# Mitigating Inductive Noise in SMT Processors[*]

Wael El-Essawy and David H. Albonesi
Department of Electrical and Computer Engineering, University of Rochester

## ABSTRACT

Simultaneous Multi-Threading, although effective in increasing processor throughput, exacerbates the inductive noise problem such that more expensive electronic solutions are required even with the use of previously proposed microarchitectural approaches. We use detailed microarchitectural simulation together with the Pentium 4 power delivery model to demonstrate the impact of SMT on inductive noise, and to identify thread-specific microarchitectural reasons for high noise occurrences. We make the key observation that the presence of multiple threads actually provides an opportunity to mitigate the cyclical current fluctuations that cause noise, and propose the use of a prior performance enhancement technique to achieve this purpose.

**Categories and Subject Descriptions:** C.1.0 [Processor Architectures]: General

**General Terms:** Reliability, Design, Performance

**keywords:** power delivery, inductive noise, clock gating, SMT

## 1. INTRODUCTION

A long-standing problem in computer systems is that of *inductive noise*. Inductive noise, or the *Ldi/dt problem*, arises when there are large fluctuations in current through the power delivery network. The resulting variations in supply voltage reduce transistor drive current, and hence speed, for supply undershoots, and increase transistor electric field magnitudes for overshoots. If not adequately limited, these fluctuations can result in operational failure.

The nature of current fluctuations during machine operation impacts the degree to which the supply voltage is affected. The magnitude of the fluctuations has an obvious impact, but the *periodicity* is also important. In particular, large, periodic current variations at the *resonance frequency* of the chip capacitances and package inductance can result in significant supply voltage variations.

From a microarchitectural standpoint, therefore, a design in which current can vary from small to large values (and vice-versa) is generally more vulnerable to inductive noise than one in which current levels are more tightly bound. For instance, clock gating, although effective at reducing average dynamic power, increases the minimum to maximum possible current swings, and thus may lead to higher inductive noise [3, 5, 14]. Furthermore, a design in which a series of microarchitectural events cause these current swings to occur at the resonance frequency is particularly vulnerable to inductive noise. While years ago the clock frequency was often well below the resonance frequency, today the situation is reversed. While microprocessor clock frequencies have increased significantly over time, due to the fact that capacitances are increasing while package inductance keeps decreasing, the resonance frequency has remained in the tens of MHz range. Thus, in modern processors, periodic behavior involving, for instance, cache misses, at resonance can seriously exacerbate inductive noise levels.

Simultaneous Multi-Threaded (SMT) processors are potentially more vulnerable to inductive noise than single-threaded superscalar designs. A natural downside of SMT processors is their larger power dissipation, due to the fact that they require additional resources (registers, for example) and that they make better use of these resources (thereby dissipating more energy) over a given period of execution. This higher power dissipation, and thus current consumption, can lead to larger current fluctuations, and thus more inductive noise. The result is that a more robust power delivery system is required for SMT than for single-threaded processor. In order for multithreading to become more prevalent in everyday systems, it is crucial to limit power delivery network costs as much as possible. Thus, higher-level microarchitectural techniques need to be devised that specifically target the causes of inductive noise in SMT processors.

The objectives of this paper, therefore, are twofold. First, we wish to shed insight on the occurrences of high inductive noise from the perspective of the microarchitecture. Through detailed simulation, we show how various microarchitectural events lead to high noise, and examine the impact of increasing the number of threads on inductive noise for a given power delivery network.

Our second objective is to devise simple mechanisms for SMT processors that complement previously devised single-threaded approaches. A key observation is that the multiple threads of an SMT processor can be harnessed to naturally even out the usage of the processor (and thus limit the occurrences of high current fluctuations).

## 2. HANDLING INDUCTIVE NOISE IN MODERN PROCESSORS

Traditionally, limiting inductive noise to a permissible level has been exclusively handled through electrical solutions such as the use of decoupling capacitance at various levels of the system. However, several microarchitectural-level approaches for handling inductive noise have been recently advocated for the purpose of reducing the rising costs of these purely electronic solutions.

Perhaps the first microarchitectural level approach to reducing inductive noise was proposed by Pant et al. [10]. The premise of this approach is that the rapid current swings introduced by clock gating are a primary reason for high inductive noise. The technique, therefore, more gradually activates/deactivates functional units. The smoother current transition of this scheme comes at the cost of both lower performance and higher average power. Tang et al. [13] attempt to reduce this overhead by predicting when an instruction is to be issued to the functional unit, and start gradual wake-up prior to the issue, thereby reducing the unit wake-up delay.

Grochowski et al. [6] propose to reduce inductive noise via a global feedback and control system. The authors create an RLC model for the power delivery network and model the system as a Linear Time Invarient system whose input is the processor current consumption, and output is the power distribution voltage. The authors suggest a global system for obtaining various unit currents on a cycle-by-cycle basis in order to schedule instruction flow.

We focus on two more recently advocated approaches that provide a more comprehensive solution to guaranteeing a particular level of supply voltage integrity. The approach of Joseph et al. [8] is to intervene before the supply voltage reaches a level that can result in failure. An on-chip voltage sensor detects when the voltage crosses a threshold approaching such an *emergency level* and this triggers either the gating or firing of functional units and caches to increase/decrease current levels to stave off the emergency.

