Pipeline Damping: A Microarchitectural Technique to Reduce Inductive Noise in Supply Voltage

Michael D. Powell and T. N. Vijaykumar
School of Electrical and Computer Engineering, Purdue University
{mdpowell, vijay}@ecn.purdue.edu

Abstract

Scaling of CMOS technology causes the power supply voltages to fall and supply currents to rise at the same time as operating speeds are increasing. Falling supply voltages cause noise margins to decrease, while increasing current and frequency makes supply noise injection larger, especially noise caused by inductance in the supply lines. Creating power distribution systems is one of the key challenges in modern chip design. Decoupling capacitance helps reduce inductance effects, but there is often a peak in the supply impedance that occurs at a resonant frequency caused roughly by the package inductance and the chip decoupling capacitors. This frequency is on the order of 100MHz, which is much lower than the operating frequency of the processor. We propose pipeline damping, an architectural technique which controls instruction issue to guarantee bounds on current variation around the frequency of the supply resonance, thus reducing the resulting supply noise. Damping is a cheaper alternative to expensive, circuit-based noise-reduction techniques. We make the fundamental observation that limiting the current flow change (di) within resonant time period (dt) controls di/dt without large performance loss. Damping guarantees bounds on current variation while allowing processor current to increase or decrease to the magnitude required to maintain performance. Our results show that a damped processor guarantees a 33% reduction in the worst-case current variation with an average performance degradation of 7% and average energy delay of 1.09 compared to an undamped processor.

1 Introduction

The downscaling of feature sizes in CMOS technologies is resulting in faster transistors and lower supply voltages. While this trend enables high overall performance and low per-transistor power, an unwanted side-effect is reduced noise margin. Furthermore, because total chip power is not decreasing, the total chip current is growing. The increasing current makes the design of the power distribution system for these chips difficult, because changes in chip current must cause only small changes in the supply voltage (i.e., supply voltage noise). Low-power techniques, such as clock gating, exacerbate supply noise because gating components on and off causes large changes in chip current. While the noise problem originates at the power supply and contributes to degraded logic signal integrity, this paper targets supply voltage noise and not logic signal noise.

To prevent current changes over a wide range of frequencies (from kHz up to the clock frequency) from becoming voltage spikes, designers create the power supply such that it has a low impedance over a wide frequency range. To create a low-impedance power supply, circuit designers use a hierarchy of decoupling capacitors and voltage regulators. Typically systems use on-die capacitors, on-package capacitors and voltage regulators, and off-package capacitors and regulators. The decoupling capacitors compensate for impedance introduced by the parasitic inductance of the power supply network at each level of the hierarchy. However, it is not easy to compensate for the inductance of the wires between the die and the package. This inductance often causes a peak of high impedance [8, 1] in the supply at the resonance of the chip capacitance and the package inductance. Noise at this *resonant frequency*, which is in the range of 10-100 MHz [1, 6], is the most dangerous and can cause reliability problems [2]. Circuit techniques for compensating for this exposed inductance, such as increased on-die capacitors [5] and on-die voltage regulators [7], are expensive.

Not all current variations cause problems: inductive noise occurs when the processor current variation *matches* the resonant frequency. The key reason for processor current variation is the uneven nature of instruction level parallelism (ILP) across program phases. In this paper, we focus on microarchitectural solutions to reducing current variation at the resonant frequency. While circuit-level solutions attempt to *cure* current variations, we *prevent* the variations at the source. We propose *pipeline damping* to limit the *rate of change* of processor current occurring at the resonant frequency by controlling instruction issue.

Unlike energy reduction schemes, which reduce the average *magnitude* of current, pipeline damping bounds the *rate of change* of current. Pipeline damping *guarantees* a *worst-case* bound on the di/dt (as opposed to reducing the *average*), which is required for circuit designers to avoid expensive solutions.

An alternative approach to control di/dt is to limit the peak current (max i) which bounds the maximum current flow change (max di). Unfortunately, throttling the peak current is equivalent to limiting the exploitable ILP, and results in substantial performance loss. We make the fundamental observation that limiting the current flow change (di) over a *window* of consecutive cycles (dt), which corresponds to the resonant time period, to a pre-specified Δ bounds di/dt without considerable performance loss. Instead of inflexibly restricting the peak, limiting the change allows the current to vary, in controlled steps of Δ, to the magnitude required to exploit the available ILP.

The main results of this paper are:

- For a resonant frequency 1/50th of the processor clock frequency, one pipeline damping configuration guarantees a 33% reduction in worst-case current variation. This result can be put in perspective by comparing to the circuit-based technique in [7] which reduces variation about 40%.
- Pipeline damping prevents processor current from increasing faster than a given bound by delaying instruction issue, trading-off performance. Damping prevents current from decreasing faster than the bound by activating otherwise-unused resources, trading-off energy. For the damped processor achieving 33% reduction in worst-case variation, average performance degradation is 7%, and average energy-delay is 1.09, relative to an undamped processor.
- Pipeline damping outperforms an inductive noise controller that limits peak current. To achieve a 33% reduction in worst-case variation, peak-current limitation incurs an average performance degradation of 55%, whereas damping incurs only 7% degradation.

In Section 2 we discuss resonant frequencies and inductive noise. Section 3 Explains pipeline damping. Section 4 describes our methodology and Section 5 presents our results. We discuss related work in Section 6 and conclude in Section 7.

2 Resonance and Inductive Noise

As discussed in Section 1, this paper targets inductive power supply noise around the resonant frequency of the power supply network where current variation causes the largest voltage noise. Therefore, we wish to prevent the current from varying at the specific resonant frequency identified by design-time CAD tools.

Microprocessor current varies due to changes in instruction level parallelism (ILP) throughout programs. ILP is not uniform throughout program execution. The medium-term ILP of a program varies substantially from the average ILP. ILP is reduced for various time periods due to cache misses, long-latency instructions, and data dependencies. Spurts of high ILP are therefore necessary to maintain performance.

Unfortunately, the changes in ILP causes variation in resource utilization which in turn cause spikes in processor current. Consequently, we wish to prevent ILP variation from occurring at the microprocessor circuits' resonant frequency. If the program causes current changes to occur at the resonant frequency, correspondingly large changes in supply voltage will result in high supply noise. An example of a program that would cause such current changes is a loop with iterations as long as the period of the resonant frequency. If the loop iterations have high ILP (high current) for their first half and low ILP (low current) for their second half, current would vary at the resonant frequency [6].

