A Cost-effective Substantial-impact-filter Based Method to Tolerate Voltage Emergencies

Songjun PAN[†‡], Yu HU[†], Xing HU[†‡], and Xiaowei LI[†*]
[†]Key Laboratory of Computer System and Architecture,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China, 100190
[‡]Graduate University of Chinese Academy of Sciences, Beijing, P.R. China, 100049
{pansongjun, huyu, huxing, lxw}@ict.ac.cn

Abstract*—Supply voltage fluctuation caused by inductive noises has become a critical problem in microprocessor design. A voltage emergency occurs when supply voltage variation exceeds the acceptable voltage margin, jeopardizing the microprocessor reliability. Existing techniques assume all voltage emergencies would definitely lead to incorrect program execution and prudently activate rollbacks or flushes to recover, and consequently incur high performance overhead. We observe that not all voltage emergencies result in external visible errors, which can be exploited to avoid unnecessary protection. In this paper, we propose a substantial-impact-filter based method to tolerate voltage emergencies, including three key techniques: 1) Analyze the architecture-level masking of voltage emergencies during program execution; 2) Propose a metric intermittent vulnerability factor for intermittent timing faults (IVF_{itf}) to quantitatively estimate the vulnerability of microprocessor structures (load/store queue and register file) to voltage emergencies; 3) Propose a substantial-impact-filter based method to handle voltage emergencies. Experimental results demonstrate our approach gains back nearly 57% of the performance loss compared with the once-occur-then-rollback approach.

I. Introduction

Advances of integrated circuit technology enable smaller feature size and lower voltage threshold for performance improvement and energy reduction, which, however, result in tighter noise margin [1]. In general, power-constrained designs are more sensitive to inductive noise (*L*di/dt*) due to low supply voltage and large current swing. Inductive noises, which mean current variation in a small time scale, will incur supply voltage fluctuations due to parasitic inductance and none-zero impedance of power delivery subsystem (PDS). When the supply voltage variation is beyond the allowed voltage threshold, a voltage emergency occurs and usually leads to timing violations by slowing logic circuits. The reliability issue caused by voltage emergencies has become a big challenge for microprocessor design.

To address the reliability issue, microprocessor designers have to set a conservative timing margin considering the worst case to ensure system reliability. However, conservative timing margin would result in significant performance degradation. A recent study shows that the POWER6 microprocessor has about 200mV drop at the supply voltage of 1.1V, causing nearly 20% frequency reduction [2]. In order to provide a constant supply voltage, designers try to add a hierarchy of decoupling capacitors and voltage regulators to reduce the impedance of PDS. This method maintains a steady supply voltage over a wide range of frequencies but at the cost of high area overhead and severe leakage power dissipation. For example, the decoupling capacitors occupy about 20% of the die area in Alpha 21264 microprocessors [3].

Recently, several sensor-based methods have been proposed at architecture level to deal with voltage emergencies [4], [5], [6]. These methods are based on voltage or current sensors to detect the upcoming voltage margin violations. Prior work also shows that voltage emergencies are closely related to microarchitecture events (such as L2 cache misses and TLB misses) and program control flow instructions [7], [8]. A variety of microarchitecture events along with control flow instructions that lead to voltage emergencies are recorded as signatures to predict the reoccurrence of voltage emergencies [9]. If a possible voltage emergency is detected or predicted, pipeline throttling is activated to prevent its occurrence. Once a voltage emergency eventually occurs, a rollback is invoked to recover microprocessor states from a pre-stored checkpoint. The instantly reacting checkpoint/rollback approach assumes that every voltage emergency would definitely manifest itself in external visible outputs and finally affects system reliability; hence this approach can protect systems from all voltage emergencies but at a heavy performance cost due to high frequency of rollbacks. We observe that not all voltage emergencies lead to erroneous program outputs. A rollback will be triggered only when voltage emergencies actually corrupt architecture states, and those voltage emergencies having no impact on program execution will not be handled to reduce performance overhead.

