Predicting Voltage Droops Using Recurring Program and Microarchitectural Event Activity

Shrinking feature size and diminishing supply voltage are making circuits more sensitive to supply voltage fluctuations within a microprocessor. If left unattended, voltage fluctuations can lead to timing violations or even transistor lifetime issues. A mechanism that dynamically learns to predict dangerous voltage fluctuations based on program and microarchitectural events can help steer the processor clear of danger.

Vijay Janapa Reddi
Meeta Gupta
Glenn Holloway
Michael D. Smith
Gu-Yeon Wei
David Brooks
Harvard University

Power-constrained processor design is impacting processor reliability. Power-reduction techniques (such as clock gating), when aggressively applied to constrain power consumption, are leading to large current swings in the processor. When coupled with the nonzero impedance of a power-delivery subsystem, these current swings can cause voltage to fluctuate beyond safe operating margins. Chip designers have traditionally dealt with such dangerous fluctuations, called *voltage emergencies*, by optimizing for the worst-case voltage flux, allocating sufficiently large voltage margins to avoid timing violations.

Unfortunately, on-chip voltage fluctuations and the margins they require are getting worse as feature size scales. A recent paper analyzing the POWER6 microprocessor's emergency-prone activity shows that required timing margins translate to operating voltage margins of nearly 20 percent of the nominal supply voltage (~200 mV for a nominal voltage of 1.1 V).[1] Such conservative operating voltage margins guarantee robust operation, but can severely degrade performance due to the lower operating frequencies or higher power budgets.

To reduce the gap between nominal and worst-case operating voltages, we propose a *voltage emergency predictor* to identify imminent emergencies. A voltage emergency predictor learns recurring activity leading to voltage emergencies during runtime and prevents subsequent occurrences of these emergencies via execution throttling. Designers can choose the form of throttling. Figure 1 is a block diagram representation of this predictor.

A voltage emergency predictor anticipates emergencies using *voltage emergency signatures*.

0272-1732/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

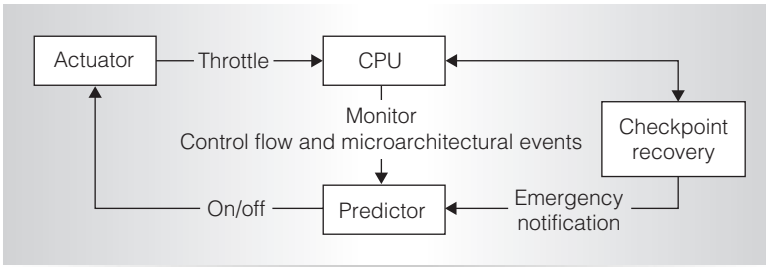

Figure 1. Block diagram showing activity-based emergency detection, prediction, and prevention.

Each signature is an interleaved sequence of control-flow events and microarchitectural events leading up to an emergency. The predictor captures a voltage emergency signature when an emergency first occurs by taking a snapshot of relevant event history and storing it in the predictor. Built-in checkpoint-recovery then rolls the machine back to a known correct state and resumes execution. Checkpoint-recovery acts as a fail-safe, albeit at a high penalty (hundreds to thousands of cycles). It is meant for infrequent use when the predictor fails. Subsequent occurrences of the same emergency signature cause the predictor to throttle execution (costing between 4 and 8 cycles) and prevent the impending emergency. By doing so, the predictor enables aggressive timing margins to maximize efficiency.

Voltage emergency prediction

To successfully predict and prevent emergencies, we must first establish leading indicators of dangerous voltage fluctuations. Our study found a strong relationship between microarchitectural events and current and voltage fluctuations within the microprocessor. We considered several microarchitectural parameters—such as the number of entries in the reorder buffer and the instruction fetch and load/store queues—along with microarchitectural events such as cache misses and pipeline flushes, and found the latter are more indicative of emergency-prone activity.