3 Pipeline Damping

In the previous section, we discussed the relationship between current variation at a circuit's resonant frequency and supply noise. In this section, we introduce pipeline damping, an architectural technique to prevent current variations at a resonant frequency. From this point, we will discus the period (time) of resonant frequencies rather than the frequency (rate) to simplify the explanations.

Recall that dealing with supply noise requires guaranteeing a worst-case bound on the di/dt, as opposed to reducing the average di/dt. This guarantee is needed by circuit designers to avoid expensive solutions.

One approach to limiting current variation (di/dt) is to limit the peak current per cycle (max i) which bounds the maximum current flow change (max di) over *any* amount of time. Unfortunately, throttling the peak current is equivalent to limiting the exploitable ILP and results in substantial performance loss. Furthermore, such a solution is overkill because it reduces di/dt over *all* time periods instead of focusing on the processor's resonant period. As we saw in Section 2, preventing current change over non-resonant periods is not crucial to supply noise reduction.

The concept of peak-current limitation is illustrated on the left side of Figure 1 for a program profile with current changing at the resonant period (T). The original current profile shown is the worst-case because of the high current value for the first half of the resonant period followed by the low current value for the second half, forming a wave with the resonant period. For the example we set the maximum allowed variation over the resonance period to be a wave with peak-to-peak magnitude M. To prevent the current from varying at peak-to-peak magnitude of 2M at the resonant period, peak-current limitation simply caps the

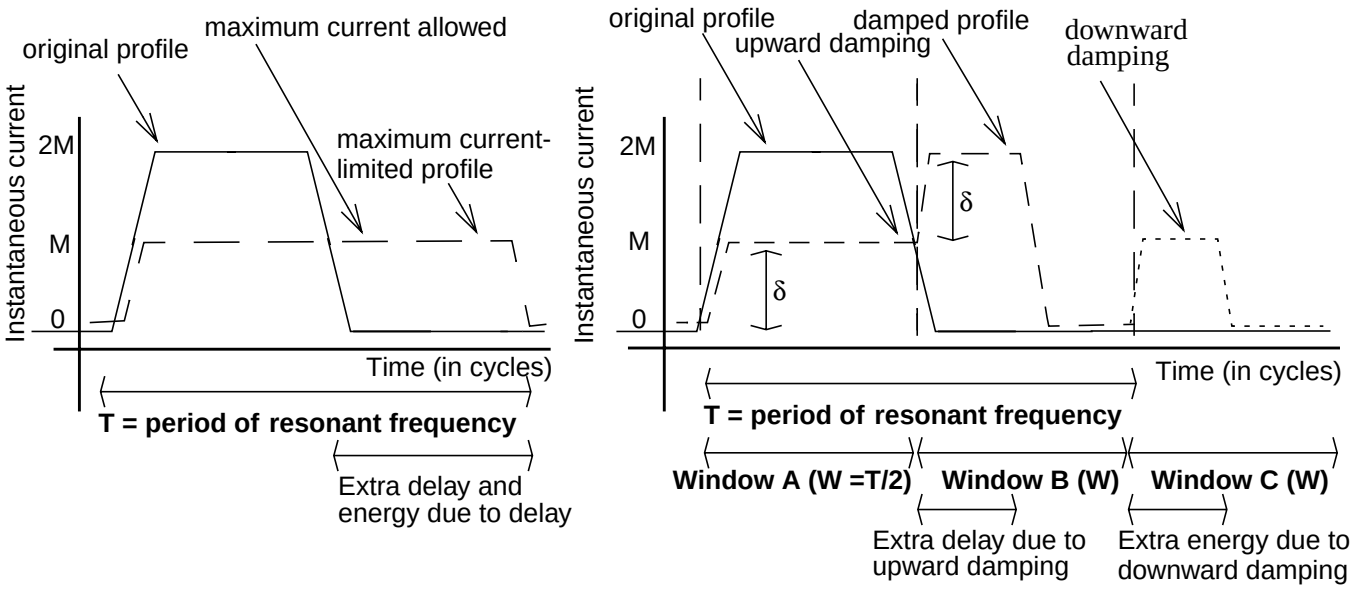

FIGURE 1: Pipeline damping to control worst-case current variation at resonant frequency.

maximum current at M. Limiting the peak current delays execution of many instructions compared to the original profile and results in T/2 additional delay.

3.1 Concept

We make the fundamental observation that limiting the *change in "total current"*—i.e., sum of all of the instantaneous currents in a "window of cycles"—between consecutive windows to a pre-specified *delta* (Δ) controls di/dt at the time period of 2 * window. We set the window to be *half* of the resonant period (W = T/2) because we wish to prevent large upward and downward changes making up the halves of a wave with the resonant period. Instead of restricting *peak* current, limiting the current flow *change* allows current to increase or decrease, in controlled steps of Δ, to the magnitude required to exploit the available ILP, without considerable performance loss.

The right side of Figure 1 illustrates damping. For the original profile in the figure, the total current flow before window A is 0, the total current during window A is MT, and the total current during window B is 0. For the profile in Figure 1 that illustrated peak-current limiting, the maximum current change allowed between windows corresponds to Δ of MT/2. Clearly, this Δ constraint is met neither between the time before window A and window A nor between windows A and B.

Pipeline damping ensures the Δ constraint is met by establishing a relationship between the sums of the current in consecutive windows. Using windows A and B from the figure as examples with I representing the total current of the window and i_n representing the current in the n^{th} individual cycle, we express the current change between consecutive windows as:

$$I_B - I_A = \sum_{n=W+1}^{2W} i_n - \sum_{k=1}^{W} i_k = \sum_{n=W+1}^{2W} (i_n - i_{n-W})$$

We wish to constrain $|I_B - I_A|$ to Δ. To do so, we constrain $|i_n - i_{n-W}|$, which is the maximum change in current allowed between cycles that are W cycles apart, to δ. Therefore:

$$\sum_{n=W+1}^{2W} |(i_n - i_{n-W})| \le \sum_{n=W+1}^{2W} \delta$$

By the triangular inequality:

$$|I_B - I_A| = \left| \sum_{n=W+1}^{2W} (i_n - i_{n-W}) \right| \le \sum_{n=W+1}^{2W} |(i_n - i_{n-W})|$$

which gives:

$$|I_B - I_A| \le \delta W$$

Therefore, by setting δ = Δ/W, we can constrain $|I_B - I_A|$ to Δ. Consequently, pipeline damping is implemented by constraining the current difference between cycles W cycles apart to be less than or equal to δ = Δ/W. In our example, δ equals M and Δ = MW. (Observe the difference between little-delta and big-delta (δ and Δ) as both will be used extensively.)