To analyze the impact of voltage emergencies on program execution, we first establish an intermittent timing fault model for voltage emergency induced timing violations, and then propose an estimation metric *intermittent vulnerability factor for intermittent timing fault* (IVF_{itf}). IVF_{itf} reflects the architecture-level masking effect of different microprocessor structures to voltage emergencies. We compute IVF_{itf} for two structures load/store queue (LSQ) and register file (REG). With the guide of IVF_{itf}, we further propose a substantial-impact-filter based method to tolerate voltage emergencies. To

*To whom correspondence should be addressed.
This work was supported in part by National Natural Science Foundation of China (NSFC) under grant No. (61076018, 60633060, 60803031, 60831160526, and 60921002), in part by National Basic Research Program of China (973) under grant No. 2011CB302503.

978-3-9810801-7-9/DATE11/©2011 EDAA

the authors' knowledge, this is the first attempt to tolerate voltage emergencies by exploiting the inherent architecture-level masking effect. Our experimental results show that the averaged IVF_{itf} for LSQ and REG across a subset of SPEC CPU2000 benchmarks are 14.8% and 31.7%, respectively. Besides, our substantial-impact-filter based method can significantly improve system reliability while gains back nearly 57% of performance loss compared with the once-occur-then-rollback approach.

The main contributions of this paper are:

- We observe that not all voltage emergencies affect program execution. In fact, only a small number of voltage emergencies eventually corrupt architecture states. We analyze the root causes that make some voltage emergencies to be masked during program execution.
- After recognizing the architecture-level masking of voltage emergencies, we build an intermittent timing fault model for voltage emergency induced timing violations, and then propose a metric IVF_{itf} along with its computation method to quantify the vulnerability of different microprocessor structures to voltage emergencies.
- In order to gain back performance loss due to unnecessary protection, we propose a substantial-impact-filter based method to invoke rollbacks only when voltage emergencies lead to wrong architecture states.

The organization of the remaining part of this paper is as follows. Section II introduces our key observation that motivates this work. Section III presents the intermittent timing fault model and the IVF_{itf} computing method for different microprocessor structures. Section IV shows our substantial-impact-filter based method. Our experimental methodology is described in Section V, followed by our experimental results in Section VI. Finally we conclude the paper in Section VII.

II. Motivation

Voltage emergencies usually affect operation speed of transistors and thus cause long propagation delay. If the delay exceeds the allowed timing margin, a timing violation occurs and will affect program execution. Structures in timing-insensitive zone, such as L1/L2 caches, are protected by ECC or parity code and will not be affected by voltage emergencies. Structures in critical paths of microprocessors, however, will be more sensitive to voltage emergencies. Fig. 1 illustrates an example of supply voltage variation in LSQ and the related delay when executing *bzip2* for a short interval. During this interval, voltage emergencies occur six times and all are caused by L2 cache misses. A L2 cache miss results in a long period of pipeline hibernation, and then a sudden increase in activity occurs when the L2 cache miss returns. We further compute the incurred delay with alpha-power model [21]. As can be seen, voltage emergencies lead to significant timing violations. For all these timing violations, we need to analyze which one will affect program execution. The upper part of Fig. 1 shows the impact of timing violations. Logic value '1' indicates a timing violation affects program execution while logic value '0' indicates no impact. Among these timing violations shown

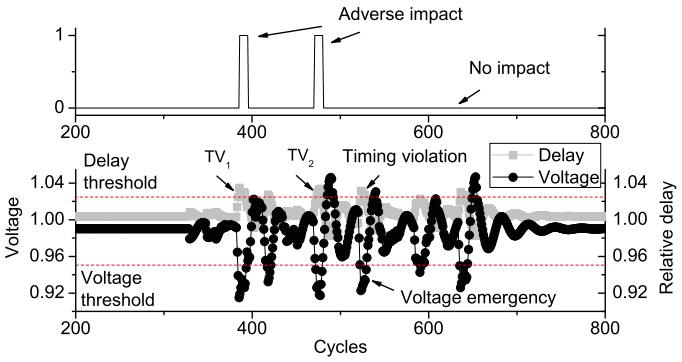

Fig. 1. The impact of voltage emergencies on program execution.

in Fig. 1, only two timing violations (TV_1 and TV_2) corrupt the value in LSQ and others have no impact on program execution. Several reasons can help to explain for this phenomenon: first, a timing violation not propagating to LSQ or not changing architecture correct execution (ACE) bits [10] will be masked; second, the affected value is proven to be a dead value.