Microarchitectural events

Figure 2a is a snapshot of pipeline activity for benchmark art from SPEC CPU2000 over 430 cycles. The figure illustrates microarchitectural events for the cache and branch predictor along with the processor's current and voltage activity. In the "pipeline stall" part of the figure, we observe an L2 cache miss (shown as a vertical bar in the "L2 miss" subgraph). During the time it takes to service the L2 miss, pipeline activity ramps down, as marker 1 in the "current" profile shows. However, when the L2 miss data is available, functional units become busy and current activity suddenly increases (marker 2). This steep increase in current causes the voltage to temporarily dip below the voltage

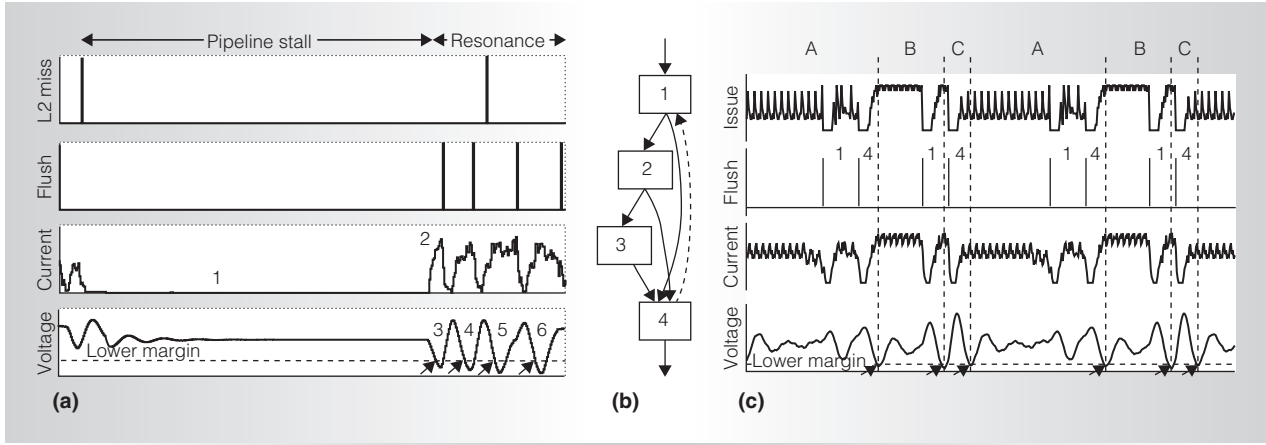

Figure 2. Snapshot of pipeline activity for the art benchmark from the SPEC CPU2000 suite illustrating how cache misses and repeated branch mispredictions lead to isolated and resonant emergencies, respectively (a). An emergency-prone loop in gcc (b), which causes a repeating emergency-prone activity (c). Arrows in the "lower margin" areas of (a) and (c) indicate that a voltage emergency has just occurred.

margin (marker 3) because of parasitic inductance in the power delivery subsystem.

Additionally, microarchitectural events that cause periodic high and low current activity can cause a resonance build up of voltage if the period coincides with the power delivery subsystem's resonance frequency. The "resonance" portion of Figure 2a shows multiple pipeline flush events occurring in close proximity. Pipeline flush events cause a sudden decrease in activity and are followed by a rush of activity as instructions are rapidly issued along the correct program. The close distribution of these events leads to a resonating effect that results in rapid current fluctuations. These successive fluctuations not only cause the voltage to swing, but also progressively increase in amplitude from one event to another (markers 4, 5, and 6).

Activity history

Whether or not a microarchitectural event at a particular location causes an emergency depends on the activity just before and just after the event. Even a small loop such as that in Figure 2b, extracted from the `init_regs` function in benchmark gcc of SPEC CPU20006, can have behavior phases with markedly different activity patterns. Figure 2c is a snapshot of activity within that loop over 880 cycles. It shows three repeating phases of the loop. Phase A uses paths $1 \rightarrow 4$ and $1 \rightarrow 2 \rightarrow 4$, whereas phase B uses only path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. The average issue rate of phase A is relatively low, whereas that of phase B is quite high. Branch mispredictions at the end of basic block 1 cause the flush events labeled 1. Those in phase B, where the issue rate is high, always cause emergencies, whereas those in phase A never do. Therefore, the interleaving of program flow and microarchitectural events is important.