It is extremely important to note that to damp variation at the resonant period, the Δ constraint must be met for *all possible pairs* of consecutive W-cycle windows, *regardless* of where the windows start in the timeline. Otherwise, supply noise will occur simply time-shifted with respect to the Δ–constrained windows. Examples of other window pairs in Figure 1 include the windows starting from the midpoints of window A and B (referred to as midpoint-A and midpoint-B). Because the δ constraint is met for *all* pairs of cycles W

cycles apart, the summations for Δ at the beginning of this subsection hold for *all* adjacent pairs of windows.

Looking at the damped profile (medium dashes) in Figure 1, we see that *upward damping* prevents the current from increasing to more than δ during window A because the individual cycle currents in the previous window (before time = 0, not entirely shown) were 0. The current is allowed to increase to 2M in window B because 2M is within δ of the current from W cycles back. Postponing the current expenditure from A to B delays execution of many instructions compared to the original profile. The total delay, with window A using M current and the first half of window B using 2M current, is T/4 over that of the original profile, compared to the T/2 additional delay for peak-current limiting.

Upward damping is only half of the requirement. We also wish to prevent large downward changes corresponding to half of a wave with the resonant period, such as the drop in the original current profile in Figure 1. Looking at the dotted profile in window C we see an extra current "bump" that corresponds to *downward damping*, which prevents the total current from decreasing more than Δ between the midpoint-A and midpoint-B windows mentioned above. With the help of the bump, the total current for the midpoint-B window is within Δ of the total current for the midpoint-A window, so the downward damping constraint is met. The bump exists soley for the pupose of meeting the Δ constraint, and therefore represents extra energy consumption for the processor.

Two observations provide an illustration of how using δ facilitates meeting the Δ constraint to reduce di/dt at the resonant period: (1) Because the current for the first half of window B is 2M, the δ constraint requires that the current increase to M only for the first half of window C. Placing the bump in the second half of window B would meet the Δ constraint between the midpoint-A and midpoint-B windows but would violate the δ constraint and still require placing an additional bump at the beginning of window C to meet the Δ constraint beetween windows B and C. (2) The drop from 2M to zero current halfway through window B *does not* violate the Δ constraint because the drop does not occur *across* adjacent windows. The drop occurs *within* a window and is not at the resonant frequency. This drop, which already exists in the original profile at the end of window A, is high-frequency di/dt that is handled by circuit techniques discussed in Section 6.

It might seem that employing the triangular inequality is conservative and may result in weak Δ constraint. In practice, we found that we achieve 33% worst-case di/dt reduction at only 7% average performance degradation.

3.2 Implementation

A real implementation requires that Ldi/dt, expressed as

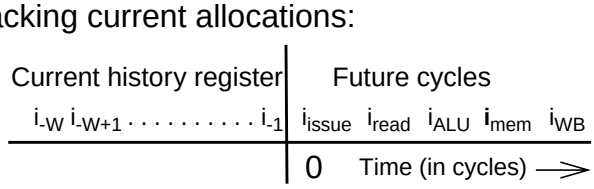

FIGURE 2: Using per-cycle current allocations to control δ for entire back-end at the issue stage.

LΔ/W, is within the noise margin of the circuit. Based on the values for the noise margin and L from circuit analysis, δ (= Δ/W) is chosen to meet the noise-margin constraint.

Implementing pipeline damping in a modern out-of-order processor requires controlling current variation to meet the δ constraint. In this subsection, we describe the sources of current variation in a microprocessor and then discuss how to control current variation by scheduling current. Pipeline damping schedules current in the same way that conventional schedulers schedule resources such as cache ports and functional units.

3.2.1 Back-End

In this discussion, we separate the pipeline into front-end and back-end, and we start with the back-end. The key to variability in the back-end is the issue stage. The back end exhibits a great deal of variability corresponding to both program phases and data dependencies, manifesting as variations in the number and type of instructions issued each cycle. The issue stage itself is a source of substantial current variation, but the effects of the issue stage ripple through the remainder of the pipeline: register read, functional units, cache access, and register writeback.

The are two key implementation concerns for damping in the back-end. First, because an instruction's current is not instantaneous and occurs over several cycles as the instruction moves through the back-end, damping must account for the current in each cycle. The δ constraint establishes a *current allocation* for each cycle that establishes how much current may be drawn (and how much current must be drawn to meet the downward damping constraint). Before issuing an instruction, damping ensures the δ constraint will not be violated for each cycle by counting the currents. Each affected cycle must be evaluated because we wish to avoid satisfying the current allocation for the present cycle while creating a violation by allocating current above or below that of the constraint in a future cycle.

The second concern is that damping constraints must be met before an instruction issues and begins consuming current, not after issue and immediately before a δ constraint

violation. It is key that damping ensure before issue that an instruction will meet all relevant δ constraints. Instructions cannot be arbitrarily stalled after issue to prevent a δ constraint violation because this would require freezing all successive instructions in the back-end. Such a stall of several pipeline stages would over-compensate and substantially reduce the current drawn (assuming an energy-efficient processor using some clock-gating), possibly violating the minimum current required for the cycle(s) by the δ constraint. In the act of preventing an upward violation, a much larger downward violation may occur. Damping avoids this problem by proactively counting all of the δ constraints at the issue stage instead of attempting to react to δ constraint violations that are about to occur throughout the back-end.

Conventional select logic already counts resources to determine if an instruction is eligible for issue. The logic must count the number of instructions to ensure the issue width is not exceeded, and it must count the number of available ALUs, floating point units, and cache ports to avoid conflicts over these resources. Select logic for pipeline damping also counts current bounds as an additional resource constraint. A key difference between counting resources and current is that processor resources exist in integral quantities (no fractions) but current magnitude is a floating-point quantity. Handling non-integral quantities at select is undesirable, so we simplify the counting process by approximating currents with small (4-bit) integers in the correct proportions. δ is then computed using the same integral units. (For example, a d-cache access might have twice the current of an ALU operation, so the d-cache would be assigned a current value of 2 and the ALU assigned a value of 1.) While pipeline damping does burden the select logic with a new constraint, we believe the benefits of addressing supply noise, a key reliability problem in microprocessors, are worth the complexity.