Our analysis of voltage emergencies on SPEC CPU2000 benchmarks enables us to make the following key observation: only a few voltage emergencies, to be specific, about 32%, will affect program execution. The key observation motivates us to analyze which voltage emergencies affecting program execution, and further improve system reliability with less overhead. Based on the analysis, we propose a metric IVF_{itf} to characterize the masking effect of voltage emergencies and present a scheme to tolerate voltage emergencies.

III. Voltage Emergency Analysis

In this section, we first build a fault model for voltage emergency induced timing violations, and then present an algorithm for IVF_{itf} computing.

A. Intermittent Timing Fault Model

Voltage emergencies occur abruptly and last for a period of time. They usually lead to timing violations and these timing violations will not disappear until the supply voltage returns to steady-state voltage. Intermittent hardware faults also appear frequently and irregularly for a short while, commonly due to process, voltage, and temperature variations [11]. Intermittent faults can be categorized into three fault models: intermittent stuck-at fault model, intermittent open and short fault model, and intermittent timing fault model [12]. Intermittent timing faults affect data propagation and have similar effect like timing violations. Therefore, it is reasonable to use intermittent timing fault model to represent voltage emergency induced timing violations. An intermittent fault has three key parameters: burst length, active time, and inactive time. These three parameters determine the characteristics of an intermittent fault and can be changed for different fault mechanisms. We have proposed a metric IVF to estimate the vulnerability of microprocessor structures to intermittent stuck-at faults [12]. The IVF is computed through analyzing ACE bits and un-ACE bits in different structures. ACE bits are those if been changed will

affect the final program output while un-ACE bits are those if been changed have no adverse impact on program execution.

To characterize the architecture-level masking effect of voltage emergencies, we further propose a metric IVF_{itf} to compute the vulnerability of different microprocessor structures to intermittent timing faults. IVF_{itf} represents the percentage of voltage emergencies that will result in wrong program outputs. As voltage emergencies are closely related to the PDS and executing programs, we obtain the information of voltage emergencies through executing different SPEC2000 benchmarks, and then set burst length, active time, and inactive time for different intermittent timing faults. Burst length is the time between the first voltage emergency and the last voltage emergency in an specific interval. Active time is the duration time of a voltage emergency. Inactive time is the dead period between two activations within the same burst length. We compute IVF_{itf} for two microprocessor structures LSQ and REG, as they have relatively higher inductive noise rate and are more vulnerable to voltage emergencies in modern microprocessors [13].

B. IVF_{itf} Computation

Before present the algorithm to compute IVF_{itf}, we need to know when an intermittent timing fault will affect program execution. To determine the impact of an intermittent timing fault, two steps are needed: first, analyze whether the fault is captured by a storage cell; second, check whether ACE bits in the storage cell have been changed. Only when an intermittent timing fault propagates to storage cells and changes ACE bits, it will affect the final program output. Otherwise, the fault will not manifest itself in external output and is said to be masked. We use an example to further explain for this. Fig. 2 illustrates a timing violation leads to capture a wrong data to a storage cell. In this figure, $D_{correct}$ represents the correct data, and D_{wrong} represents the data affected by an intermittent timing fault. If there is no timing violation, the propagation delay for $D_{correct}$ and D_{wrong} will be the same. If an intermittent timing fault occurs, the data propagation in D_{wrong} will be affected. As can be seen, an intermittent timing fault occurs at Cycle 2 and lasts for several cycles until ending at Cycle *N*. Due to the timing violation, D_{wrong} propagates much slower than $D_{correct}$, which leads to excessive delay during program execution. Due to the accumulative delay, a wrong data will be captured at Cycle *N-1*, which means the intermittent timing fault has propagated to a storage cell.

We need to further analyze whether ACE bits in that cell have been changed by the fault. If ACE bits are upset, the fault will affect the external visible output. Otherwise, it is said to be masked at architecture level. There are mainly two scenarios that an intermittent timing fault will be masked during program execution: first, the data in a storage structure is proved to be a dead value; second, the captured data only changes un-ACE bits. If an intermittent timing fault is in either of these two scenarios, it will not affect program execution. Which scenario occurs is determined by analyzing ACE bits and un-ACE bits in different structures. For example, if the result of a dead instruction [14] is changed by an intermittent timing fault, even if an incorrect data has been written to REG, the fault will not affect program execution.