An alternative recent technique of Powell and Vijaykumar [11] prevents large current fluctuations from occurring at the resonance fre-

[*]This work was supported in part by NSF grants CCR–9701915, CCR–9811929, by DARPA/ITO under AFRL contract F29601-00-K-0182; by an IBM Fellowship; and by an IBM Faculty Partnership Award.

quency. The authors observe that such fluctuations are a result of variations in instructions per cycle (IPC) at resonance. The proposed *damping* technique identifies when such resonances can potentially occur, and limits the permissible variations in current that can occur at resonance. Large current increases are prevented by gating commit and possibly issue where appropriate, while large current downswings are avoided by firing gated-off units.

Although both of the latter two techniques are potentially effective for single-threaded designs, multi-threaded processors stress these approaches in different ways. The sense-and-intervene approach incurs a performance overhead whenever active units must be gated off to reduce current, and expends additional energy whenever a clock gated unit must be fired to increase it. The power delivery network must be robust enough to limit the extent of these occurrences, lest too large a performance and/or energy penalty be paid. As shown in Section 4, current fluctuations grow in general with the number of threads and thus for a constant power delivery network, so does the number of interventions. This in turn, increases the performance and power overheads of intervention. Thus, to keep these to a tolerable level, an SMT processor demands a more expensive packaging solution than a single-threaded one with the use of intervention.

In terms of the damping technique, we note that the amount of damping, and thus the performance and energy overheads, is partly a function of the frequency with which large fluctuations in instruction issue occur at resonance. For many common applications such as SPEC2000, issues of more than four instructions are rare, and this characteristic fundamentally limits the damping overhead to a reasonable level in conventional processors. SMT processors, by their nature in sharing issue bandwidth among multiple threads, increase the prevalence of large group issues, while at the same time, events such as cache misses still cause periods of low instruction issue. Thus, the range of IPC and therefore the overhead of damping naturally rises in an SMT processor. As with intervention, one solution is to employ a more expensive package with an SMT processor and to lessen the rules for engaging damping.

We make the key observation that the multiple threads of an SMT design can be used to even out the usual fluctuations caused by events that, when occurring in a periodic manner, can result in large current fluctuations at resonance. We propose that intelligent *thread management* techniques such as those proposed to improve performance [15] or reduce energy [4] can be used to naturally even out the usage of processor resources in an SMT machine, and thereby reduce the occurrence of large resonances. It is important to note that we do not propose to guarantee a limit on the amount of noise that can occur. Rather, we propose to reduce the *frequency* of high noise situations in SMT processors to an acceptably small level, and use a technique such as damping to intervene on these rare occurrences. With these two complementary mechanisms in place, a less expensive power delivery system can be safely employed while keeping performance and power overheads to reasonable levels.

Before discussing such techniques in Section 5, we make some observations about inductive noise in SMT machines in Section 4. First, we discuss our simulation methodology.

## 3. SIMULATION METHODOLOGY

We use a heavily modified version of the SimpleScalar toolset [2] for our simulations. We have created a version that models in detail an SMT processor running multiple programs as independent threads. The baseline microarchitecture resembles the Mips R10000 and Alpha 21264 with a Reorder Buffer (ROB) and separate integer, floating point, and load/store queues. Each thread has a separate Program Counter and ROB but otherwise shares the resources of the machine. The major simulation parameters are shown in Table 1.

In Watch [1], the absence of clock gating simply causes the peak power value to be reported. We produced a more realistic no-clock-gating model in which the full clock power (including latches) is consumed every cycle, but the combinational logic power varies according to activity [3]. We compare the results of this model with that of the typically-used CC3 clock gating model. Our modified version of Wattch tracks power dissipation (and thus current delivery) in all microprocessor units on a cycle-by-cycle basis, and calculates noise using a power delivery model based on that of the Pentium 4 microprocessor at 1.5 GHz [7]. The model, shown in Figure 1, includes the inductance and resistance in the power delivery system as well as both high frequency ceramic and low frequency bulk decoupling capacitances. It accounts for on chip decoupling and capacitor parasitics: the ESR (effective series resistance) and ESL (effective series inductance) of the typical industrial capacitors used in such a network. It also models the effective resistance and inductance of the board and package wires. The resonance frequency of the network is 68MHz, or roughly 22 processor clock cycles at 1.5GHz. The simulator tracks activities at fine granularity and outputs dynamic statistics for relating performance, power, and inductive noise to microarchitectural behavior.

**Table 1: SMT simulator parameters.**

| Parameter | Value (for 1/2/4/8 threads) |
|---|---|
| Clock Frequency | 1.5 GHz |
| Fetch/Decode width | 4/6/6/8 instructions |
| Branch Target Buffer | 2K entry, 2-way associative |
| Return Address Stack entries | 32 |
| Branch predictor | combination of 2K bimodal and 2-level |
| Branch mispredict penalty | 8 cycles |
| Reorder Buffer entries/thread | 128 |
| Fetch policy | ICOUNT.2.8 [16] |
| Integer physical registers | 80/164/256/480 |
| Floating point physical registers | 72/164/256/480 |
| Integer Issue Queue entries | 48/96/128/172 |
| Floating Point Issue Queue entries | 32/64/96/128 |
| Load/Store Queue entries | 64/88/106/160 |
| Issue width | 4/6/6/8 |
| Commit width | 4/6/6/8 |
| Integer ALUs | 4/6/6/6 |
| Integer mult/div | 1/2/2/3 |
| Floating point ALUs | 2/3/3/4 |
| Floating point mult/div | 1/2/2/2 |
| ICache | 32KB, 2-way, 1/2/4/8 banks |
| DCache | 32KB/64KB/64KB/128KB, 2-way, 2/3/4/4 ports |
| L2 Cache | 2MB, 8-way, 12 cycle latency |
| Main Memory latency | 160 cycles |