To track the counts for each cycle's current allocation, damping maintains a history register containing the current allocations for the next W cycle similar to the branch history register in the L1 of a two-level branch prediction. The allocations are based on the previous W cycles (W being the window size from the previous subsection) with any units of already-allocated current deducted. Figure 2 illustrates the decision-making process for a back-end where each architectural stage is one-cycle. The current allocation constraint for each of the four cycles with a component for the ALU instruction (issue, read, Ex, and WB; but not mem because the ALU instruction doesn't access the d-cache) must be met in order to issue the instruction.

Downward damping follows a similar procedure as upward damping but ensures that the present and future current values are not too low to meet the minimum current allocation. We implement downward damping by "issuing" extraneous integer ALU operations that fire-up the issue logic, register read ports, and an unused ALU (but do not activate result busses or write-back). The sole purpose of these extraneous operations is to draw current necessary to meet the δ constraint.

Not all operations are scheduled at issue, such as stores and predictor updates. However, the resources for these operations, such as cache ports, still must not conflict with instructions at issue. Conventional pipelines must handle contention for cache ports at select because loads and stores share the same resource. Similarly, damping requires that the current for stores and branch predictor updates be included in the current-allocations for the cycles in which they occur. The counts for these currents may be included in the select process for damping.

Pipeline damping faces two issues regarding d-cache misses: the current variability from the miss due to squashed instructions in the pipeline and the current of the corresponding L2 access. Load misses conventionally cause instructions that issued after the offending load to squash. Aggressive clock-gating may save energy by preventing the squashed instructions from propagating down the pipeline. Such clock gating could result in a large downward spike in processor current. Instead, to reduce supply noise, squashed instructions may be allowed to continue down the pipeline as extraneous, "fake," events, similar to downward damping. D-cache misses also initiate L2 accesses, which have a low per-cycle current because they are spread over many cycles. L2 accesses can be handled by deducting the appropriate values from the current allocations of the affected cycles. In some processors, the L2 may be included on a separate on-chip power grid and may be irrelevant to pipeline damping in the core.

3.2.2 Front-end

The front-end of the pipeline is fairly consistent in current drain and is not a key source of variability. The i-cache, accounting for about 10% of maximum processor current [3] is a large component of front-end current. Variability in the i-cache access rate corresponds to misses caused by changes in instruction working sets. Although front-end variability is irrelevant to back-end current because downward damping at issue compensates for any deficiency of instructions in the issue queue, requirements for a tight overall current variation constraint might require mitigating variability in the front end itself.

One simple solution for front-end variability is to activate all i-cache ports and all decode/rename logic every cycle. This "always-on" solution is a simplistic form of downward damping in that it never allows the current to drop. While the energy overhead of this solution may seem high (there is no performance overhead), that may not be the case in light of typically low i-cache miss-rates. If i-cache accesses occur in the vast majority of cycles in a conventional system, the additional energy overhead of firing up the front-end for the remaining cycles is small. For example, with i-cache

accesses occurring in 90% of cycles, the energy overhead would be 2.5% if the front-end accounts for 25% of processor energy.

If having an always-on front-end is undesirable, the front-end variability can be accounted for using the current allocation scheme described for the back-end in Section 3.2. Using damping, a fetch does not occur unless the current for the corresponding fetch, decode, and rename cycles fall within the δ constraints for those cycles. The process is the same as back-end damping with the control at fetch instead of at issue. Some coordination may be necessary between the front-end and back-end to ensure that fetches are not starved in favor allocating current to instruction issue, or vice versa.

3.3 Implementation Simplifications

In this section, we observe two potential simplifications for implementing pipeline damping. Our first observation is that not all components of the processor may need to be damped. The relevance of a particular component to pipeline damping depends on both variability in usage and the magnitude of the current. For example, if the i-cache were accessed every cycle, it would not be a source of current variability regardless of the magnitude of its current. Acceptable current bounds may be established without damping the current of some variable, but low-current, components. Excluding components from damping would extend the Δ equation as follows:

$$\Delta_{actual} = \delta W + W\sum i_{undamped}$$

where the $i_{undamped}$ terms are the maximum currents of components not included in pipeline damping. In this case, damping guarantees a looser Δ constraint.

Our second observation is that damping may be simplified if the δ constraint is applied over sub-windows of adjacent cycles. As clock frequencies become faster in future technologies, the number of cycles in the processor's resonant period may increase from tens of cycles to hundreds of cycles. For such long windows, it may be infeasible to maintain a history register containing the current allocation for each cycle in a window or compute the current allocations for each operation at issue. We can aggregate adjacent cycles into sub-windows and then construct damping windows from the sub-windows. An example for a window size of 500 cycles would be utilize 20-cycle sub-windows and then construct the 500 cycle window from 25 of the sub-windows. The δ constraint would then be applied to pairs of sub-windows separated by 25 sub-windows. This coarser-grained solution would have a somewhat looser Δ constraint because of uncertainty in the individual cycles at the window edges. However, in terms of the total current feasible over a window of hundreds of cycles, the slack introduced by uncertainty in a few tens of cycles might only slightly loosen the bound on di/dt over the full window.

A coarse-grained solution also could have a substantial advantage in simplifying the pipeline-damped scheduler. If the sub-window size is larger than the depth of the pipeline back-end, it may not be necessary to separately track the current allocations for each stage of the pipeline. An aggregate current allocation that included all of the back-end current could be used. Instead of counting current allocations for each affected cycle as described in Section 3.2.1, only a single lumped current count would be necessary to determine if an instruction may be issued.

3.4 Effect of inaccuracies in current estimation

Because pipeline damping is based on predetermined estimates of resource current, inaccuracies in the estimation are a concern. For example, an estimator may assume that all integer adds consume approximately the same current. Because high-performance circuits are implemented in dynamic logic and dynamic logic power is dominated by the clock, this assumption is not unreasonable. Though clock power is dominant, some variability will still occur due to differences in the inputs.

Even in the presence of estimation inaccuracies, it is possible to use pipeline damping to establish current variability bounds. If the current change between windows is estimated at Δ but actually may be x% higher or lower, then the actual maximum variability is an increase from the minimum current, (1 - x/100)Δ, to the maximum current (1 + x/100)Δ. The total worst case variability is then (1 + 2x/100)Δ. For example, if the actual current change between windows could be 20% higher or lower than Δ, then the actual current bound would be 1.4Δ instead of Δ.

