *429.mcf*, and to *462.libquantum* that exhibit a large number of microarchitectural events such as cache misses. Overall, high prediction accuracy is observed across both the integer and floating-point benchmarks.

Tolerance. Figure 7(b) shows that when we pair power delivery packages Pkg 1, Pkg 2, and Pkg 3 with our baseline microprocessor design Arch 1 (Table 1), average prediction accuracy remains high (93%, 96%, and 95%, respectively) despite decreasing package quality. Signatures consistently enable emergency prediction with over 90% accuracy without specialization. By comparison, sensor-based schemes require careful configuration of soft thresholds [9]. When we pair package Pkg 1 with a simpler out-of-order processor Arch 2 (one with the same structure as that in Table 1, but with half-sized fetch and decode widths and half-sized buffers, queues, and caches), the accuracy of our predictor still remains high at 97%.

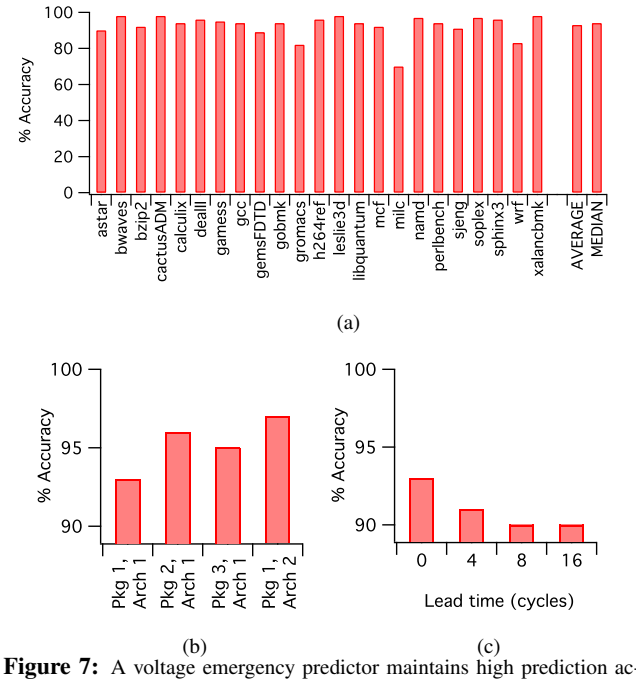


Figure 7: A voltage emergency predictor maintains high prediction accuracy across different (a) program types and (b) power delivery packages and microarchitecture combinations. (c) The predictor is also capable of predicting emergencies with sufficient lead time.

Lead time. Predicting an emergency with sufficient lead time enables the machine to throttle execution and successfully avoid an impending emergency. Figure 3(b) illustrates this notion of lead time using the Lead time label. Up to this point in the paper, we assumed a lead time of 0 cycles to initially validate that signatures are good predictors of emergencies. However, real systems require non-zero lead times to account for circuit delays and allow for throttling to take effect. To experiment with other lead times, we can erase trailing segments of the signatures that we capture. Figure 7(c) shows accuracy slightly degrades from 93% as lead time increases. However, even with 16 cycles of lead time, ample time to prevent an emergency, prediction accuracy remains high at 90%.

It is important to note that throttling cannot prevent all emergencies even when they are correctly predicted with 16 cycles of lead time. In such cases, the fail-safe mechanism must recover processor state and the machine incurs rollback penalties. However, our experimental data (not shown) verifies that the number of such emergencies is only 1% of the total emergencies that occur without throttling and resulting penalties are very low.

6. Performance Evaluation

An aggressive reduction in operating voltage margins can translate to higher performance or higher energy efficiency. Since performance and power are inextricably tied, we focus on clock frequency performance improvements. Assessing performance also enables straightforward accounting of penalties resulting from throttling and rollbacks. We evaluate the maximum attainable performance within the context of all runtime costs previously illustrated in Figure 3(a) and compare to a variety of idealized and non-ideal approaches. While the initial analysis makes optimistic assumptions in regards to hardware implementations of the voltage emergency predictor, Section 6.2 explores design tradeoffs and shows a resource-constrained predictor implementation that retains high accuracy and performance improvements.

Designers typically build in conservative margins (guardbands) to safeguard against potentially large voltage droops that can lead to timing violations. Such margins translate to clock frequency reductions and performance loss. Recent papers on industrial designs have shown that 15% to 20% operating voltage margins would be required to protect against voltage emergencies [3, 12]. Similarly, our analysis of our baseline system (Pkg 1 and Arch 1) reveals a worst-case droop of 13%.