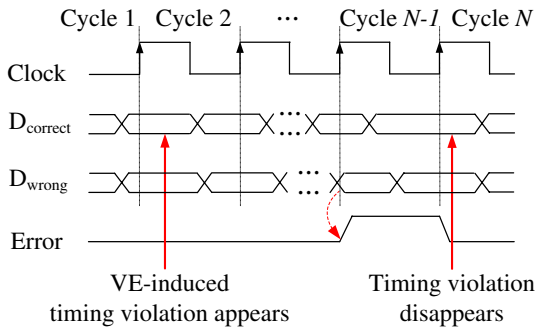

Fig. 2. A timing violation results in writing a wrong data to a storage cell.

As LSQ is used to buffer and maintain all in-flight memory instructions in program order, we analyze ACE bits in it by monitoring instructions when these instructions go through all stages of the pipeline. Meanwhile, REG is used to store and provide operation data for in-flight instructions, we analyze ACE bits in it based on its related operations, such as read, write and evict [12]. Only those faults propagating to storage cells and changing ACE bits contribute to IVF_{itf} computing. Based on the above analysis, the equation to compute IVF_{itf} for a structure can be expressed as:

$$IVF_{itf} = \frac{P_{num} - (N_{dead} + N_{un-ACE})}{NUM_{total}} \quad (1)$$

where NUM_{total} represents the total number of intermittent timing faults during executing a program; P_{num} represents the number of intermittent timing faults propagating to the structure; N_{dead} represents the number of faults only affecting dead values; N_{un-ACE} represents the number of faults only changing un-ACE bits. If N_{dead} and N_{un-ACE} are set to zero, it is the upper bound of IVF_{itf}. With this equation, we can compute IVF_{itf} for different structures. Though we only consider two structures in this work, without loss of generality, our method can be easily extended to other storage structures, such as issue queue and reorder buffer.

IV. Substantial-impact-filter Based Method

In the above section, we introduce the algorithm to compute IVF_{itf} for LSQ and REG. With the help of IVF_{itf}, we can get the masking information of different microprocessor structures to intermittent timing faults and guide reliability design. To tolerate voltage emergencies, several methods can be utilized: first is to activate a protection scheme once a voltage emergency occurs, namely the once-occur-then-rollback approach; second is to activate a protection scheme only when a voltage emergency affects final program execution, namely the ideal method. For the ideal method, it is necessary to analyze the number of N_{dead} and N_{un-ACE}. As dead values and un-ACE bits in a structure are mainly determined by the characteristics of a program, it is not possible to predict them before program execution. Besides, the time to determine a dead value or un-ACE bits usually takes about hundreds of

cycles [10], therefore, the performance overhead is unacceptable. To reduce the overhead, we set N_{dead} and N_{un-ACE} to zero and propose a substantial-impact-filter based method to tolerate voltage emergencies when architecture states are affected. Our design is a tradeoff between the once-occur-then-rollback approach and the ideal method. Another possible solution is to reduce the total number of voltage emergencies (reduce NUM_{total}) that occur during program execution. This solution, however, is orthogonal to our method and has not been considered in this work. Next we introduce our proposed method in detail.

A. Structure of Substantial-impact-filter Based Method

The key idea of our substantial-impact-filter based method is to differentiate these voltage emergencies which have impact on program execution. Fig. 3 illustrates the block diagram of our proposed method. In this design, a circuit-level delay sensor and a substantial-impact-filter node are combined for each structure we analyzed. The delay sensor is used to detect timing violations while the substantial-impact-filter node is used to determine whether a fault affects architecture states. The output of each filter will be used to trigger a program rollback.

