Measuring Code Optimization Impact on Voltage Noise

Svilen Kanev Timothy M. Jones* Gu-Yeon Wei David Brooks Vijay Janapa Reddi†
Harvard University, The University of Cambridge, The University of Texas at Austin†*
{skanev, guyeon, dbrooks}@eecs.harvard.edu, timothy.jones@cl.cam.ac.uk, vj@ece.utexas.edu

***Abstract*—In this paper, we characterize the impact of compiler optimizations on voltage noise. While intuition may suggest that the better processor utilization ensured by optimizing compilers results in a small amount of voltage variation, our measurements on a Intel® Core^{TM}2 Duo processor show the opposite – the majority of SPEC 2006 benchmarks exhibit more voltage droops when aggressively optimized. We show that this increase in noise could be sufficient for a net performance decrease in a typical-case, resilient design.**

I. Introduction

Voltage variation is one of the major reliability challenges that processor designers face. Sudden changes in processor current draw, coupled with the non-zero parasitic impedance of the power delivery network result in fluctuations of supply voltage from its nominal value, also known as di/dt *noise*. Of such deviations, *voltage droops*, or downward voltage transients, are particularly dangerous, since they can lead to circuit timing errors and incorrect execution.

Traditionally, designers allocate generous operating margins to ensure correctness in the presence of voltage noise. Processors are typically designed for the worst-case, such that even under extreme current fluctuations, voltage does not cross the operating margin and circuit timing is met. Worst-case voltage margins can easily reach 20% of the nominal supply voltage [1], resulting in significant overheads in either performance or power consumption.

Researchers have proposed a variety of *resilient architectures*, which have tighter voltage margins to avoid the efficiency overheads of operating circuits far from their typical-case voltage targets. Such typical-case designs avoid correctness issues through different fallback recovery mechanisms that handle the rare cases when the aggressive voltage margins have been violated. The benefits from such resilient designs compared to the conventional case range from 20 to 40% in terms of either power consumption or performance [2], [3], [4].

In this paper, we assume such a resilient system and evaluate the effect of compiler optimizations on its noise behavior. We first go on to confirm the correlation between voltage noise and microarchitectural stall events, suggesting that stalls are a potential cause of voltage noise. This implies that an optimizing compiler, part of whose job is to eliminate stalls in order to achieve better machine throughput, can also eliminate some amount of voltage droops, decreasing the pressure on a recovery system.

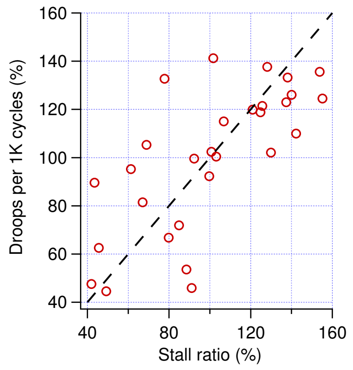


Fig. 1: Correlation between microarchitectural stalls and voltage droops for CPU2006 (normalized to suite average).

However, we show that the relationship between voltage noise and compiler optimizations is more complicated than such a simple causality. For the SPEC CPU2006 suite, on average, more aggressively optimized code also shows a larger number of voltage droops, with potentially larger magnitudes. In the context of resilient architectures, this implies that aggressive optimizations can even hurt performance, after one factors in the performance penalty of recovering from the larger number of voltage droops.

We finally investigate the microarchitectural reasons for this impact. We show that different stall events contribute to voltage noise to a different extent. For example, branch mispredictions create an almost 50% larger voltage swing than L2 cache misses. Thus, the typical optimizing compiler behavior of eliminating stalls that bottleneck performance does not necessary translate to eliminating noise-critical stalls. Such considerations need to be taken into account if one is to create a voltage-noise aware compiler for resilient architectures.