The nearly-linear relationship between operating voltage and clock frequency facilitates translation of voltage margin reductions into performance gains. Based on detailed circuit-level simulations of an 11-stage ring oscillator consisting of fanout-of-4 inverters, we observe a 1.5x relationship between voltage and frequency at the PTM 32nm node [30]. This relationship is consistent with results reported by Bowman *et al.* [3], which show that a 10% reduction in voltage margins leads to a 15% improvement in clock frequency. While we use this 1.5x voltage-to-frequency scaling factor throughout the rest of the paper, we also observe a disconcerting trend across technologies. Simulation results reveal voltage-to-frequency scaling factors of 1.2x, 1.5x, 2.3x, and 2.8x for PTM nodes at 45nm, 32nm, 22nm, and 16nm, respectively. Given a slowdown in traditional constant-field scaling trends, sensitivity of frequency to voltage is growing, which further stresses the need for techniques that can efficiently reduce voltage noise in future processors.

Based on the 1.5x scaling factor, the 4% operating voltage margin assumed in this paper corresponds to a 6% loss in frequency. Similarly, a conservative voltage margin of 13%, sufficient to cover the worst-case droops observed, leads to 20% lower frequency. If we take this conservative margin as the baseline for comparisons and the 13% margin can reduce to 4% while avoiding voltage emergencies, the corresponding clock frequency improvement offers system performance gains of 17.5%. This sets the upper bound on maximum performance gains achievable. We make a simplifying assumption that frequency improvements directly translate to higher overall system performance.

6.1. Comparison of Schemes

To thoroughly evaluate the benefits of using our signature-based predictor, we compare it to variety of other schemes that also use throttling and/or checkpoint-recovery. We assume a half-rate throttling mechanism that gates every other clock cycle. For sensor-based schemes, we assume sensors are ideal with zero delay, and can instantly react to either resonant or single-event-based voltage emergencies.

Schemes		Performance Gain (%)
Predictor throttling	Oracle	14.2
	Voltage emergency signature	13.5
	Microarchitectural event	4.1
Ideal sensor throttling	2% soft threshold	2.2
	3% soft threshold	9.0
Explicit checkpoint and recovery		-13.0
Delayed commit and rollback (DeCoR)		13.0

Table 3: Performance comparison across different flavors of throttling and checkpoint-recovery for handling voltage emergencies.

For our predictor, we assume an unbounded prediction table with a voltage emergency signature predictor with 16 cycle lead time. Calculation of performance gains shown for each scheme begins with the maximum 17.5% gains possible, which then scales down by accounting for all performance overheads. Again, a conservative voltage margin of 13% allows for emergency-free, lower-frequency operation and is the common baseline for all comparisons. Table 3 shows the performance gains of all schemes.

Oracle predictor. To set an upper bound on the potential benefits of prediction-based schemes, we consider an oracle predictor. It throttles exactly when an emergency is about to occur, and it always prevents the emergency. It does not waste throttles nor does it incur rollback penalties. By removing all voltage emergencies, the resulting performance gain of 14.2%, is the best achievable performance while incurring only 2.9% throttling overhead.

Voltage emergency signature predictor. Our signature-based prediction scheme incurs performance overhead of 3.5% on average, due to throttling and rollbacks that are needed to detect emergencies and also due to emergencies that throttling cannot avoid. The slightly higher overhead translates to performance gain relative to our baseline of 13.5%, just 0.7% less than the oracle predictor.

Microarchitectural event predictor. We also evaluate a simpler prediction scheme that associates an emergency with the most recent microarchitectural event and the address of the instruction responsible for it [10]. Whenever that combination recurs, this scheme throttles execution to prevent another emergency. The prediction accuracy of this simple scheme is poor, translating to large amounts of unnecessary throttling that severely degrades performance. Large overheads limit performance gain to only 4.1% with this method.

Ideal sensor. Still using a 4% operating margin as the hard lower operating voltage margin, we evaluate sensor-based schemes for two soft voltage threshold settings, a conservative threshold of 2% and an aggressive one of 3%. We optimistically assume a 0-cycle sensor delay and that all emergencies that would occur after crossing the soft threshold are prevented. Note that an actual sensor would have a delay of several cycles and so would give poorer performance results. Despite the optimistic assumptions, performance gains for the the 2% and 3% soft thresholds are only 2.2% and 9.0%, respectively. These low gains are due to the high fraction of benign soft threshold crossings that lead to unnecessary throttling penalties, shown earlier in Figure 2(b).

Explicit checkpoint and recovery. Gupta *et al.* propose the use of checkpointing specifically for the purpose of handling voltage emergencies [9]. They demonstrate that explicit checkpoint-recovery schemes cannot be directly applied to handling voltage emergencies due to their high rollback costs. Our results confirm their claim. We observe a 13% performance loss when using an explicit checkpoint-recovery mechanism that has a 100-cycle rollback penalty.