We encapsulate the activity leading up to an emergency in a voltage emergency signature, comprising both control flow and microarchitectural events. Tracking the instruction stream captures a program's dynamic path. Capturing the interleaving of program path with microarchitectural events generates a representative snapshot of the corresponding dynamic current and voltage activity resulting from program interactions with the underlying microarchitecture.

Capturing emergency signatures

The following logic is necessary to capture voltage emergency signatures. Because voltage emergencies contribute to timing faults, we assume all necessary logic is carefully designed with sufficient conservative timing margins. But because these structures are not timing critical, there are no performance implications. Any state corruption in the logic leads to incorrect predictions, and will therefore only affect system performance because of unnecessary throttling; it will not violate correctness guarantees.

Emergency detector

With our scheme, capturing a voltage emergency signature requires an emergency to occur at least once. We therefore need a mechanism to monitor operating voltage margin violations. In our implementation, we rely on a voltage sensor.

Fail-safe mechanism

Processor state can be corrupted as emergencies occur because voltage emergencies induce timing faults. We therefore require a fail-safe to recover from emergencies. The fail-safe mechanism initiates a recovery whenever the sensor detects an emergency. We assume coarse-grained checkpoint recovery as our fail-safe because such a mechanism is already shipping in today's production systems.[2,3]

Event history register

The predictor relies on a shift register to capture and subsequently predict the interleaving sequences of control flow instructions and architectural events that will give rise to an emergency. A voltage emergency signature is a snapshot of this history register when the detector detects an emergency.

Emergency signature semantics

A voltage emergency signature precisely indicates 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

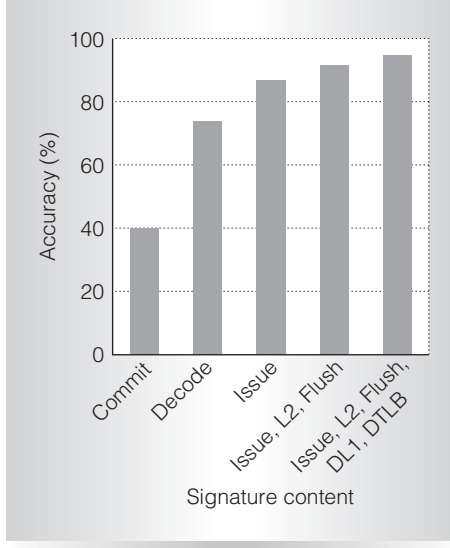

Figure 3. Prediction accuracy for different signature types. Accuracy improves as signature contents more closely represent machine activity.

of predicted emergencies that become actual emergencies.

Contents

The information in a voltage emergency signature must correspond to parts of the execution engine that experience large current draws, as well as dramatic spikes in current activity. For instance, control flow can be tracked 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 speculation's impact 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.

Figure 3 shows the accuracies of different signature types (assuming a signature size of 32 entries, which we discuss next). The predictor's prediction accuracy is 40 percent if it tracks only control sequences. But we can achieve 72 percent accuracy by tracking information at the decode stage, because this stage captures the speculative control flow path. We can further improve accuracy by tracking control flow at the issue stage, because we 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 2c immediately following pipeline flush events).

The last two bars in Figure 3 show accuracy improvements from tracking microarchitectural event activity as well. The second-to-last bar represents the effect of capturing events that have the potential to induce large voltage swings—pipeline flushes and secondary (L2) cache misses. We achieve an additional 5 percent improvement by accounting for flushes and L2 misses. Capturing the more frequently occurring events, such as data translation lookaside buffers (DTLB) and data L1 misses, contributes another improvement of around 4 percent. Microarchitecture perturbations resulting from instruction cache activity—that is, instruction L1 and instruction TLB (ITLB)—are negligible and do not improve accuracy.

In our design, we track information at the pipeline's issue stage along with microarchitectural-event activity. More formally, we track control flow instructions at the execution stage, along with L1 and L2 cache and TLB misses. Lastly, we also record pipeline flushes.

Size

Accuracy depends not only on recording the correct interleaving of events, but also on balancing the amount of information we keep. Accuracy improves as the history register's length increases. However, increasing the length of information beyond a certain count can be detrimental. Numerous entries in a signature can cause unnecessary differentiation between similar signatures—that is, signatures whose most recent entries are identical and whose older entries are different, but not significantly.