Delay sensors are widely used and serve as canary circuits [15]. The measured resolution of a delay sensor can easily reach 5ps at 90nm technology [16], which is enough for us to detect the induced timing violations in microprocessors. The key parameter of a delay sensor is the timing threshold. If the timing threshold is set too pessimistic (tight), many false voltage emergencies will be detected and lead to unnecessary rollbacks. If the timing threshold is set too optimistic (loose), substantial voltage emergencies will be missed. Based on the alpha-power model [21], we compute the timing threshold when a voltage emergency is about to occur and set it 2.5% longer than the normal delay. When a delay exceeds the timing threshold, a timing violation occurs.

Fig. 4 further shows the concept of a substantial-impact-filter node. It contains a pair of D flip-flops (a master D flip-flop and a slave D flip-flop). The number of D flip-flop pairs is equal to the number of write ports of the structure under analysis. They are used to deal with the situation when multiple data are written to a structure at the same time. The master flip-flop is controlled by normal *clk* and the slave flip-flop is controlled by *clk_delay*. *clk_delay* is generated by an added circuitry and can be tuned for different microprocessors. The values of flip-flop pairs will be initialized to ZERO by setting signal *reset* to TRUE. When a delay sensor detects a timing violation, signal *timing violation* will be TRUE. Once a write signal (e.g. *w_*e_1) comes, the *enable* (abbreviated as *E* in Fig. 4) signal of D flip-flop pairs is TRUE and the substantial-impact-filter node will be triggered. During the lifetime of an intermittent timing fault, the data captured in the master D flip-flop and the slave D flip-flop will be compared. If they are equal, then the intermittent timing fault has not propagated to the structure. Otherwise, a wrong data has been captured. The comparison results from different flip-flop pairs will make an OR operation. If the output of the OR gate is TRUE, the program will roll back to ensure system correctness.

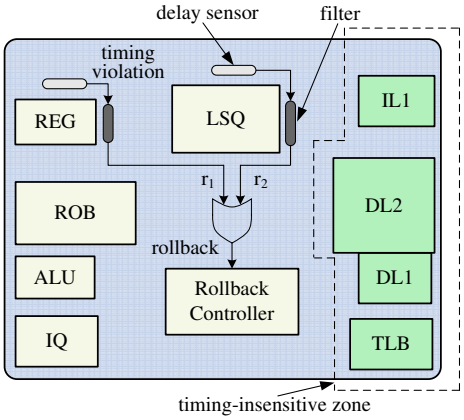

Fig. 3. Block diagram of the substantial-impact-filter based method.

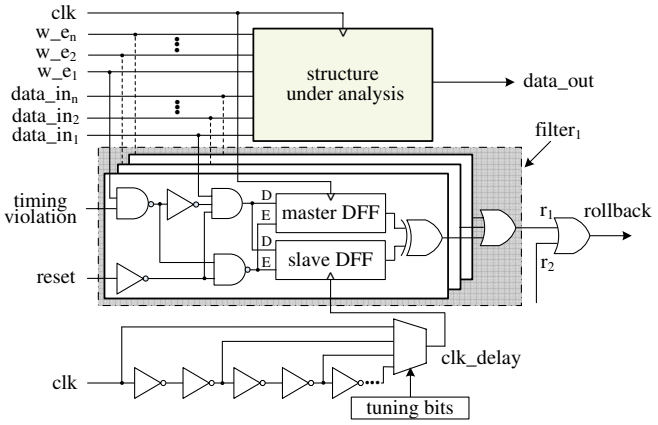

Fig. 4. Schematic diagram of the substantial-impact-filter.

Meanwhile, the values in these D flip-flop pairs will be reset to ZERO. Signal r_1 and r_2 represent the comparator results from two filters. With the proposed method, we can effectively avoid the rollbacks to these voltage emergencies which have no adverse effects on program execution. In order to avoid the reoccurrence of voltage emergencies during the rollback stage, the microprocessor will execute at a slower frequency for a short interval, such as at half of the normal frequency.

B. Performance and Area Overhead Analysis

We further analyze the performance and area overhead of our proposed method. In this design, as the added delay sensors and filters are not in critical paths, they will not affect system performance. The performance penalty of our method mainly comes from rollbacks and succeeding recovery executions. A program needs to roll back and rerun when a voltage emergency is proved to affect program execution. As many voltage emergencies will be masked, the number of rollbacks can be significantly reduced. Experimental results in Section VI demonstrate that our proposed method is cost-effective.