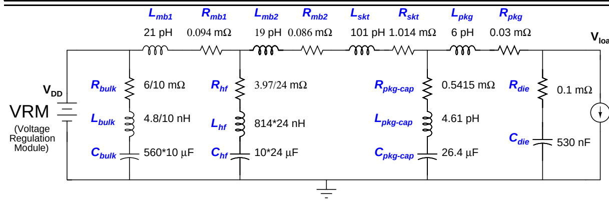

**Figure 1: Power delivery network model of the Pentium 4.**

We assume that the system is required to guarantee that no voltage variations higher or lower than 5% of the assumed 1.2V power supply voltage can occur. We constructed a variety of multi-threaded workloads from the SPEC2000 benchmarks, in order to generate a wide range of noise scenarios that would commonly occur in a real machine. This permits us to observe general trends and to study how microarchitectural events lead to high inductive noise as the number of threads is varied.

The workload mixes that we create for this purpose are shown in Table 2. We use the reference set for each benchmark and run each simulation for 100 million cycles after fast-forwarding each benchmark past the initialization phase (as identified in [12]).

The performance and energy of these workloads are given in Figure 2. For the rest of this paper, performance and energy results will be given relative to the data in this figure.

## 4. SMT INDUCTIVE NOISE ANALYSIS

In this section, we use our toolset to examine inductive noise in SMT processors. We maintain a constant power delivery model (the Pentium 4 model described in the prior section) as we vary the number

**Table 2: Workload mixes.**

| One thread | | Two threads | | Four threads | | Eight threads | |
|---|---|---|---|---|---|---|---|
| # | Benchmark | # | Benchmark | # | Benchmark | # | Benchmark |
| 1 | applu | 1 | applu, art | 1 | applu, art, equake, lucas | 1 | galgel, swim, mgrid, mesa, applu, art, equake, lucas |
| 2 | art | 2 | equake, lucas | 2 | gcc, mcf, perlbmk, parser | 2 | twolf, bzip2, gzip, vpr, gcc, mcf, perlbmk, parser |
| 3 | bzip2 | 3 | galgel, swim | 3 | galgel, swim, mgrid, mesa | 3 | galgel, parser, lucas, twolf, equake, bzip2, applu, vpr |
| 4 | gcc | 4 | mgrid, mesa | 4 | twolf, bzip2, gzip, vpr | 4 | mgrid, mcf, equake, bzip2, applu, vpr, art, gzip |
| 5 | equake | 5 | galgel, applu | 5 | applu, vpr, art, gzip | 5 | galgel, twolf, lucas, parser, equake, perlbmk, applu, cc1 |
| 6 | galgel | 6 | mesa, lucas | 6 | equake, bzip2, applu, vpr | 6 | mgrid, gzip, equake, perlbmk, applu, cc1, art, mcf |
| 7 | gzip | 7 | mgrid, equake | 7 | galgel, parser, lucas, twolf | 7 | applu, art, equake, lucas, twolf, bzip2, gzip, vpr |
| 8 | lucas | 8 | swim, art | 8 | mgrid, mcf, equake, bzip2 | 8 | galgel, swim, mgrid, mesa, cc1, mcf, perlbmk, parser |
| 9 | mcf | 9 | galgel, twolf | 9 | galgel, applu, swim, art | 9 | galgel, applu, swim, art, gzip, perlbmk, vpr, parser |
| 10 | mesa | 10 | mesa, vpr | 10 | gzip, perlbmk, vpr, parser | 10 | mgrid, equake, mesa, lucas, twolf, cc1, bzip2, mcf |
| 11 | mgrid | 11 | mgrid, gzip | 11 | mgrid, equake, mesa, lucas | 11 | applu, art, equake, lucas, cc1, mcf, perlbmk, parser |
| 12 | parser | 12 | swim, bzip2 | 12 | twolf, gcc, bzip2, mcf | 12 | galgel, swim, mgrid, mesa, twolf, bzip2, gzip, vpr |
| 13 | perlbmk | 13 | bzip2, mcf | 13 | applu, cc1, art, mcf | 13 | galgel, parser, lucas, twolf, applu, vpr, art, gzip |
| 14 | swim | 14 | gzip, perlbmk | 14 | equake, perlbmk, applu, cc1 | 14 | galgel, applu, swim, art, twolf, cc1, bzip2, mcf |
| 15 | twolf | 15 | twolf, gcc | 15 | mgrid, gzip, equake, perlbmk | 15 | mgrid, equake, mesa, lucas, gzip, perlbmk, vpr, parser |
| 16 | vpr | 16 | vpr, parser | 16 | galgel, applu, twolf, cc1 | 16 | galgel, twolf, lucas, parser, applu, cc1, art, mcf |