Delayed commit and rollback. To overcome limitations of explicit checkpoint-recovery, Gupta *et al.* propose an implicit checkpointing scheme called DeCoR that speculatively buffers register file and memory updates until it has been verified that no emergency has occurred during a period long enough to detect an emergency [9]. The commit proceeds as usual unless an emergency is detected, in which case the machine rolls back and resumes execution at a throttled pace. We assume a 5-cycle sensor delay for DeCoR, which represents the best case as demonstrated by its designers.

DeCoR's performance gain is 13.0%, so our signature-based predictor outperforms it, but only slightly. However, the benefits of using a signature-based predictor outweigh using DeCoR for a general-purpose processor design. DeCoR's implicit checkpointing requires changes to traditional microarchitectural structures. In comparison, coarse-grained checkpoint-recovery is already shipping in production systems [1,26] and can serve multiple purposes ranging from boosting processor performance [15,16,27] to fault detection [25] and debugging [19]. A signature-based predictor leverages the coarse-grained checkpoint-recovery hardware, thereby retaining all the benefits of coarse-grained checkpoint-recovery while also reducing voltage emergencies.

Issue-rate staggering. Pipeline muffling [20, 23] and a floor-plan aware di/dt controller [17] both stagger issue rates to combat cycle-to-cycle high-frequency noise within individual microarchitectural units. In contrast, this paper considers inductive noise in the mid-frequency (10-100MHz) range that impacts the entire chip over periods of tens of cycles. As discussed in [23], issue-ramping strategies are not suitable for mid-frequency noise because ramping current over such a large number of cycles is not practical; these strategies are thus orthogonal to our approach.

6.2. Proof-of-Concept Implementation

Up to this point we assume unbounded resources for matching voltage emergency signatures. In this section, we show one way to implement a resource constrained predictor. Our implementation combines a content-addressable memory (CAM) with a Bloom filter. We discuss why this combination is more efficient than a CAM or a Bloom filter by itself. Using a 8KB table, we observe a performance gain of 11.1%, as compared to the 13.5% gain for the unbounded predictor (described in Section 6.1).

Prediction table. A prediction table is a hardware structure for recognizing voltage emergency signatures. Lookups in the prediction table happen whenever the processor updates the contents of the event history register. The processor combines the event sequence from the history register with the address of the last issued branch instruction to form a signature, and then tries to match that signature in the prediction table. If the match succeeds, the processor throttles execution to prevent a potential emergency. We assume the prediction table is managed by firmware.[2] When an emergency occurs, the firmware makes a signature by combining the contents of the event history register with the most recently issued branch address and enters it in the prediction table.

CAM. A CAM is a natural structure for implementing a prediction table. However, our analysis shows that at least 8,000 entries would be needed to achieve good performance. At 16 bytes per entry, such a large CAM would require too much area and power. With a smaller CAM, capacity misses could prevent emergencies from being detected, which could lead to severe rollback penalties.

Bloom filter. A Bloom filter is a compact lookup structure that saves space, but may sometimes return a false match. It is a probabilistic hash table that maps keys to boolean values, implemented using a bit vector and k hash functions. The procedure to add a key to the Bloom filter hashes the key k ways and sets the bits in the bit vector corresponding to the k indices returned by the hash functions. A key matches in the Bloom filter if and only if the bits for all k indices hashed from that key are set. With some probability, all of the indices for a key that has never been entered may nevertheless be set, in which case matching that key produces a false positive result.

For our purposes, false positives can be tolerated because they only affect performance, not correctness. However, we find that a Bloom filter by itself needs to be quite large to give acceptable performance. Smaller Bloom filters have higher false positive rates, and the resulting unnecessary throttling severely degrades performance. While a 64KB Bloom filter could yield a performance gain comparable to our unconstrained signatured-based predictor, that for a Bloom filter of a more practical size, such as 8KB, falls to less than 2%.

CAM plus Bloom filter. By screening the anchor PC components of signatures using a CAM, we can reduce the number of lookups in the Bloom filter, which reduces the number of times false positives cause throttling. In our experiments we observe that the working set of anchor PCs is small enough that a CAM is practical. Sizing the CAM appropriately is important, however, because capacity misses allow emergencies to happen, which leads to rollbacks. At CAM sizes of 32 and 64 entries, our results show that rollback penalties reduce performance gains by as much as 50% and 10%, respectively. But with a 128-entry CAM, the performance loss due to capacity misses is negligible.