II. Stall events cause voltage noise?

The methodology to measure voltage noise in production processors has been thoroughly described in prior work [5]. To summarize, we use a high-frequency oscilloscope probe and a signal analyzer to connect to the voltage sense pins on a processor package. This allows is to measure supply voltage unintrusively, as the processor is executing real workloads. In the following experiments we use a Intel® Core^{TM}2 Duo processor, manufactured with 45nm process technology. The particular chip used for measurements in this paper has been

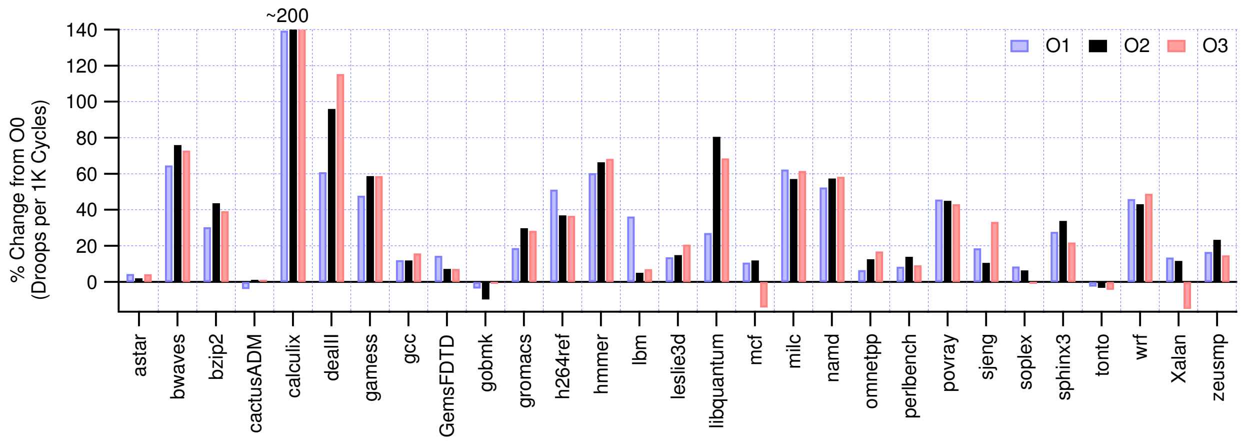


Fig. 2: Noise behavior for different degrees of compiler optimizations. O1 includes basic optimizations (f.e. dead code elimination), O2 increases their aggressiveness without increasing code size (f.e. instruction scheduling), while O3 includes the most aggressive and computationally heavy optimizations (f.e. function inlining and vectorization).

instrumented to retain only 3% of its original package decoupling capacitance, which brings its voltage noise levels close to projections for today's 22nm node. For subsequent experiments, we classify reductions in its supply voltage below an aggressive 2.3% margin as *voltage droops*[1].

Using this experimental setup, we can establish a correlation between voltage droops and microarchitectural stalls. Figure 1 shows the amount of voltage noise for the first 60 seconds of execution of the SPEC CPU2006 benchmarks. The metric shown, *Droops per 1000 cycles*, characterizes noise activity for the benchmarks, similarly to the more popular cache metric misses per 1K cycles. The different benchmarks show a large variance in droops per 1K cycles, indicating a heterogeneous mix of voltage noise characteristics. But, more interestingly, droop activity is correlated with the *Stall ratio*. Stall ratio is an aggregate metric of microarchitectural stalls, conveniently provided by VTune [6]. The linear correlation coefficient between droops and the stall ratio is 0.97.

III. Code optimization increases noise

The analysis so far strongly suggests a correlation between voltage noise and microarchitectural stalls. This can lead to the false assumption of direct causality – namely, that eliminating stalls directly decreases voltage noise. In order to show why the connection is more complex than that, we evaluated the effects of compiler optimizations on voltage noise. Compiler optimized code experiences a greater number of voltage droops, and in certain cases the magnitude of the droops is also noticeably larger. In a resilient design, this can eventually lead to a performance loss for the more aggressively optimized case.

A. *Impact on Droop Counts*

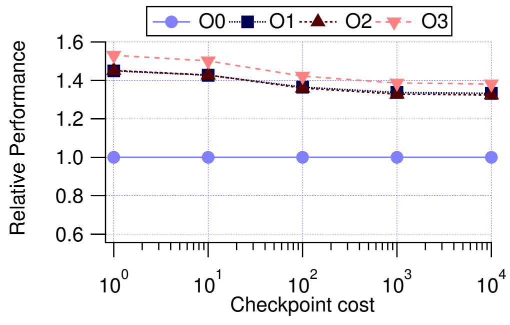
Typically, the task of an optimizing compiler is to increase instruction throughput through the processor. A large body of well-known optimizations (such as loop unrolling, instruction scheduling, register allocation) achieve that by eliminating various microarchitectural stalls. Thus, if the analysis in Figure 1 was interpreted as a simple and direct causality relation between the number of stalls and the number of voltage droops, one would expect that higher compiler optimization levels would decrease the amount of voltage noise, while increasing performance.