The area overhead of our method is mainly incurred by the added delay sensors and substantial-impact-filter circuits for each structure under analysis. After synthesized with Synopsis Design Compiler, the netlist of our added circuits only contains

about 5,000 logic gates. Comparing with a microprocessor having hundreds of millions logic gates, the additional area overhead by these extra hardware is rather negligible.

V. Evaluation Methodology

We use a cycle-accurate execution-driven simulator SimpleScalar 3.0d [17] to evaluate our IVF_{itf} computation and substantial-impact-filter based method. Table 1 lists the configuration parameters used to initialize the simulator for our baseline microprocessor design. Wattch [18] is combined to model the power consumption at the structure level. To model a PDS, we utilize Matlab to implement a second order linear model based on the characteristics of the Pentium 4 package [19] which is also used by prior works [4], [6]. In this model, we assume a voltage emergency occurs when noise-margin violation is beyond 5% of a 1V supply voltage. The cycle-level current is computed through dividing the power consumption to the assumed supply voltage. With the cycle-level current and impulse response of the linear model, the voltage is a convolution summation of the cycle-level current and the impulse response of the circuit model. We choose 16 SPEC CPU2000 benchmarks (10 INT, 6 FP) to evaluate our method. All the benchmarks are compiled for the Alpha ISA. In order to reduce simulation time, we use Simpoint tool [20] to pick the most representative simulation point for each benchmark and each benchmark is fast-forwarded to its representative point before detailed performance simulation takes place. Each benchmark is evaluated for 100 million instructions using the full reference input set.

Besides, the delay of a structure should be computed when an intermittent timing fault occurs. The delay of a gate (T_{delay}) is mainly determined by the supply voltage (V_{dd}), the threshold voltage (V_{th}), and the effective channel length (L_{eff}). The variation of these parameters will directly affect T_{delay}, as is expressed by the alpha-power model [21]:

$$T_{delay} \propto \frac{L_{eff} \times V_{dd}}{\mu \times (V_{dd} - V_{th})^{\mu}} \tag{2}$$

$$V_{th} \propto V_{th0} + k_1 \times (T - T_0), \mu \propto T^{-1.5} \tag{3}$$

where μ is the carrier mobility, and α is typically 1.3. Both μ and V_{th} are associated to the temperature (T) of a structure. We compute T_{delay} for different structures considering the variation of V_{dd} with this model. Besides, L_{eff} and V_t vary within-die due to process variation, and T varies across different structures due to temperature variation. As these two variations are not considered in this work, we use constant V_{th} and L_{eff} for each structure generated by VARIUS model [22], and a constant T (80C) in our following experiments.

VI. Experimental Results

In this section, we first present IVF_{itf} for two microprocessor structures, and then describe the performance overhead of our substantial-impact-filter based method.

Fig. 5 shows the upper bound of IVF_{itf} for LSQ and REG across different benchmarks. We assume the values of N_{dead} and N_{un-ACE} are zero and all these timing violations affecting architecture states have been considered. The average values for these two structures are 16.6% and 36.4%, respectively. We can see that the number of timing violations propagating to REG is much higher than that propagating to LSQ. For LSQ, the related write operations occur when the result is written into it by store instructions, and the result will not be written to cache until the commit stage; while for REG, the related write operations take place when an instruction commits or when a value is loaded from memory. The number of instructions writing to REG is much more than the number of memory access instructions.

TABLE I
Simulated microprocessor configuration

Parameters	Configuration
Clock Frequency	3.0 GHz
Fetch/Decode Width	8 instructions/cycle
Branch-Predictor Type	64 KB bimodal gshare/chooser, 1K entries
Reorder Buffer Size	128
Unified Load/Store Queue Size	64
Physical Register File	32-entry INT, 32-entry FP
INT ALU, INT Mul/Div, FP ALU, FP Mul/Div	8/2/4/2
L1 Data Cache	64KB, 2-way, 32B line-size, 1-cycle latency
L1 Instruction Cache	64KB, 2-way, 32B line-size, 1-cycle latency
L2 Unified Cache	2MB, 4-way, 64B line-size, 16-cycle latency
I-TLB/D-TLB	128-entry, fully-associative