Figure 4 shows that prediction accuracy improves as signature size increases. Accuracy is only 13 percent on average for a signature containing only one entry, which supports our earlier claim that voltage emergencies do not depend solely 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 percent for a signature size of 64 entries.

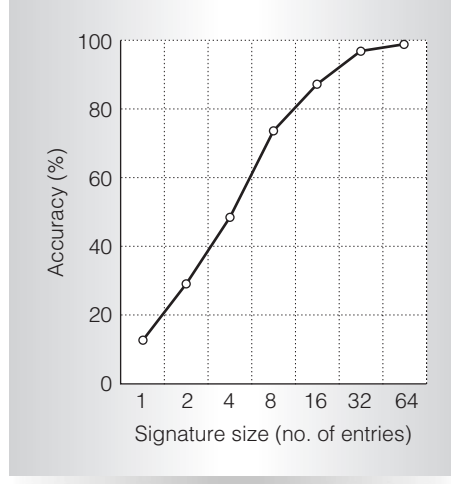

Figure 4. Prediction accuracy for various signature sizes. Accuracy is sensitive to the amount of information tracked within a signature.

Emergency signature compaction

Hardware implementations are resource constrained. Thus, the number of bits representing a signature in a realistic hardware implementation matters. We devised signature encoding and compaction techniques to constrain the resource requirements necessary for implementation.

Signature encoding

To avoid large overheads, we use 3-bit encoding per entry in the event history register. The 3-bit encoding compactly captures all of the relevant information, consisting of different processor events, and accounts for the edge taken by each branch (that is, it encodes fall-through paths as 000 versus 001 for taken edges).

However, 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 PCs have the added benefit of implicitly providing the complete path information leading up to the most recent event in the history register.

The resulting compact representation is 16 bytes long (4 bytes for the anchor PC and 12 bytes for a signature size of 32 entries with 3 bits per entry).

Signature folding

We can further reduce hardware overhead by condensing 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

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

If the signatures are identical, s is 1. If no two corresponding elements are the same, s is 0. 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. We then use the resulting partition instead of the original signature set.

The number of recurring signatures per benchmark varies significantly. Benchmark 403.gcc has almost 87,000 signatures that repeatedly give rise to emergencies. Benchmark 462.libquantum is at the other end of the spectrum, with only 39 signatures. Applying signature compaction on 403.gcc reduces the number of signatures to 29,000.

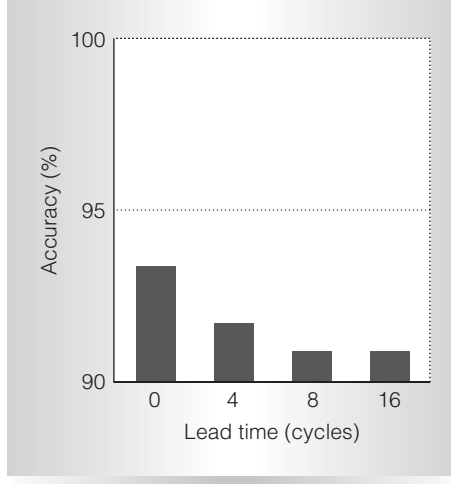

Figure 5. Prediction accuracy for CPU2006 benchmarks. Accuracy is high even when predicting several cycles ahead of time.

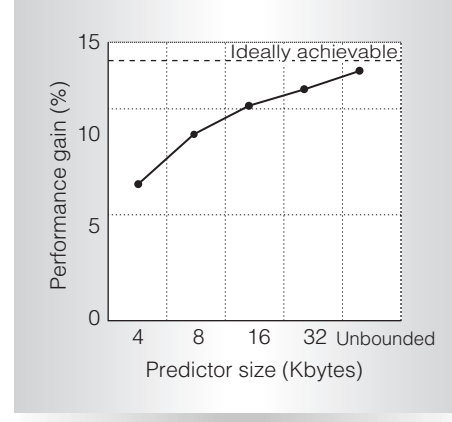

Figure 6. Performance gain using a signature-based throttling scheme. We achieve substantial gains using an aggressive 4 percent voltage margin.