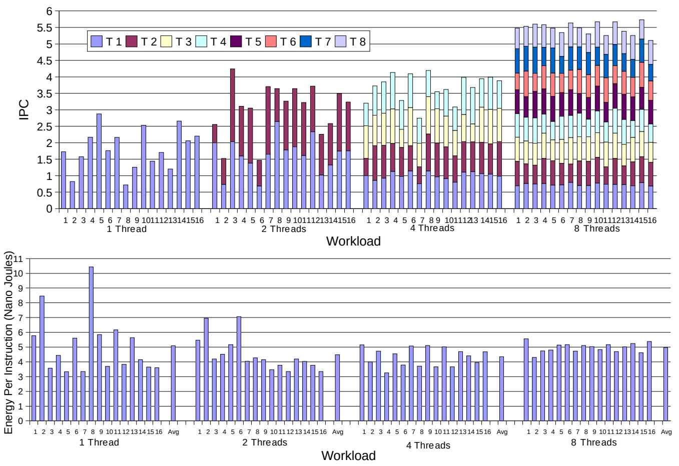

**Figure 2: Instructions Per Cycle (top) and Energy Per Instruction (bottom) of the baseline configurations of Table 1 for the workloads in Table 2. The IPC breakdown shows the individual thread contributions to the overall workload IPC.**

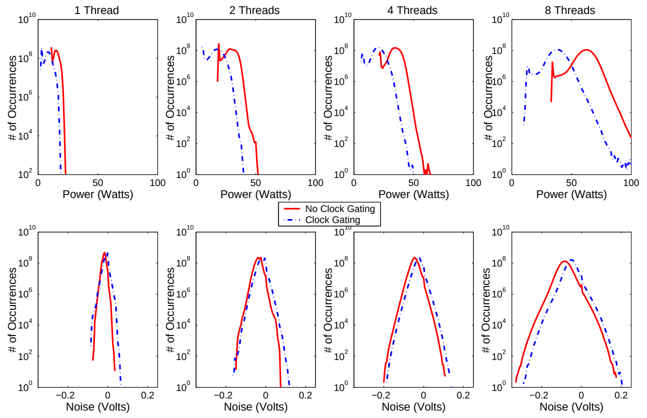

**Figure 3: Histograms of power and noise for clock gating, and no clock gating, with 1, 2, 4, and 8 threads.**

of threads. This serves to demonstrate how the magnitude of inductive noise, as well as the frequency of high noise situations, grows in general with the number of threads.

We first show general trends as the number of threads supported by the processor is increased. Figure 3 shows histograms of power dissipation and supply voltage noise (actual voltage value minus 1.2V) for one, two, four, and eight threaded machines, accumulated over all of our workloads, with resources scaled appropriately as described in Section 3 to match the number of threads. These plots show the number of occurrences of a given power or noise value during simulation with and without clock gating. The absence of clock gating results in a greater average power dissipation as shown by the power graphs for no clock gating being shifted to the right of that with clock gating in all four plots. Due to the resistance in the power delivery network, this results in greater undershoots as indicated by the noise plot for no clock gating being shifted to the left. This difference becomes more pronounced as the number of threads, and thus the power dissipation, is increased. However, we note that this resistive supply noise is small relative to the added inductive noise of clock gating.

As the number of threads is increased, more processor resources are needed and utilized, and thus the maximum power increases. The minimum power increases as well, but to a lesser extent, as the increase is largely due to static power, while both static and dynamic power contribute to the increase in maximum power. This greater power distribution variation results in greater supply noise as the number of threads is increased. For the case of clock gating, the noise almost doubles with an eight-threaded machine compared to one with two threads.

We now demonstrate the differences in high noise scenarios in single versus multi-threaded machines through our simulation tool, which we use to capture microarchitectural events that occur near high noise occurrences. Figure 4 shows a single-threaded example, the highest noise occurrence for *twolf* within the simulated window. Microarchitectural events such as the occurrence of cache accesses and misses (IL1 Access, DL1 Miss, DL2 Miss), register files becoming full, buffer occupancies, and issued, completed, and committed instructions, as well as current levels and voltage noise, are shown for the case of clock gating. This figure illustrates the primary cause of high noise in single-threaded machines: a microarchitectural event, in this case an L1 data cache miss, that causes a large drop in current, followed by a large increase in current, occurring repeatedly at resonance. The first series of L1 Dcache misses in this example occurs at a faster rate than the resonance frequency, while the second series occurs almost precisely at resonance. The resulting current oscillation from this second series of misses results in high supply noise. We have observed that a series of properly spaced L1 Icache misses or branch mispredictions can cause a similar situation. As is noted in [11], variation in issue rate at resonance is a good indicator of a high noise situation, and we see this correlation in this figure.

As the number of threads increases, the likelihood that a series of L1 cache misses or branch mispredicts will result in high noise decreases. With only two threads, certainly one thread may stall for a long period of time, for instance due to an L2 cache miss, while the other exhibits the periodic behavior such as that shown in Figure 4. However, with more running threads, other threads are likely to occupy the machine when these series of events occurs with a given thread, preventing the wide current swings that cause high noise levels in the single-threaded case. Although it is certainly conceivable to construct an L1 miss scenario where high noise can occur with this many threads, the key point is that the *probability* of such events, and thus the frequency in which intervention must occur, is drastically reduced. Such relatively short