By knowing in advance the maximum error in the current change estimate, a Δ that will lead to a suitable actual current bound may be chosen. While accounting for estimation accuracies may lead us to tighten Δ, in Section 5.1 we show that tightening Δ does not result in large performance or energy degradation. However, a fundamental limitation is that an x% error in current estimates implies that damping cannot bound current variation to a value less than x%. That is, Δ cannot be set to less than x% of the total current. Therefore, less error in the estimation is desirable.

4 Methodology

Table 1 shows the base configuration for the simulated system. We modify Wattch [3] and incorporate SimpleScalar 3.0b [4] modifications to simulate a high-performance, out-of-order microprocessor executing the Alpha ISA. To facilitate more accurate estimation of per-cycle energy, we modified Wattch to use energy-efficient L1 caches. To estimate the rate of change of current flow (di/dt) we extend Wattch to compute current for each cycle in addition to energy based on component activity. To enable calculation

Table 1: System parameters.

instruction issue	8, out-of-order
Issue queue/ROB	128 entries
L1 caches	64K 2-way, 2 cycle, 2 ports
L2 cache	2M 8-way, 12 cycles
Memory latency	80 cycles
Fetch	up to 8 instructions/cycle with 2 branch predictions per cycle
Int ALU & mult/div	8 & 2
FP ALU & mult/div	4 & 2

of per-cycle current, we spread the execution energy of multi-cycle functional units and pipeline events (e.g., register reads) over each of the relevant cycles. We calculate current by observing that current is proportional to power with a coefficient of (1/Voltage). To compute the actual di/dt in amperes/s, this current change would have to be divided by the cycle time. However, the average change in current over adjacent windows is linearly proportional to the actual di/dt, and allows us to abstract away clock speeds. Because we did not want to assume specific clock speeds, we measure di/dt as the average change over adjacent windows of cycles.

We use 23 of the 26 applications in the SPEC 2K benchmark suite (*ammp*, *mcf*, and *sixtrack* are excluded due to simulation time), fastforwarding 2 billion instructions to pass initialization code, and then running 500 million instructions. The base (undamped) IPC for each application are shown above the names in Figure 3.

To evaluate the effectiveness of pipeline damping we compare the *worst-case* di/dt that can occur at the resonant period in an undamped system to the *worst-case that is guaranteed not to be exceeded* in the damped system. This comparison corresponds to the worst-case nature of the inductive problem—ensuring correctness requires ensuring that di/dt never exceeds the guaranteed value. We compute the worst-case di/dt from Wattch's current values.

We also evaluate performance degradation due to upward damping and energy increase due to downward damping. To measure energy increase, we use the relative energy-delay metric common in low-power research. Because damping increases both execution time and energy, energy-delay relative to the undamped case will have values greater than one.

As discussed in Section 3.2.1, we approximate microprocessor current components by integral units (using 4-bit integers) to be used when counting current allocations. These integral values are used to compute current bounds and are based on the currents reported by Wattch. Table 2 shows the latencies and integral estimates of per-cycle current for each of the variable current components in our microprocessor. Each integral unit corresponds approximately to 0.5 A in a 2 GHz 1.9 V processor.

Table 2: Integral unit current estimates and latencies of variable components.

Component group/Item	**latency (cycles)**	**per-cycle current**
Front-end (fetch--rename)	N/A	10
Wakeup/Select	1	4
Register Read	1	1
Int. ALU	1	12
Int. Multiply	3	4
Int Divide	12	1
FP ALU	2	9
FP Mult	4	4
FP Divide	12	1
D-cache	2	7
D-TLB	1	2
LSQ Access	1	5
Result Bus	3	1
Register Write	1	1
Branch Pred., BTB, RAS	1	14

While we acknowledge that Wattch's models may have some error in their estimates, damping is tolerant of estimation inaccuracies, as discussed in Section 3.4. Though some of the integral estimates, such as those of the ALUs, may seem high, we note that overestimating current is a conservative choice for our simulations, because pipeline damping will experience greater performance and energy degradation by damping overestimated component currents to fit into a given Δ.

For the purposes of our simulations, each component in Table 2 is assumed to dissipate equal current over its entire latency. Changing this assumption would merely require changing the allocations and would not substantially alter pipeline damping. Non-variable components, such as the global clock, do not contribute to current variability and are not included. The front-end is shown as a single value because we do not individually damp front-end components.

5 Results

First, we present our bounds on current variability using pipeline damping compared to an undamped processor. Second, we show that pipeline damping effectively reduces current variability with small performance degradation and energy-delay increase. Then we show results for pipeline damping for different resonant periods. Finally, we compare the performance and energy impact of pipline damping to a simple peak-current limitation technique for reducing current variation.

5.1 Bounding variability with pipeline damping

5.1.1 Bounding current variability

From Section 3.1, we know that $\Delta = \delta * W$, where Δ is the worst-case current variability allowed during W cycles. W, half of the resonant period, is known at design-time. As per this equation, we pick δ to guarantee current variability over W cycles to be less than Δ. In this section, we show how our worst-case guarantee of Δ, corresponding to representative values of δ, compares to the worst-case current variability in an undamped processor. δ and therefore Δ are specified using the same integral units used in Table 2. Our representative values for δ are 50, 75, and 100. We show results assuming a resonant period of 50 cycles (the window size, W, is 25 cycles). Other resonant periods will be shown in the next section.

Because some of our configurations do not damp the pipeline front-end, we use the equation shown in Section 3.3 to compute Δ instead of the simple $\Delta = \delta W$. In the equation, as shown in the first row of Table 3, $\Delta = \delta W$ + maximum undamped components. The undamped component is the per-cycle front-end current times the window size for the configurations where the front-end is not "always on" as discussed in Section 3.2.2. When the front-end is always on, the undamped component is zero. Table 3 shows the values of the undamped components, δW, and Δ for our values of δ both with and without the "always-on" technique.

The worst-case current variation in the undamped processor is shown in the last row of Table 3. This value is computed by assuming the processor has minimum clock-gated current corresponding to zero instructions issued in one window, and increases rapidly to maximum current corresponding to the maximum number of ALU instructions issued in the next window. Because there are 8 integer ALUs with one-cycle latency they are a better choice to maximize current than less available or longer-latency resources. The current is lower for the first few cycles of the ramp-up as the first operations propagate down the pipeline and begin consuming current at the ALUs, result busses, and register write. The details of the computation are not shown.

The *relative worst-case* Δ values shown in the right-most column are ratios of guaranteed worst-case current variation for the given damping configuration to worst-case current variation in an undamped processor. We see that pipeline damping reduces the worst-case current variability between 14 and 61 percent compared to an undamped processor.