Overall, compaction reduces the number of signatures by more than 61 percent. The biggest winners are benchmarks exhibiting numerous signatures.

Signature-based throttling

To operate future processors with aggressive voltage margins, we propose using voltage emergency signatures to anticipate emergencies, and using the prediction to decide when to actuate throttling to prevent an emergency. We demonstrate the signature-based predictor's capabilities using simulations of a representative superscalar microprocessor in which we treat fluctuations beyond 4 percent of nominal voltage as voltage emergencies.

Accuracy

The predictor must anticipate an emergency accurately and do so with sufficient lead time for throttling to take effect. Signatures predict emergencies with an average accuracy of 93 percent across the entire spectrum of CPU2006 benchmarks, as the 0-cycle lead time bar in Figure 5 illustrates. A lead time of 0 cycles optimistically assumes there is no delay to actuate throttling to prevent an emergency, thus representing an ideal scenario. However, real systems require nonzero lead times to account for circuit delays. As lead time increases, accuracy degrades just slightly from 93 percent, as Figure 5 shows. Even with 16 cycles of lead time, which is ample time to predict and prevent an emergency, accuracy remains high at 90 percent.

Throttling cannot prevent all emergencies, even if prediction accuracy is high. In such cases, the fail-safe checkpoint-recovery mechanism recovers processor state, albeit at much higher penalty. But we found the number of such emergencies is only 1 percent of all emergencies that occur without throttling. Therefore, the associated total recovery penalty is very low in our quantitative analysis.

Gains

An aggressive reduction in operating voltage margins translates into higher performance or better energy efficiency. However, throttling penalties to prevent emergencies and checkpoint-recovery rollbacks to train the predictor offset the benefits to some degree. Because performance and power are inextricably tied, we focus on clock frequency improvements to demonstrate overall effectiveness. Based on a 1.5× relationship between voltage and frequency at the predictive technology model (PTM) 32-nm node,[4] we observe an ideal performance gain of 14.2 percent using an oracle throttling scheme (see Figure 6). By comparison, the voltage emergency predictor comes to within 0.7 percentage points of the ideal scheme, assuming infinite or *unbounded*

resources to implement the predictor. But even under strict physical resource constraints, an intelligent bloom filter-based predictor ranging in size between 4 and 32 Kbytes delivers substantial gains.

Robustness

An added benefit of signature-based emergency prediction is that the predictor does not require fine tuning based on specifics of the microarchitecture nor the power delivery subsystem, as is the case with prior work. The current and voltage activity within a microprocessor are products of machine utilization that are specific to running workload dynamic demands. Capturing that activity in the form of emergency signatures lets the predictor 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.

Long-term impact and vision

According to International Technology Roadmap for Semiconductors (ITRS) projections, nominal supply voltage is gradually scaling down with newer process technologies, but threshold voltage scaling has all but stopped. Consequently, circuit delay sensitivity to margins is increasing with each technology node. Figure 7 plots peak frequency at different voltage margins across four PTM technology nodes (45 nm, 32 nm, 22 nm, and 16 nm) based on detailed circuit-level simulations of an 11-stage ring oscillator consisting of fanout-of-4 inverters. The plot indicates that at today's 32-nm node, a 20 percent voltage margin translates to a 33 percent frequency degradation. As technology nodes continue to evolve, the situation gets much worse. Practical limitations on reducing power delivery impedance combined with large current fluctuations make the industry-standard practice of overprovisioning voltage margins inefficient in the longer term.