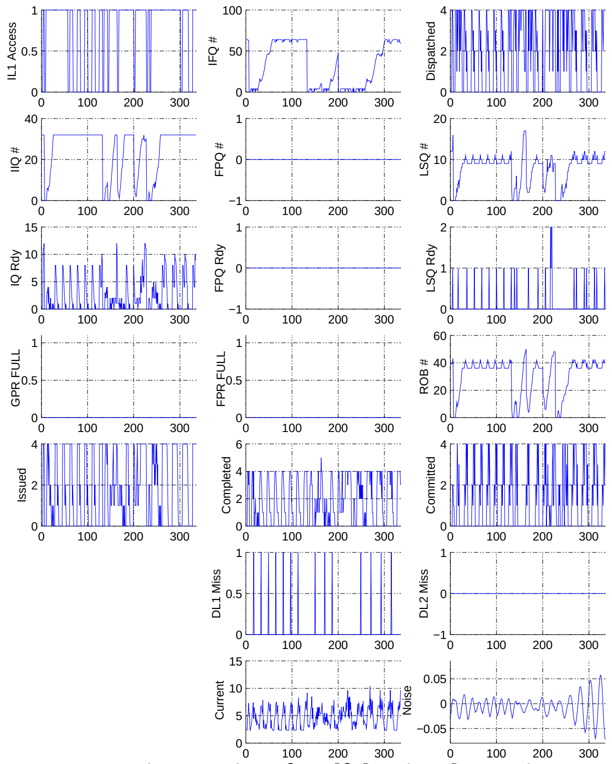

**Figure 4: Dynamic execution of *twolf* showing the maximum noise scenario within a 330 cycle window.**

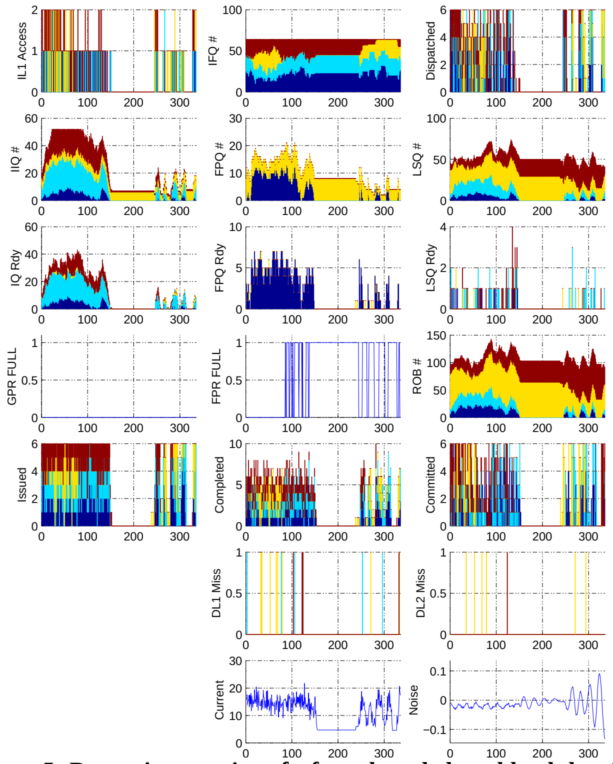

**Figure 5: Dynamic execution of a four-threaded workload showing the maximum noise scenario.**

latency events become less likely to result in a high noise situation as the number of threads increases.

Using our tool, we have discovered that the main reason for high noise with many threads is due to the hoarding of processor resources by one or a few threads, with periodic release of a subset of these resources, resulting in bursts of activity at resonance. This hoarding occurs when a non-blocking event causes a thread to fetch and execute a large number of instructions yet the event must complete before these instructions can be committed, thereby causing the thread to tie up many machine registers and issue queue slots. Events such as L1 cache misses that hit in the L2 cache are serviced fast enough to prevent significant resource hoarding from occurring. L2 cache misses, on the other hand, are long enough to cause this phenomena. Karkanis and Smith observed that the SPEC2000 integer benchmarks can continue to execute far beyond an L2 cache miss, so far as to fill the (single-threaded) machine resources [9]. Figure 5 shows one example of this resource hoarding with four threads and periodic freeing of a subset of resources as the result of a series of L2 cache misses. The different colors in the graphs show the resource occupancies and events for different threads. Two of the threads in this example experience L2 cache misses and find enough independent instructions to eventually accumulate most of the machine resources. As the data from cache misses returns, some of the machine resources are freed as dependent instructions execute. This permits dispatch of new instructions from the other threads and their subsequent execution until the point where the resources are once again fully consumed. These bursts of high activity occur periodically as L2 misses are returned (or as other resources are freed by other delayed instructions), resulting in high noise levels if these bursts occur at resonance (as in this figure). Note that issue rate variations remain a good indicator of high noise events.

To address this issue, we make the key observation that the multiple threads of a machine can be exploited to smooth out current flow. Other threads at times will naturally take up the slack in an L2 miss situation thereby avoiding high current fluctuations and the need for intervention or damping. In a similar fashion, resource hoarding can be *proactively* avoided in an SMT machine through intelligent thread management policies. It is this same *consistent balance* of resource allocation among threads that is sought in an SMT processor to achieve good performance and energy efficiency that can be exploited to reduce the probability of the occurrence of high inductive noise. With intervention or damping in place as a backup safeguard, a guaranteed noise limit can be achieved with little performance and power overhead.