[2]The use of firmware to manage the prediction table is consistent with systems in which firmware manages energy and deals with processor design errors [5, 18, 24, 28]. Firmware implementation details are beyond the scope of this paper.

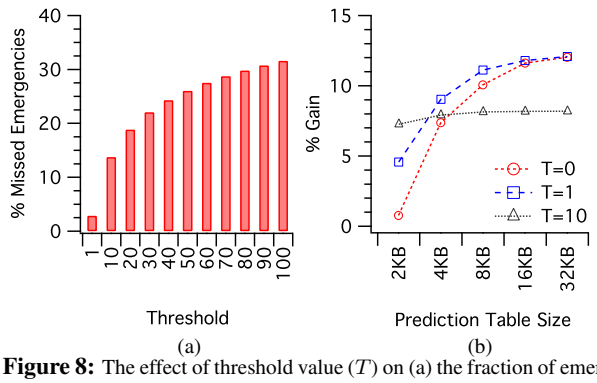


Figure 8: The effect of threshold value (T) on (a) the fraction of emergencies not handled by the predictor and (b) performance gains when voltage margin is reduced from a conservative 13% to an aggressive 4% setting.

Thresholds. The other way to reduce false positives is to keep the occupancy of the Bloom filter low. That can be done by excluding the less frequently occurring emergency signatures. The trade-off is that with higher thresholds, we miss more emergencies and incur more rollback costs. The firmware that manages the prediction table could at the same time profile signature occurrences and exclude those signatures whose occurrence counts fall below a chosen threshold.

To investigate the effects of thresholds, we used a prediction table combining a 128-entry CAM (one 32-bit address per entry) with a Bloom filter that uses three hash functions. Figure 8(a) shows that a threshold of one captures all but 2.8% of all emergencies. Larger thresholds cause so many emergencies to be missed that performance degradation due to rollbacks is severe.

Figure 8(b) shows the performance gains with different prediction table sizes for a variety of threshold values. For small table sizes, a higher threshold yields better performance because it reduces the false positive rate. With a 2KB prediction table size, performance gain is only 0.8% without a threshold (T=0). But a threshold of T=10 reduces throttles caused by false positives so much that performance gain increases to 7.3%, despite increased rollback penalties. On the other hand, as table size grows, the false positive rate drops so that lower thresholds are more attractive. With an 8KB prediction table size, performance gain for a threshold of T=10 is 3 percentage points less than that for a threshold of T=1, because false positives are reduced so much that rollback penalties dominate. With T=1 (which simply excludes all non-recurring emergency signatures), the performance gain for an 8KB table is 11.1%, as compared to the 13.5% gain for the unbounded prediction table described in Section 6.1.

7. Summary and Conclusions

With continued technology scaling, the inductive noise problem is an increasingly important design challenge. Several

architectural solutions have been proposed in the past to deal with inductive noise in processors. However, these solutions either have trouble guaranteeing correctness or they incur severe performance penalties. This paper proposes a novel *voltage emergency predictor* that learns to predict recurring voltage emergencies by collecting *signatures* of the program behavior and processor activity that leads to such emergencies. Our proposed predictor-based architecture uses the collected signatures to anticipate emergencies and proactively avoid them via throttling, while relying on a checkpoint-restart fall-back scheme already available in today's production systems to train the throttling predictor. Our signature-based voltage emergency predictor operates independently of sensor delays, package characteristics, and microarchitecture details, and it enables operation at aggressive voltage margins without compromising correctness. With an aggressive margin of 4%, it can enable a performance gain of as much as 13.5%, compared to 14.2% for an ideal oracle-based throttling mechanism.

Acknowledgments

We are thankful to our colleagues in industry and academia for the many discussions that have contributed to this work. We are also grateful to the anonymous reviewers for their comments and suggestions. This work is supported by gifts from Intel Corporation, and National Science Foundation grants CCF-0429782 and CSR-0720566. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] H. Ando and et al. A 1.3 GHz Fifth-Generation SPARC64 Microprocessor. In *In Proceedings of Design Automation Conference*, 2003.

[2] K. Aygun, M. J. Hill, K. Eilert, R. Radhakrishnan, and A. Levin. Power Delivery for High-Performance Microprocessors. *Intel Technology Journal*, 9, 2005.

[3] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. Lee, C. B. Wilkerson, S.-L. Lu, T. Karnik, and V. De. Energy-efficient and metastability-immune timing-error detection and instruction replay-based recovery circuits for dynamic variation tolerance. In *ISSCC 2008*, 2008.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a Framework for Architectural-level Power Analysis and Optimizations. In *27th Annual International Symposium on Computer Architecture*, 2000.