Reduction in worst-case current variation at the resonant frequency corresponds to reduction in worst-case supply noise. Our reductions can be put in perspective by comparing to the expensive, circuit-based voltage regulator proposed in [7]. Figure 10 in [7] shows that their regulators reduce voltage variation from about 0.2 volts to about 0.1 volts. The reduction is about 40%, similar to the relative worst-case Δs for pipeline damping (recall current variation is proportional to voltage variation as L * di/dt). Given a value of L for the circuits, our reduction in worst-case current variation will correspond to a specific voltage variation and can be ensured to be within the noise margin of the processor's circuits. The value δ can be adjusted to provide an appropriate *guarantee* of worst-case voltage variation, reducing the need for expensive circuit solutions.

While Table 3 shows the *guaranteed* worst-case current variation (Δ) computed using our values for δ, we now show *observed* worst-case current variations over 25 cycles for our benchmarks using simulation. The variation shown for each benchmark is the largest current variation observed during the simulation.

Our simulation results show that the observed worst-case current variation stays well within the guaranteed worst-case shown in the right-most column of Table 3. The top graph of Figure 3 shows current variation for the top three damping configurations in Table 3 and the undamped case, based on actual currents reported by Wattch (*not* our integral estimates used for counting current allocations in damping and establishing current bounds). The observed worst-case current variation, shown on the Y axis, are all relative to the worst-case current variation in the undamped case. The various dashed lines represent the guaranteed worst-case variation for each δ value from the right column of Table 3.

For the δ = 50, 75, 100, and undamped cases respectively, the largest observed worst-case variation is 83% (*gap*), 68% (*gap*), 58% (*gap)*, and 78% (*crafty)* of the guaranteed worst-case bound. While the difference between the observed worst-case and guaranteed worst-case may seem large, it is important to note that guaranteed bounds are theoretical worst-case, and most applications do not demonstrate theoretical worst-case behavior at the resonant frequency. Although the theoretical worst-case variation may not often be observed, such variation is possible and *must* be within the constraints of the circuit; guaranteeing better bounds aids circuit designers in avoiding expensive solutions.

Even under damping, the observed worst-case does not always approach the guaranteed worst-case because damping controls discrete, high-current events, such as integer ALU operations. Because of the high current of many events, it may not be feasible for damping to allow variation to approach arbitrarily close to the bound while guaranteeing that future cycles will not exceed the bound.

5.1.2 Performance and energy impact

In this section, we evaluate the performance and energy impact of the pipeline damping configurations discussed in the previous subsection. As discussed in Section 3.2.1, upward damping decreases performance by slowing increases in ILP, while downward damping increases energy by activating otherwise-unneeded functional units to maintain current. The lower graph of Figure 3 depicts performance degradation (black sub-bars, scale on left) and

energy-delay (full bars, scale on right) for each benchmark with respect to the undamped case. There is no bar for the undamped case because it is the reference.

The performance and energy penalties decrease with the tightness of the δ constraint. The tight constraint of δ = 50 results in substantial performance and energy penalty, while the looser constraints have less severe impact. For δ of 50, 75, and 100, the average performance degradations are 14%, 7%, and 4%, respectively. The corresponding average processor energy-delays are 1.17, 1.09, and 1.05.

The tight δ=50 constraint results in substantial performance penalty for some applications in order to achieve the 61% reduction in worst-case current variation. *Fma3d* stands out particularly with a 51% performance degradation and a relative energy-delay of 1.74. *Fma3d* the highest-IPC benchmark (4.1) in our simulations and is unable maintain that throughput under the constraints of damping at this frequency. The energy-delay increase in this case is due primarily to the increased execution time, not downward damping.

Using the "always on" front-end damping technique further reduces the variation bound and narrows the gap between the maximum observed current variation and the worst-case allowed Δ at the expense of additional energy. The middle three rows (W = 25) of Table 4 summarize results for all applications both without front-end damping (left half) and with front-end damping (right half). The left half results repeat those already given and are shown for reference. We see that the expense of the tighter current variation bound of the "always on" front-end is an average relative energy delay increase between 0.07 and 0.14. The slight narrowing of the gap between maximum observed Δ and worst-case allowed Δ occurs because the uncertainty of the undamped front-end is removed.

5.2 Pipeline damping at different periods

In the last section, we showed results with W = 25, but the resonant time period could have a value other than 25 that is on the order of 10 to 100 times the clock period. We show other values of W in this section. We expect a specific resonant period value not to affect damping, and that damping will achieve similar variation bounding, performance penalty, and energy-delay penalty for any resonant period. While it may seem that using the same δ for a larger (or smaller) window would loosen (or tighten) the variation bound, it is important to remember that in terms of di/dt, δW is an expression of di. Because the corresponding dt of the resonant period is also expressed by W, di/dt is controlled by δ, independent of W.

Table 4 shows results for damping corresponding to resonant periods of 30, 50, and 80 cycles (and W values of 15, 25, and 40, respectively). The results for the W of 25 have already been discussed in the previous section. Because the details of Δ computation for W values of 15 and 40 are identical to that for W of 25 (discussed in Section 5.1.1), we omit the details here. We show the relative worst-case Δ corresponding to earlier values in the rightmost column of Table 3. The "observed worst-case" column represents the worst-case variation observed among all 23 benchmarks simulated; the performance and energy-delay values are averages across the 23 benchmarks.

From the relative worst-case Δ columns, we see that for the same δ value, the guaranteed current bound becomes slightly tighter (i.e., smaller in value) for longer periods. The bounds become tighter because the first few low-current cycles in the worst-case current window for an undamped processor (discussed in Section 5.1.1) are less dominant over longer windows. We also see that the worst-case observed in our simulations, as a percent of Δ, approaches higher values for the shorter windows, even reaching 100% once. Shorter windows are more likely to experience bursts of worst-case variation than long windows. For example, it is unlikely that a processor would issue at the maximum issue width for 40 consecutive cycles.

Performance degradation and energy-delay increase do not change substantially with window sizes. For all window sizes, average performance degradation for δ of 100 is 5% or less, and the average degradation for δ of 75 is 8% or less. δ of 50 has substantial average performance and energy penalties for all window sizes because the bound is so tight.