Data that we gathered for the GCC compiler contradicts with such a notion. Figure 2 shows the noise behavior of the single-core CPU2006 benchmarks, when compiled with optimization levels ranging from O0 to O3. We can see that increasing the aggressiveness of performance optimization with respect to the O0 baseline leads to a larger number of droops per 1K cycles in the majority of benchmarks. Out of the 29 runs in this experiment, 19 binaries compiled with maximum optimization result in a more than 10% increase in the number of droops, compared to the respective non-optimized versions. `454.calculix` shows the largest increase – at O3 its droop counts more than triple. The fluctuations for the other 10 benchmarks in the experiment are predominantly smaller. Note that in several cases these small fluctuations are negative, that is, better optimized code results in fewer droops.

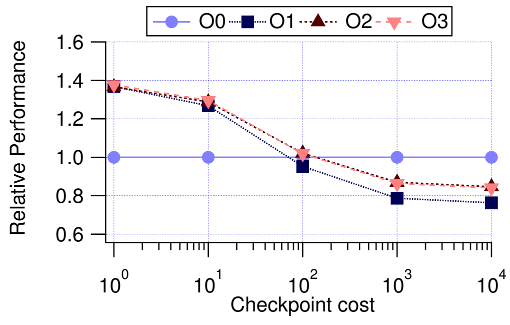
Looking at the more moderate optimization levels O1 and O2 does not show a qualitative difference. The set of benchmarks that show a large increase in droops remains largely unchanged, with the difference being in the magnitude of the increase. However, O3 does not always result in the largest noise increase, compared to O1 and O2.

The behavior of the majority of the benchmarks is easily explicable. When better optimized at O3, benchmarks exhibit a higher number of instructions per cycle. At the microarchitectural level, this implies that pipeline utilization is high, and consequently switching factors are larger, therefore the core consumes a relatively larger amount of current. On a stall, the net change in current is larger than in the unoptimized case. Since voltage fluctuations are proportional to such changes in

[1]The particular size of this margin has been chosen to cover idle OS activity, such that deviations below that value are caused by the respective benchmark.

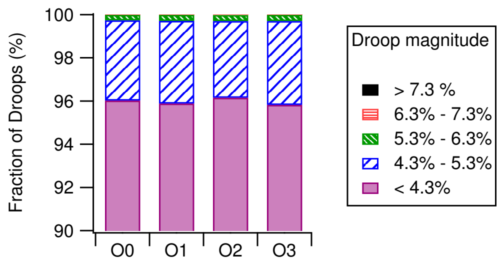


(a) `473.astar`

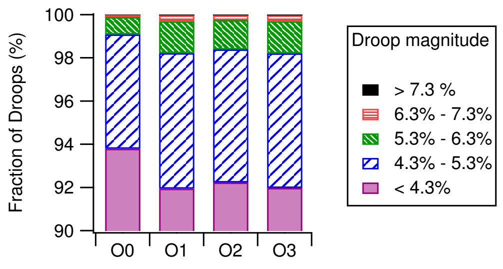


(b) `435.gromacs`

Fig. 3: Influence of compiler optimizations on performance for varying recovery costs in a resilient architecture.



(a) `473.astar`



(b) `435.gromacs`

Fig. 4: Influence of compiler optimizations on the magnitude of droops (relative to nominal voltage).

current, each stall is more likely to cause a subsequent voltage droop. This effect leads to a larger aggregate number of droops over the whole execution, even though the number of stalls may be smaller.

B. Impact on Net Performance

In a resilient design, the increased number of droops at higher optimization levels has a respective performance penalty. As a proof-of-concept that this penalty may be sufficiently large to even offset the initial performance gains from optimizing more aggressively, we analyze the net performance of one benchmark from each of the two groups outlined above (those that experience little change in their noise profiles, represented by `473.astar`, and those that have a significant increase, represented by `435.gromacs`). We account for the recovery cost of each voltage emergency below the hypothetical 2.3% margin using a simple performance model of a resilient architecture. Namely, each crossing of the aggressive margin triggers a fallback mechanism with a set checkpoint recovery cost in cycles. The cycles spent in recovery are added to the conventional running time of the benchmark for an estimation of its runtime on a resilient architecture with the specific recovery cost.