Voltage emergency prediction demonstrates that voltage fluctuations in a processor are not random, as previously thought. Rather, it is recurring program and corresponding microarchitectural activity combined with the underlying power delivery subsystem's

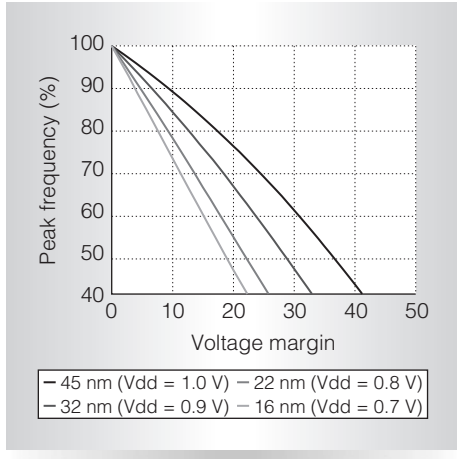

Figure 7. Peak frequency at different voltage margins for four technology nodes. Margins increasingly limit frequency as technology scales.

interactions that repeatedly give rise to emergencies. This discovery can enable new solutions to mitigate voltage noise. Our signature-based predictor is a specific hardware instantiation exploiting the recurring nature of emergencies. But as technology innovation continues and software layers such as virtualization become an integral part of the hardware platform, voltage emergency prediction can enable software-layer solutions to go beyond just repeatedly throttling to prevent emergencies. Software can eliminate emergencies altogether.

Hardware-only techniques do not fully exploit the repetitive nature of emergencies. To help demonstrate this, Figure 8 shows the number of distinct static program locations responsible for all emergencies in the program across three benchmark suites. The log-scale plot indicates that on average just a few hundred static program addresses are responsible for several hundreds of thousands of emergencies that manifest at runtime. A few emergency-prone code regions are responsible for almost the entire emergency problem. Unfortunately, hardware cannot leverage this information. Even an ideal oracle hardware prevention mechanism incurs recurring throttling penalties—at least one throttle per emergency.

Instead of letting hardware repeatedly, and consequently inefficiently, prevent voltage emergencies, voltage emergency prediction

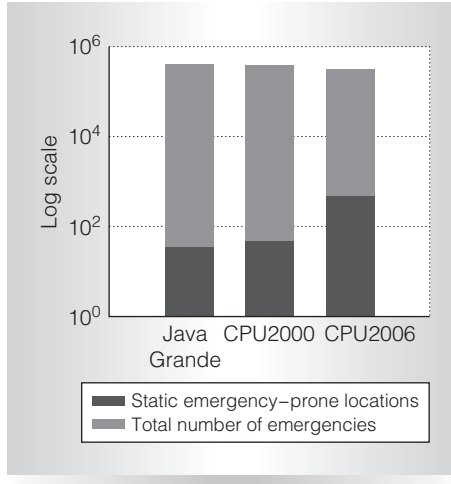

Figure 8. The number of static program locations responsible for all emergencies in a system. A few hundred static program locations cause all emergencies.

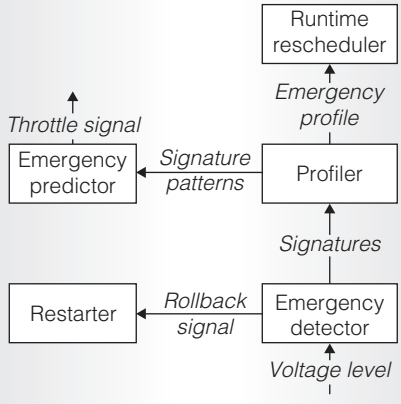

Figure 9. Our envisioned voltage emergency suppression system. Our system eliminates emergencies by detecting them and restructuring the runtime code for the emergency signature, or by predicting and avoiding emergencies by throttling processor activity.

can enable a hardware-software collaborative architecture that attempts to eliminate emergency-prone static program locations through code restructuring. By eliminating emergencies, software can run more efficiently because of fewer execution rollbacks and throttles.

In essence, optimizing away voltage emergencies is analogous to removing cache misses or branch mispredictions to achieve better performance. To enable software architects to perceive voltage noise as an opportunity for runtime performance improvement, the underlying hardware must provide pertinent information so that code restructuring efforts can be made intelligently. Information pertaining to an emergency is available in its corresponding voltage emergency signature. Preliminary efforts successfully demonstrate that using signatures to restructure emergency-prone code can eliminate more than 60 percent of all emergencies.[5]