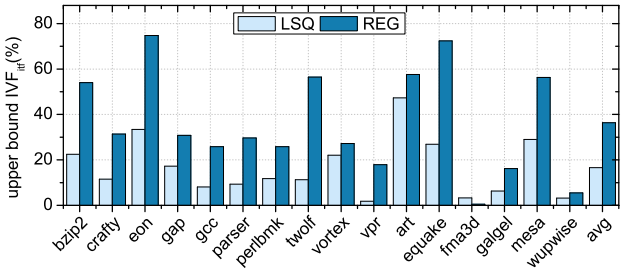

Fig. 5. Upper bound of IVF_{itf} for LSQ and REG.

Fig. 6 shows the IVF_{itf} results obtained by the computation method presented in Section III.B for LSQ and REG. The average IVF_{itf} for these two structures are 14.8% and 31.7%, respectively. The IVF_{itf} of REG is also higher than that in LSQ. Compared with the upper bound value, the IVF_{itf} reduction of these two structures are about 1.8% and 4.7%, respectively. The reduction is due to these voltage emergencies which only affect dead values and un-ACE bits are excluded during computation. When executing the benchmark *fam3d*, voltage emergencies occur very rare, the number of which affecting program execution is also very small.

Fig. 7 illustrates the performance overhead of the once-occur-then-rollback approach, a prior proposed delayed-commit and rollback (DeCoR) mechanism [5], and our substantial-impact-filter based method. We use a system without voltage emergencies tolerance as a baseline. As checkpoints can be taken at different intervals (e.g. from 50 to 1000 cycles), we

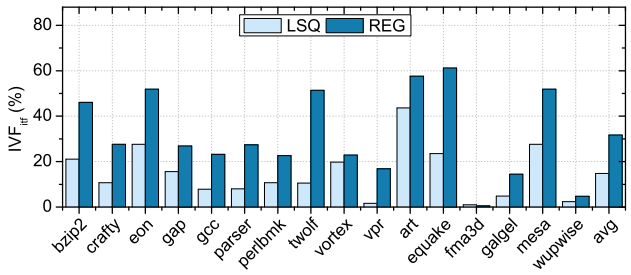

Fig. 6. IVF_{itf} of LSQ and REG.

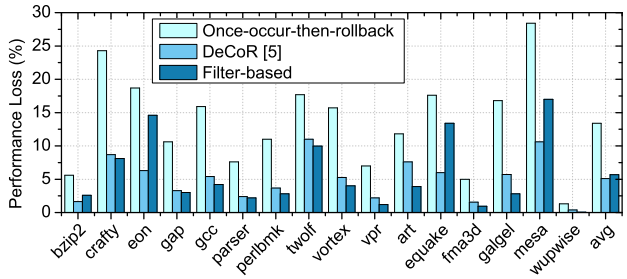

Fig. 7. Performance loss of the once-occur-then-rollback approach, DeCoR mechanism [5], and our substantial-impact-filter based method.

assume a 100-cycle rollback penalty for each recovery. As can be seen, comparing with the baseline system, the average performance overhead for these three methods are 13%, 5.1%, and 5.6%, respectively. Our substantial-impact-filter based method can gain back about 57% performance loss from the once-occur-then-rollback approach as many rollbacks are avoided. Besides, for most benchmarks, our method has less overhead than DeCoR. There are two reasons for this. First, our method exploits the architecture-level masking of voltage emergencies and reduces the cost of recovery; Second, DeCoR delays the commit to the microprocessor state and needs to rollback for all voltage emergencies. We can also observe a notable exception that our method has much higher performance loss than DeCoR for some benchmarks (such as *eon* and *equake*). The reason is that the percentage of voltage emergencies to be masked is very small and the performance loss due to rollbacks increases.

Our proposed substantial-impact-filter based method focuses on these voltage emergencies that will change architecture states. In this work, voltage emergencies only affecting dead values or un-ACE bits have not been considered, which leaves the optimization space. How to reduce performance overhead when tolerating the two kinds of voltage emergencies is left for our future work. Ernst et al. [23] also propose a similar scheme named "Razor" to detect and correct path delay failures. "Razor" aims to design low power pipeline through dynamic voltage tuning, but our method aims to tolerate voltage emergencies and reduces performance overhead, which is the main difference between these two methods.