Figure 3 shows performance improvement achieved by the two representative benchmarks for different costs. Both benchmarks rightfully receive a significant performance gain from higher levels of performance-centric compiler optimizations. For `473.astar`, this gain is sufficient to sustain higher net performance even at very large recovery costs. Even though the gains diminish because of the slightly increased droop counts (and therefore emergency recoveries), in this case net performance is dominated by factors other than noise. For workloads represented by `435.gromacs`, fine-grained recovery presents similar results. However, after a certain recovery cost (100 cycles in this particular case), voltage noise effects begin to dominate over the initial performance gains and less optimized binaries achieve better net performance, after factoring in emergency recovery penalties. Even the modest 30% increase in relative droop counts that `435.gromacs` shows is sufficient to offset the 50% initial performance gains from compiling with O3.

C. Impact on Droop Magnitude

The magnitude of single droops can also be influenced by varying compiler optimizations. While droop magnitude is a second order effect that our performance model does not include, larger droops require more time for voltage to stabilize, which can potentially limit the minimum cycle cost for emergency recovery. We quantify the distribution of droop magnitudes for our two representative benchmarks. Figure 4 shows the top 10% of droops, broken down by the magnitude of voltage swing. The first benchmark, `473.astar`, does not show a significant change in droop magnitude across increasing compiler optimizations. This behavior is similar to the total number of droops seen in Figure 2. For benchmark `435.gromacs`, the relative severity of droop magnitude increases (see Figure 4b).

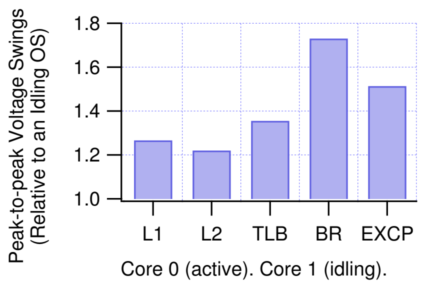


Fig. 5: Varying amount of noise for different stall events.

IV. Microarchitectural explanation

In order to better understand why more aggressively optimized code can lead to a larger number of voltage droops, we look more closely at the microarchitectural foundations of droops. By using microbenchmarks, we show that some stall events generate significantly more voltage noise than others. Going back to the two representative benchmarks, we show that the increase in frequency of noise-critical events coincides with the increase in voltage droops.

We first show that different stall events contribute differently to the amount of voltage variation. Figure 5 illustrates this with microbenchmarks consisting of simple sequences of stall events – L1 cache misses (L1), L2 cache misses (L2), data TLB misses (TLB), branch mispredictions (BR) and hardware exceptions (EXCP). The bars show the peak-to-peak magnitude of voltage swings, caused by the different stall events, normalized to the swing magnitude of the idle loop of the operating system. We can see that branch mispredictions and exceptions cause a significantly larger voltage swing than cache misses – f.e. the difference between the branch misprediction swing and that of an L2 miss is close to 50%.

There is an intuitive explanation for the results in Figure 5. Voltage noise is an artefact of rapid changes in processor activity. Before a miss event, the processor is executing instructions, switching factors are high and current consumption is relatively large. A miss event throttles execution, but to a varying degree – the more severe the miss event, the larger portion of the chip stays idle to recover, hence the lower the current consumption and the larger the voltage swing. Out-of-order pipelines are designed to mask memory misses by continuing execution, explaining the low voltage swings for L1 and L2 misses. TLB misses require more special handling (page-walking) that could keep a larger portion of the chip idle. Finally, branch mispredictions and exceptions require flushing most of the pipeline, resulting in very low activity before the core pipeline is filled up, leading to a large voltage swing.

If we profile the two benchmarks in the running case study, we can further confirm the noise-criticality of some stall events. In order to demonstrate that, Figure 6 shows measured miss ratios for `435.gromacs` and `473.astar`. All data in the figure are relative to the case with no optimizations (O0) and optimization aggressiveness grows to the right. For both benchmarks, higher levels of optimization lead to higher branch misprediction and TLB miss ratios (Figures 6a-6b).