## 5. THREAD MANAGEMENT POLICIES FOR REDUCING HIGH NOISE EVENTS

Fortunately, this "thread resource hoarding" behavior has been previously identified as a source of performance loss and energy inefficiency in SMT processors. In terms of the former, Tullsen and Brown [15] propose a scheme in which fetching is blocked from threads with an L2 cache miss and instructions from that thread following the miss are flushed from the machine. This frees up resources for other threads to make forward progress. Although proposed purely for performance reasons, this *flushing* scheme is designed to prevent thread resource hoarding and thus has the potential to reduce the frequency of high noise situations.

Similarly, El-Moursy and Albonesi [4] propose schemes for reducing the energy dissipation of the issue queues in SMT processors. The idea is to prevent the queues from being filled with instructions that are likely to sit idle in the queue for a long time by gating fetching from those threads under particular circumstances. However, because this approach focuses on shorter latency events that are not the primary cause of high noise with many threads, we chose to implement the flushing scheme.

We experimented with three different flushing approaches [15]. In each, instructions are flushed from a thread whenever it experiences a single L2 cache miss. No further instructions are fetched from the thread until the data returns. In the simplest implementation, all instructions in the ROB that follow the load that misses (and are from the same thread) are flushed from the pipeline. We chose this approach as it provides the best performance and was effective in reducing resource hoarding.

Because flushing only serves to reduce the frequency of high noise without guaranteeing a particular noise bound, a technique like intervention or damping must be in place as a safeguard. Although either can be used, we chose to implement damping in the results presented in the next section.

## 6. RESULTS

In this section, we compare the costs of various damping techniques with and without flushing compared to a baseline processor without such safeguards. With damping, it is necessary to maintain a limit on the amount of current variation that can occur between two points at half the resonance period [11]. If a larger than permissible current drop is to occur, *dummy instructions* are fired to limit the drop. In our implementation, we fire enough of these instructions to make up the current drop in the following priority order: floating point multiply, integer ALU, loads from L1 Dcache, and floating point ALU operations. These instructions consume power without altering the processor status. Similarly, if a larger than permissible current rise is to occur upon an instruction commit or issue, we gate instruction commit to the extent possible and also instruction issue if necessary. More details on damping can be found in [11]. Unlike the original approach, which fires the front end all the time, we permit clock gating as usual in the front end. Unlike single threaded machines, SMT processors have higher fetch power, and therefore forcing the front-end to be fully operational all the time becomes an expensive solution to the noise problem.

The amount of permissible current variation with damping is a tradeoff between the effectiveness in reducing noise and the performance and power overheads incurred. The smaller the permissible range, the less noise that is incurred but at greater performance and power overheads. As the best implementation is a function of the current range and thus the number of threads, we implemented several versions of damping with permissible current ranges of 2, 3, 4, 6, and 8 amps. These are referred to as the *SXX* configuration where *XX* represents the number of 100mA units in the permissible current range. We examined the effect of damping alone and when coupled with flushing as the number of threads is varied. For each option, we determined to what degree the maximum noise is reduced and the performance and power overheads.

Figure 6 shows the degree of reduction in the maximum noise undershoots and overshoots for the baseline configurations and with damping. For the baseline machine, the maximum noise undershoot increases by a factor of 2.2 for four threads, and by 3.3 for eight threads relative to the single threaded configuration. The undershoot noise is higher than the overshoot noise due to the resistive noise. This is exacerbated in larger machine configurations due to their larger current levels. As shown in this figure, the more tight the permissible damping current limit, the lower the noise level. This however comes, as we will present later, at both a performance and power cost. While tightening the allowed current limit up to 2 Amperes (S20) is effective in the single threaded configuration, most of the noise limiting gains in SMT processors are obtained by the 4 Amperes (S40) limit and above. These results demonstrate that architectural level techniques like damping can complement circuit level approaches to reduce overall design cost. The results show a reduction in the maximum noise of more than a half. This permits a less costly power delivery network to be used.

Damping combined with flushing achieves this goal at a lower energy and performance cost, and yet maintains the same noise levels obtained by damping alone. Part of the reason for this lower performance and energy cost is illustrated in Figure 7, which compares the power and energy histograms for the baseline and with flushing for the case of clock gating. With four and especially with eight threads, there are far fewer high noise occurrences with the use of flushing. (In fact, with eight threads, flushing even reduces the maximum over and undershoot values for our workloads.) The implication is that damping needs to be engaged less often with the use of flushing.

However, Table 3 shows that in practice, this is only partially true. This table shows the percentage of cycles in which the different damping schemes (with and without flushing) limit commit/issue to bound the current increase. Also shown is the percent reduction in these activities with the addition of flushing. As the number of threads increases, so does the amount of limit commit/issue activity. The impact is particularly pronounced for schemes with a less restrictive current limit. Less restrictive current limits cause intervention only when the processor switches between idle (because of resource hoarding, for example) and busy states. In all cases, flushing dramatically reduces the amount of this type of damping activity.

However, this table also shows that while flushing prevents resource hoarding by one or more threads, in doing so, it suddenly removes many instructions from the processor and hence causes a drop in the processor activity. When this drop exceeds the permissible current limit, firing occurs. With two threads the probability of exceeding the permissible current limit increases, causing a 19% increase in firing intervention cycles, and about a 25% increase in the firing power. The actual firing power, however, is very small; on average, the firing power for S40, S60, and S80 are 10, 50, and 150 mW, respectively. With more threads, the probability of falling below the permissible current limit because of flushing decreases, due to the fact that while one thread is being flushed many more threads are still active. Damping S60 and Damping S80 reduce firing incidents for four threads by 30% and 54%, respectively, and reduce the firing power by 29% and 52%, respectively. The tighter current limit of Damping S40 causes firing power to decrease by only 1% with four threads. For the eight thread configuration, the current variation permissible limit is too small causing an increase in firing power incidents for all three damping schemes. The increase is lower with less tight current limits, however.