VII. Conclusions

We have analyzed the characteristics of voltage emergencies and categorized the induced timing violations as intermittent timing faults. We computed IVF_{itf} for two microprocessor structures (LSQ and REG). With the guide of IVF_{itf}, we proposed a substantial-impact-filter based method to tolerate voltage emergencies. Our experimental results show the averaged IVF_{itf} for LSQ and REG are 14.8% and 31.7%, respectively. Besides, our proposed method can guarantee system reliability while gains back nearly 57% of performance loss compared with the once-occur-then-rollback approach.

References

[1] J. W. McPherson. "Reliability challenges for 45nm and beyond," In *DAC*, 2006.

[2] N. James, P. Restle, J. Friedrich, B. Huott, and B. McCredie. "Comparison of Split-Versus Connected-Core Supplies in the POWER6TM Microprocessor," In *ISSCC*, 2007.

[3] M. K. Gowan, L. L. Biro, and D. B. Jackson. "Power Considerations in the Design of the Alpha 21264 Microprocessor," In *DAC*, 1998.

[4] E. Grochowski, D. Ayers, and V. Tiwari. "Microarchitectural Simulation and Control of di/dt-induced Power Supply Voltage Variation," In *HPCA*, 2002.

[5] 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*, 2008.

[6] R. Joseph, D. Brooks, and M. Martonosi. "Control Techniques to Eliminate Voltage Emergencies in High Performance Processors," In *HPCA*, 2003.

[7] 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*, 2007.

[8] 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.

[9] V. J. Reddi, M. S. Gupta, G. Holloway, G. Y. Wei, M. D. Smith, and D. Brooks. "Voltage Emergency Prediction: Using Signatures to Reduce Operation Margins," In *HPCA*, 2009.

[10] S. S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin. "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," In *MICRO*, 2003.

[11] P. M. Wells, K. Chakraborty, and G. Sohi. "Adapting to Intermittent Faults in Multicore Systems," In *ASPLOS*, 2008.

[12] S. Pan, Y. Hu, and X. Li, "IVF: Characterizing the Vulnerability of Microprocessor Structures to Intermittent Faults," In *DATE*, 2010.

[13] 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.

[14] B. Fahs, S. Bose, M. Crum, B. Slechta, F. Spadini, T. Tung, S. Patel, and S. Lumetta. "Performance Characterization of a Hardware Mechanism for Dynamic Optimization," In *MICRO*, 2001.

[15] R. B. Staszewski, S. Vemulapalli, P. Vallur, J. Wallberg, and P. T. Balsara, "1.3 V 20 ps time-to-digital converter for frequency synthesis in 90-nm CMOS," *IEEE Trans. on Circuits and Systems II*, 2006.

[16] S. Henzler, S. Koeppe, W. Kamp, H. Mulatz, D. Schmitt-Landsiedel. "90nm 4.7ps-Resolution 0.7-LSB Single-Shot Precision and 19pJ-per-Shot Local Passive Interpolation Time-to-Digital Converter with On-Chip Characterization," In *ISSCC*, 2008.

[17] D. Burger and T. M. Austin. "The SimpleScalar Tool Set, Version 2.0," Computer Architecture News, pp. 13-25, June 1997.

[18] D. Brooks, V. Tiwari, and M. Martonosi. "Wattch: a Framework for Architectural-level Power Analysis and Optimizations," In *ISCA*, 2000.

[19] K. Aygun, M. Hill, K. Eilert, R. Radakrishnan, and A. Levin. "Power Delivery for High-Performance Microprocessors," Intel Technology Journal, 9(4), Nov. 2005.

[20] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. "Automatically Characterizing Large Scale Program Behavior," In *ASPLOS*, 2002.

[21] T. Sakurai and R. Newton. "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *Journal of Solid-State Circuits*, 1990.

[22] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *IEEE TSM*, Feb. 2008.

[23] D. Ernst, N. Kim, S. Das, S. Pant, R. Rao, T. Pham, T. Austin, et al. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," In *MICRO*, 2003.