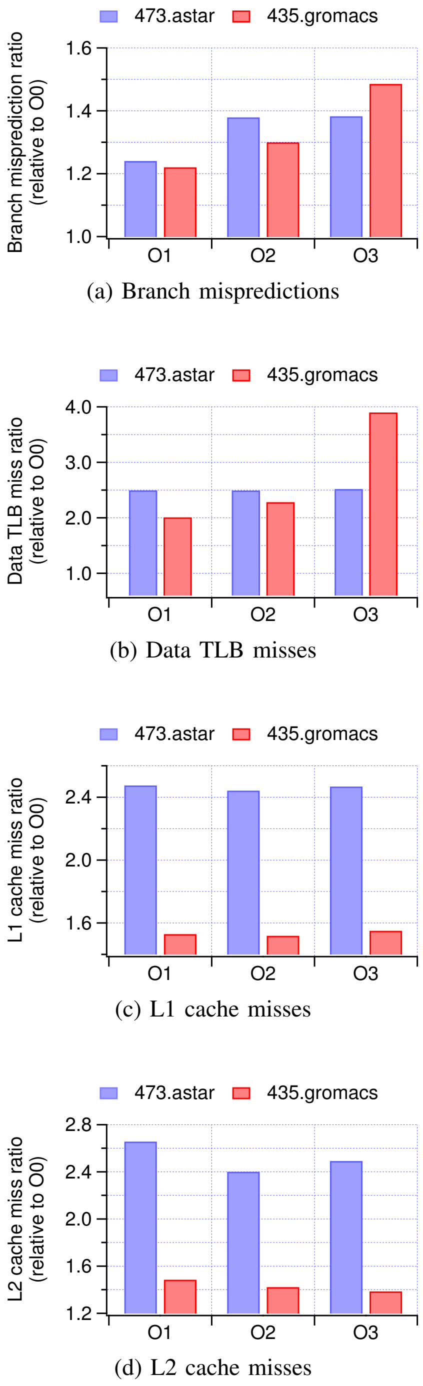


Fig. 6: Performance counter data for `473.astar` and `435.gromacs` under different optimization levels.

Both metrics are eventually higher at O3 for `435.gromacs`, which also exhibits a large increase in voltage droops between O0 and O3. On the other hand, both L1 and L2 miss ratios increase significantly more for `473.astar` (Figures 6c-6d), without a corresponding increase in noise activity.

These results strengthen the hypothesis that performance-critical and noise-critical stall events are not necessarily the same. In terms of code optimization, this suggests that optimizations targeting cache behavior (f.e. software prefetching) are unlikely to also reduce voltage noise. Such differences inspire future work in finding the optimal set of code transformations for a resilient architecture.

V. Conclusion

We have shown that higher levels of optimization amplify the amount of voltage noise that workloads exhibit. In the context of resilient architectures, the larger amount of voltage noise can lead to a net performance degradation.

For a typical-case design, these results add voltage noise behavior as yet another variable in the already complex compiler problem of picking an optimal set of code optimizations. If such a voltage-noise-aware compiler is to be created, it could address recovery-related performance bottlenecks by either: (i) statically eliminating the noise-critical stall events identified earlier; (ii) dynamically adjusting degrees of optimization (as demonstrated in prior work [7]). In either case, the considerable influence of compiler optimizations on noise suggests that designing a resilient architecture requires careful collaboration between the layers of the computing stack.

Acknowledgements

The material reported in this paper is based partly upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-13-C-0022.

References

[1] N. James *et al.*, “Comparison of split-versus connected-core supplies in the POWER6 microprocessor,” in *ISSCC*, February 2007.

[2] D. Ernst *et al.*, “Razor: A low-power pipeline based on circuit-level timing speculation,” in *MICRO*, 2003.

[3] C. Lefurgy *et al.*, “Active management of timing guardband to save energy in POWER7,” in *MICRO*, 2011.

[4] K. Bowman *et al.*, “A 45 nm resilient microprocessor core for dynamic variation tolerance,” *Solid-State Circuits, IEEE Journal of*, 2011.

[5] V. Reddi *et al.*, “Voltage smoothing: Characterizing and mitigating voltage noise in production processors via software-guided thread scheduling,” in *MICRO*, 2010.

[6] “Intel VTune,” http://software.intel.com/en-us/intel-vtune/.

[7] V. Reddi *et al.*, “Software-assisted hardware reliability: abstracting circuit-level challenges to the software stack,” in *DAC*, 2009.