Nevertheless, the firing penalty is smaller than the energy savings from flushing due to more efficient resource usage. Figures 8 and 9 compare the relative weighted performance improvement [15] and energy per instruction (EPI) of the various schemes relative to the baseline machine with no damping or flushing. Damping effectively reduces maximum noise levels with a modest overhead for a small number of threads. For example, S40 lowers noise levels significantly with a 0.6% performance penalty for a single thread, and a 2.4% performance overhead with four threads. The energy overheads are similarly modest. With eight threads, the performance and energy costs of S40 increase to 5% and 10%, respectively, and noise levels are reduced by about 40% compared to the baseline. S80 provides more tolerable performance and energy overheads for eight threads but at the cost of a 15% increase in the noise levels compared to S40.

When flushing is added to S40 with four and eight threads, performance improves by 13% and 5%, energy per instruction is reduced by and 14% and 9%, and noise levels are still within those achieved by damping. For four threads, adding S40 to the baseline reduces performance by an average of 2.4% and a maximum of 8.5%. When S40 is added to a baseline that uses flushing, the average and maximum performance degrades by only 1.1% and 2.3% respectively. Average and maximum energy do increase, but by a lesser amount: from 0.5% and 1.2% without flushing, to 1% and 1.9% with flushing. The fact that the maximum noise levels are reduced with flushing for eight threads

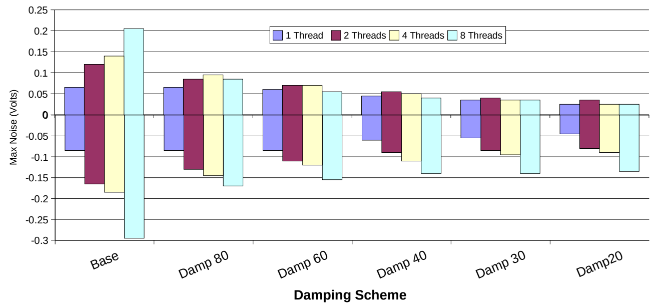

**Figure 6: Maximum noise overshoots and undershoots for the baseline and with damping with varying number of threads and with damping factor.**

**Table 3: Percentage of cycles in which damping limits commit/issue and fires units averaged over all the workload mixes for a given configuration. Also shown is the reduction in these events with the addition of flushing.**

| Configuration | # Threads | Limit Commit/Issue | | | Firing Incidents | | | Firing Power per Cycle (W) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | No Flushing | With Flushing | Reduction | No Flushing | With Flushing | Reduction | No Flushing | With Flushing | Reduction |
| Damping S40 | 1 | 1.4% | – | – | 1.1% | – | – | 0.04 | – | – |
| | 2 | 5.6% | 4.9% | 12% | 2.4% | 2.8% | -19% | 0.12 | 0.15 | -25% |
| | 4 | 5.3% | 3.3% | 38% | 3.0% | 2.9% | 2% | 0.16 | 0.16 | 1% |
| | 8 | 13.3% | 11.2% | 15% | 15.5% | 16.8% | -8% | 1.63 | 1.79 | -10% |
| Damping S60 | 1 | 0.3% | – | – | 0.2% | – | – | 0.01 | – | – |
| | 2 | 2.2% | 1.7% | 24% | 0.7% | 0.9% | -22% | 0.04 | 0.05 | -31% |
| | 4 | 2.2% | 1.0% | 54% | 0.8% | 0.6% | 30% | 0.04 | 0.03 | 29% |
| | 8 | 6.4% | 4.9% | 24% | 6.5% | 6.8% | -4% | 0.58 | 0.60 | -4% |
| Damping S80 | 1 | 0.0% | – | – | 0.0% | – | – | 0.00 | – | – |
| | 2 | 0.9% | 0.5% | 43% | 0.2% | 0.3% | -28% | 0.01 | 0.01 | -35% |
| | 4 | 1.0% | 0.3% | 73% | 0.3% | 0.1% | 54% | 0.01 | 0.01 | 52% |
| | 8 | 3.5% | 2.5% | 31% | 2.3% | 2.2% | 6% | 0.19 | 0.17 | 9% |