[5] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco. Software-based online detection of hardware defects: Mechanisms, architectural support, and evaluation. In *MICRO 2007*, 2007.

[6] E. Grochowski, D. Ayers, and V. Tiwari. Microarchitectural Simulation and Control of di/dt-induced Power Supply Voltage Variation. In *Int'l Symposium on High-Performance Computer Architecture*, 2002.

[7] M. S. Gupta, J. L. Oatley, R. Joseph, G.-Y. Wei, and D. M. Brooks. Understanding voltage variations in chip multiprocessors using a distributed power-delivery network. In *DATE*, 2007.

[8] M. S. Gupta, K. Rangan, M. D. Smith, G.-Y. Wei, and D. M. Brooks. Towards a Software Approach to Mitigate Voltage Emergencies. In *ISLPED '07*, 2007.

[9] M. S. Gupta, K. Rangan, M. D. Smith, G.-Y. Wei, and D. M. Brooks. DeCoR: A Delayed Commit and Rollback Mechanism for Handling Inductive Noise in Processors. In *HPCA '08*, 2008.

[10] M. S. Gupta, V. J. Reddi, M. D. Smith, G.-Y. Wei, and D. M. Brooks. An event-guided approach to handling inductive noise in processors. In *DATE*, 2009.

[11] Intel. Intel Pentium 4 Processor in the 423 Pin/Package /Intel 850 Chipset Platform, 2002.

[12] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie. Comparison of Split-Versus Connected-Core Supplies in the POWER6 Microprocessor. In *ISSCC 2007*, 2007.

[13] R. Joseph, D. Brooks, and M. Martonosi. Control Techniques to Eliminate Voltage Emergencies in High Performance Processors. In *Int'l Symposium on High-Performance Computer Architecture*, 2003.

[14] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *HPCA*, 2007.

[15] N. Kirman, M. Kirman, M. Chaudhuri, and J. Martinez. Checkpointed Early Load Retirement. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005.

[16] J. F. Martínez, J. Renau, M. C. Huang, M. Prvulovic, and J. Torrellas. Cherry: Checkpointed Early Resource Recycling in Out-of-order Microprocessors. In *International Symposium on Microarchitecture (MICRO)*, 2002.

[17] F. Mohamood, M. Healy, S. Lim, and H.-H. S. Lee. A Floorplan-Aware Dynamic Inductive Noise Controller for Reliable Processor Design. In *MICRO*, 2006.

[18] S. Narayanasamy, B. Carneal, and B. Calder. Patching processor design errors. In *ICCD*, 2006.

[19] S. Narayanasamy, G. Pokam, and B. Calder. BugNet: Continuously Recording Program Execution for Deterministic Replay Debugging. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, 2005.

[20] M. D. Pant, P. Pant, D. S. Wills, and V. Tiwari. An architectural solution for the inductive noise problem due to clock-gating. In *ISLPED*, 1999.

[21] E. Perelman, G. Hamerly, M. V. Biesbrouck, T. Sherwood, and B. Calder. Using simpoint for accurate and efficient simulation. In *SIGMETRICS '03*, New York, NY, USA, 2003. ACM.

[22] M. Powell and T. N. Vijaykumar. Exploiting Resonant Behavior to Reduce Inductive Noise. In *ISCA*, 2004.

[23] M. D. Powell and T. N. Vijaykumar. Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise. In *Int'l Symposium on Low Power Electronics and Design*, 2003.

[24] S. Sarangi, S. Narayanasamy, B. Carneal, A. Tiwari, B. Calder, and J. Torrellas. Patching processor design errors with programmable hardware. *IEEE Micro*, 2007.

[25] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, and T. Austin. Ultra Low-Cost Defect Protection for Microprocessor Pipelines. *ASPLOS-XII*, 2006.

[26] T. J. Slegel, R. M. A. III, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb. Ibm's s/390 g5 microprocessor design. *IEEE Micro*, 19, 1999.

[27] D. J. Sorin, M. M. K. Martin, M. D. Hill, and D. A. Wood. Fast Checkpoint/Recovery to Support Kilo-instruction Speculation and Hardware Fault Tolerance. Computing science technical report, University of Wisconsin-Madison, 2000.

[28] I. Wagner, V. Bertacco, and T. Austin. Shielding against design flaws with field repairable control logic. In *IEEE/ACM DAC*, 2006.

[29] N. J. Wang and S. J. Patel. ReStore: Symptom-Based Soft Error Detection in Microprocessors. *IEEE Trans. Dependable Secur. Comput.*, 3(3), 2006.

[30] W. Zhao and Y. Cao. Predictive technology model for sub-45nm early design exploration. *ACM JETC*.