5.3 Comparison to peak current limitation

In this section, we compare the performance and energy

Table 3: Computed integral current bounds for window size (W) of 25 cycles

Configuration	Max undamped over W	δW	Δ=worst-case variation over W	Relative worst-case Δ
δ=50	250	1250	1500	0.47
δ=75	250	1875	2125	0.66
δ=100	250	2500	2750	0.86
δ=50, frontend always on	0	1250	1250	0.39
δ=75, frontend always on	0	1875	1875	0.59
δ=100, frontend always on	0	2500	2500	0.78
undamped processor (no δ)	N/A	N/A	undamped variation = 3217	1.00

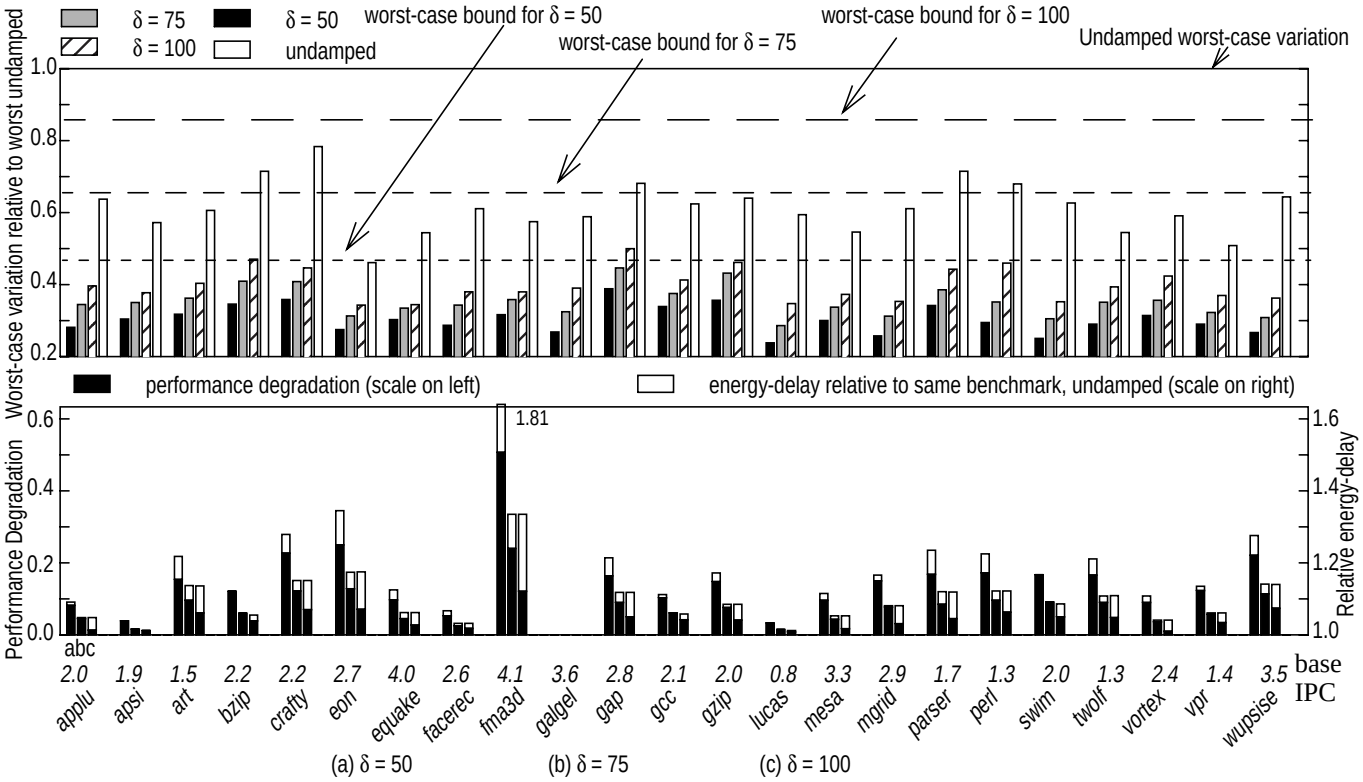

FIGURE 3: Current variation and performace/energy-delay penalty for W=25.

penalties of pipeline damping to schemes which reduce current variation by limiting *peak* current instead of limiting rate of change like pipeline damping. When using peak current limitation to achieve the same current variation bounds as pipeline damping, we expect large performance and energy penalties.

The graphs in Figure 4 plot guaranteed worst-case variation bounds against performance degradation and energy-delay for six peak-current-limiting configurations (**a** through **f**) and three pipeline damping configurations (**S** through **U**). The performance degradation and energy-delay values are averages across the 23 benchmarks simulated. The window size is 25 cycles, and front-end damping is not applied. The current limiting configurations achieve current variation bounds the same as those of the damping schemes by setting the peak per-cycle current to be the same as δ. Thus, the maximum total current over a window of W cycles is the peak per-cycle current multiplied by the window size.

Limiting peak current results in performance and energy-delay penalties that dramatically increase as the bound becomes tighter. Comparing the performance and energy-delay trends of the peak current limiting schemes to the

Table 4: Results for W = 15, 25, and 40.

W	δ	(without front-end damping)				(front-end "always on")			
		Relative worst-case Δ	**observed worst-case as % of Δ**	**avg perf. penalty**	**avg e-delay**	**Relative worst-case Δ**	**observed worst-case as % of Δ**	**avg perf. penalty**	**avg e-delay**
15	50	0.53	95	12	1.15	0.41	100	12	1.23
	75	0.72	77	6	1.07	0.62	78	6	1.14
	100	0.92	67	3	1.04	0.83	66	3	1.11
25	50	0.47	83	14	1.17	0.39	89	14	1.26
	75	0.66	68	7	1.09	0.59	70	7	1.23
	100	0.86	58	4	1.05	0.78	59	4	1.12
40	50	0.45	65	15	1.18	0.38	70	15	1.27
	75	0.64	54	8	1.10	0.58	55	8	1.17
	100	0.83	46	5	1.06	0.75	46	5	1.12

Peak Current Limiting Schemes		Damping	
id	Max current allowed from variable components	id	δ
a	100	S	100
b	90	T	75
c	80	U	50
d	75		
e	60		
f	50		