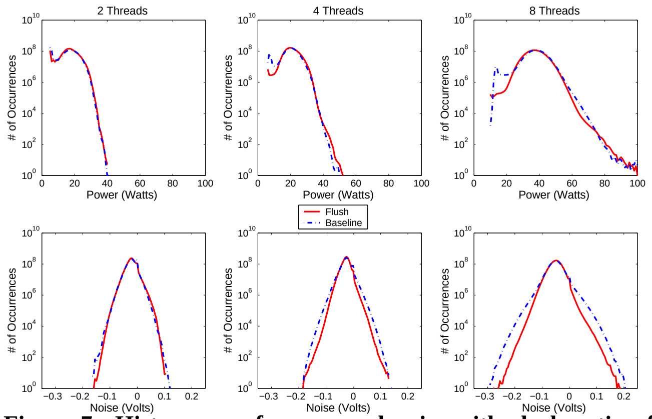

**Figure 7: Histograms of power and noise with clock gating for baseline and with flushing.**

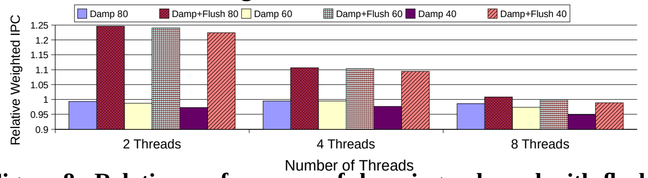

**Figure 8: Relative performance of damping only and with flushing averaged over the workload mixes. For a given number of threads, performance is relative to the baseline with that number of threads.**

indicates that it has mitigated most of the highest noise events found in our workload mixes, and that these are indeed due to resource hoarding as described in Section 4.

In summary, these results demonstrate how the multiple threads in an SMT machine provide the means to even out current flow in the presence of a potential high noise scenario. While we have shown that this may not always occur naturally, the use of proactive thread management techniques (such as flushing) can be used to mitigate these noise occurrences. With a complementary failsafe mechanism such as damping in place, along with an appropriate power delivery network, safe noise levels can be guaranteed with a large number of threads without compromising performance or energy.

## 7. CONCLUSIONS AND FUTURE WORK

Due to increasing current swings and the relationship between the operating frequency of the microprocessor and the resonance frequency of the power delivery system, inductive noise has become a major concern for microprocessor developers. Through a detailed modeling methodology, we have shown how SMT processors exacerbate inductive noise in ways that make current microarchitectural level techniques ineffective without more expensive electrical solutions or additional control. We make the observation that intelligent thread management can be used to provide this additional control. In particular, we use a previously developed performance technique to prevent L2 cache misses from one or more threads from hoarding machine resources and then periodically releasing a subset, a scenario that results in bursts of activity at resonance. With this approach applied to an SMT processor with damping, performance improves, energy per instruction is reduced, and noise is reduced to acceptable levels with a less expensive power delivery network.

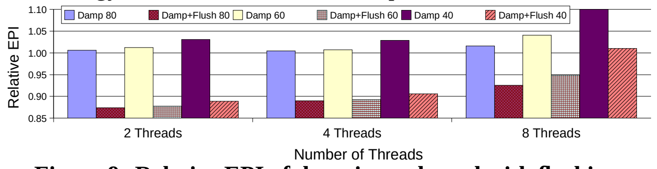

**Figure 9: Relative EPI of damping only and with flushing.**

## Acknowledgements

We wish to thank Ali El-Moursy for writing most of the baseline SMT-Simulator, Larry Smith for helpful discussions on power delivery network trends, Michael Powell for answering our questions regarding damping, and Dean Tullsen for answering our questions regarding flushing.

## 8. REFERENCES

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *27th International Symposium on Computer Architecture*, June 2000.

[2] D. Burger and T. Austin. The SimpleScalar tool set, version 2.0. Technical Report CS-TR-97-1342, June 1997.

[3] W. El-Essawy, D. H. Albonesi, and B. Sinharoy. A microarchitectural-level step-power analysis tool. *International Symposium on Low-Power Electronics and Design*, August 2002.

[4] A. El-Moursy and D. Albonesi. Front-end policies for improved issue efficiency in SMT processors. *9th International Symposium on High-Performance Computer Architecture*, February 2003.

[5] M. Gowan, L. Biro, and D. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. *35th Design Automation Conference*, June 1998.

[6] E. Grochowski, D. Ayers, and V. Tiwari. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. *8th International Symposium on High-Performance Computer Architecture*, February 2002.

[7] Intel Corporation. Intel Pentium 4 processor in the 423 pin package / Intel 850 chipset platform. *Intel Design Guide*, February 2002.

[8] R. Joseph, D. Brooks, and M. Martonosi. Control techniques to eliminate voltage emergencies in high performance processors. *9th International Symposium on High-Performance Computer Architecture*, February 2003.

[9] T. Karkhanis and J. E. Smith. A day in the life of a data cache miss. *Workshop on Memory Performance Issues*, May 2002.

[10] M. Pant, P. Pant, D. Wills, and V. Tiwari. An architectural solution for the inductive noise problem due to clock-gating. *International Symposium on Low-Power Electronics and Design*, August 1999.

[11] M. Powell and T. Vijaykumar. Pipeline damping: A microarchitectural technique to reduce inductive noise in supply voltage. *30th International Symposium on Computer Architecture*, June 2003.

[12] S. Sair and M. Charney. Memory behavior of the SPEC2000 benchmark suite. Technical Report RC21854, IBM, Watson, Oct. 2000.

[13] Z. Tang, N. Chang, S. Lin, W. Xie, S. Nakagawa, and L. He. Ramp up/down floating point unit to reduce inductive noise. *Workshop on Power-Aware Computer Systems*, November 2000.

[14] V. Tiwari et al. Reducing power in high-performance microprocessors. *35th Design Automation Conference*, June 1998.

[15] D. Tullsen and J. Brown. Handling long-latency loads in a simultaneous multithreading processor. *34nd International Symposium on Microarchitecture*, December 2001.

[16] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. *26th International Symposium on Computer Architecture*, May 1996.