Figure 9 illustrates our vision for emergency elimination. The system has a *detector* that triggers a *restarter* (such as the general-purpose, high-penalty checkpoint-recovery mechanism) to roll back execution when it detects an emergency. The detector then feeds an emergency *profiler* (either in software or hardware) with the corresponding voltage emergency signature. The profiler accumulates signatures to guide a code *rescheduler* (in software) capable of eliminating recurring emergencies via runtime code restructuring. The restructuring algorithms can extract valuable information embedded within the signatures—such as the dynamic control flow and sequence of microarchitectural events leading to an emergency—to help remove the emergency-prone hotspot. Code restructuring might not always succeed. In such cases, the profiler arms the low-penalty signature-based *emergency predictor* to instead predict and suppress recurring emergencies by throttling processor activity.

Considering voltage noise's impact on processor efficiency, aggressive operating voltage margins are becoming increasingly necessary. Characterizing the recurrence of voltage emergencies in terms of code behavior lets us not only predict and prevent emergencies intelligently, but also explore long-term solutions such as software-based elimination of voltage emergencies. MICRO

References

1. N. James, ''Comparison of Split-Versus Connected-Core Supplies in the POWER6 Microprocessor,'' *Solid-State Circuits Conf., Digest of Technical Papers* (ISSCC 07), IEEE Press, 2007, pp. 298-299.

2. H. Ando et al., ''A 1.3-GHz Fifth Generation SPARC64 Microprocessor,'' *Proc. 40th Ann. Design Automation Conf.* (DAC 03), ACM Press, 2003, pp. 702-705.
3. T. Slegel et al., ''IBM's S/390 G5 Microprocessor Design,'' *IEEE Micro,* vol. 19, no. 2, 1999, pp. 12-23.
4. W. Zhao and Y. Cao, ''Predictive Technology Model for Nano-CMOS Design Exploration,'' *ACM J. Emerging Technologies in Computing Systems,* vol. 3, no. 1, 2007, article 1.
5. V.J. Reddi et al., ''Software-Assisted Hardware Reliability: Abstracting Circuit-Level Challenges to the Software Stack,'' *Proc. 46th Ann. Design Automation Conf.* (DAC 09), ACM Press, 2009, pp. 788-793.

Vijay Janapa Reddi is a PhD candidate in the computer science department at Harvard University. His research interests are in the area of computer systems. He is specifically interested in applying his background in virtual machines, runtime compiler systems, and architecture to address processor power and reliability challenges via hardware-software codesign. Janapa Reddi has an MS in computer engineering from the University of Colorado, Boulder.

Meeta S. Gupta is a research staff member in the reliability and power-aware microarchitecture group at IBM TJ Watson, Yorktown Heights. Her interests include power-aware, reliable microarchitecture design, and high-performance computing. Gupta has a PhD in electrical engineering from Harvard University.

Glenn Holloway is a staff member in the Electrical and Computer Science Department at Harvard University. His technical interests are code optimization, computer architecture, and circuit design. Holloway has an MS in physics from Harvard University.

Michael D. Smith is a John H. Finley, Jr., Professor of Engineering and Applied Sciences and the Dean of the Faculty of Arts and Sciences at Harvard University. His research interests include dynamic optimization, machine-specific and profile-driven compilation, high-performance computer architecture, and practical applications of security. Smith has a PhD in electrical engineering from Stanford University.

Gu-Yeon Wei is a Gordon McKay Professor of Electrical Engineering in the School of Engineering and Applied Sciences at Harvard University. His research interests span several areas: high-speed, low-power link design; mixed-signal circuits for communications; power regulation circuitry and management; codesign of circuits and architecture for high-performance and embedded processors to address process-voltage-temperature variability and power consumption that plague nanoscale CMOS technologies; and developing an electronic brain and nervous system for the Harvard RoboBees project. Wei has a PhD in electrical engineering from Stanford University.

David Brooks is a professor of computer science at Harvard University. His research interests include architectural and software approaches to address power, thermal, and reliability issues for embedded and high-performance computing systems. Brooks has a PhD in electrical engineering from Princeton University.

Direct question and comments Vijay Janapa Reddi at 33 Oxford St (Room 307/ Brooks Group), Harvard Univ., Cambridge, MA 02138; vj@eecs.harvard.edu.