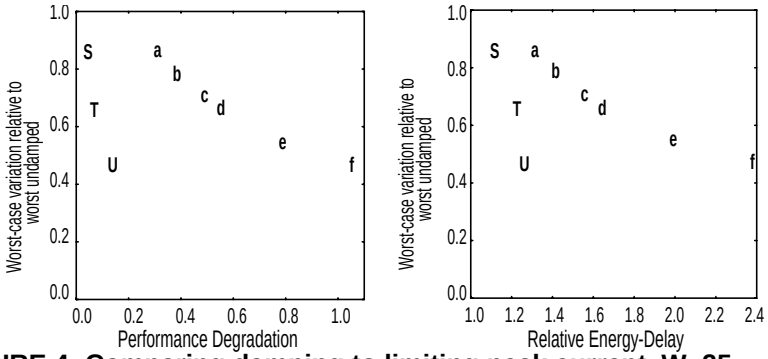

FIGURE 4: Comparing damping to limiting peak current. W=25.

trend of the damping schemes indicates that damping penalties increase more slowly as current bounds become tighter. To achieve the same variation bound as our δ = 100 scheme, peak current limitation incurs a total performance degradation of 31% with relative energy-delay of 1.31. Pipeline damping's degradation of 4% with relative energy-delay of 1.12 seems small in comparison. As the variation bound becomes tighter, the performance degradation of the current-limiting scheme increases to 105%. Relative energy-delay increases to 2.39 compared to the undamped case. Both are much worse than the 14% performance penalty and 1.26 relative energy-delay experienced by our tightest damping configuration with δ of 50.

6 Related Work

Previous circuits work focused on current spikes due to component-level clock gating but not processor-level ILP variation. [10] proposed a circuit-level mechanism to reduce current spikes due to clock gating by slowly turning off clock-gated units at a modest cost in hardware and performance. Others have improved this scheme to reduce the performance loss [11].

A recent paper discusses microarchitectural simulation and control of supply noise [6]. The authors propose a resonant circuit model for supply noise and observe that avoiding stimulus at the resonant frequency addresses the supply noise problem. The authors then suggest an architectural framework that prevents constraint violations by reacting to large changes in current. Their technique computes weighted sums of previous cycle currents, converts the values to voltage, and uses a "convolution engine" to determine if additional instructions may be issued without violating voltage constraints.

The authors of [6] mention that the latency of their convolution engine may necessitate pipelining it; the convolution engine thus would be placed in parallel to the front-end of the pipeline so that results would be available in time for issue. Delay in the convolution engine may complicate reacting to changes in voltage before constraint violations occur. In contrast to the complications of the convolution engine, damping involves simple counting of current allocations.

A simultaneous architectural work on supply noise in the 10-100MHz range appears in [9]. The authors create a "di/dt stressmark" to stimulate a microprocessor at its resonant frequency and evaluate the resonant behavior of applications. The authors also propose an architectural technique to react to voltage emergencies by gating/firing functional units when the supply voltage drops too low/high.

The technique in [9] senses small variations in voltage and responds, after allowing for sensor delay, by gating functional units and caches before violation of worst-case constraints. Pipeline damping and this technique are *fundamentally* different. Pipeline damping can be thought of as proactively preventing variation while this technique aims to cure reactively variations before constraint violations occur.

7 Conclusions

Inductive noise in power supply induced by switching-current surges in the processor circuitry degrades data integrity causing reliability problems. The key reason for inductive noise is ILP variation causing current changes at a specific, resonant frequency of the processor's RLC circuits and stimulating large variations in the supply voltage. We proposed pipeline damping, an architectural technique that controls instruction issue to guarantee bounds on current variation at resonant frequencies which are 1/10th to 1/100th of the clock frequency. Damping is an alternative to expensive, circuit-based noise-reduction techniques. We made the fundamental observation that limiting the current flow change (di) to a pre-specified value within the resonant time period (dt) controls di/dt without large performance loss. Damping guarantees bounds on current variation while allowing processor current to increase or decrease to the magnitude required to maintain performance.

We showed that pipeline damping can guarantee reductions in worst-case current variation for resonant frequencies in the range of 1/10th to 1/100th of the clock frequency. For a resonant frequency which is 1/50th of the clock frequency, pipeline damping guarantees a 33% reduction in

worst-case inductive noise while incurring only a 7% performance degradation and 1.09 relative energy-delay over an undamped processor. Pipeline damping compares favorably to peak-current limitation schemes, which incur 55% performance degradation to achieve the same bound on current variation. The performance gap between damping and peak-current limitation increases as current variation bounds become stricter. As supply voltages and processor noise-margins continue to shrink, damping-like techniques will become important for bounding inductive noise while maintaining performance.

Acknowledgements

We would like to thank Mark Horowitz for his help in fine-tuning the description of inductive noise in the power supply and for pointing out the effect of inaccuracies in current estimation. We would also like to thank David Ayers, Doug Carmean, and Edward Grochowski for their insight on inductive noise in microprocessors.

This research is supported in part by NSF under CAREER award 9875960-CCR, NSF Instrumentation grant CCR-9986020, SRC contract 2000-HJ-768, DARPA contract F33615-02-1-4003, an Intel Ph.D. fellowship and an NSF Graduate Research Fellowship.

References

[1] W. Becker, H. Smith, T. McNamara, P. Muench, J. Eckhardt, M. McAllister, and G. Katopis. Mid-frequency simultaneous switching noise in computer systems. In *1997 Electronic Components and Technology Conference*, pages 676–681, 1997.

[2] D. Boggs. Breathing life into a paper tiger. In *Keynote speech: 33rd International Symposium on Microarchitecture*, Dec. 2000.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.

[4] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, June 1997.

[5] D. Carmean. Personal communication. Feb. 2002.

[6] E. Grochowski, D. Ayers, and V. Tiwari. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *Eighth International Symposium on High Performance Computer Architecture (HPCA)*, pages 7–16, Feb. 2001.

[7] R. Heald, K. Aingaran, and et. al. A third-generation SPARC v9 64-b microprocessor. *IEEE Journal of Solid-State Circuits*, 35(11):1526–1538, Nov. 2000.

[8] D. J. Herrell and B. Beker. Modeling of power distribution systems for high-performance microprocessors. *IEEE Transactions on Advanced Packaging*, 22(3):240–248, 1999.

[9] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high-performance processors. In *Ninth International Symposium on High Performance Computer Architecture (HPCA)*, pages 79–90, Feb. 2003.

[10] M. Pant, P. Pant, D. Willis, and V. Tiwari. An architectural solution for the inductive noise problem due to clock-gating. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 255–257, 1999.

[11] A. Tang, N. Chang, S. Lin, W. Xie, S. Nakagawa, and L. He. Ramp up/down floating point unit to reduce inductive noise. In *Lecture Notes in Computer Science*, volume 2008, pages 291–321, 2001.