# Performance Boosting under Reliability and Power Constraints

Youngtaek Kim    Lizy Kurian John
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA
young.kim@utexas.edu    ljohn@ece.utexas.edu

Indrani Paul    Srilatha Manne    Michael Schulte
AMD Research
Advanced Micro Devices, Inc.
Austin, TX, USA
{indrani.paul, srilatha.manne, michael.schulte}@amd.com

***Abstract*—Voltage droops resulting from inductive noise are common in state-of-the-art processors. Many of the techniques used to reduce energy consumption -- clock gating, power gating, process shrinks, and voltage reduction -- lead to increased voltage droops or increased sensitivity to voltage variations. Designers use voltage guardbands to minimize errors due to voltage fluctuations and inductive noise; however, this leads to lower performance because the voltage and frequency points are set to deal with voltage droops from a worst-case benchmark or stressmark. Although most applications do not approach the voltage droop caused by the stressmark, there is no mechanism to guarantee correct operation outside the tested range.**

**In this paper, we examine floating-point issue throttling (FP throttling), a hardware technique that reduces worst-case voltage droop. By lowering the issue rate in the FP scheduler, the processor can significantly reduce the maximum voltage droop in the system. We show the impact of FP throttling on voltage droop, and translate this reduction in voltage droop to an increase in operating frequency (and hence increased performance) because an additional guardband is no longer required to guard against droops resulting from heavy FP usage.**

**We then examine the impact of FP throttling and guardband reduction on the SPEC CPU2006 benchmarks and show that some benchmarks benefit from the frequency improvements with FP throttling while others suffer due to reduced FP throughput.**

**Finally, we present two techniques to determine dynamically when to trade FP throughput for reduced voltage margin and increased frequency, and show performance improvements of up to 15% for CINT2006 benchmarks and up to 8% for CFP2006 benchmarks. Our studies are done on hardware in which FP units generate the worst-case voltage droop. The technique can be modified for architectures in which other units cause the worst droop.**

***Keywords— Low Power; Performance and Power Optimizations; Reliability management; Di/Dt; Voltage Guardband***

## I. Introduction

Processor designers use a voltage guardband (a.k.a. voltage margins) to guarantee correct operation under worst-case conditions. The guardbands protect against process variations, system power supply variations, and workload-induced voltage droops. These margins are set conservatively, and are on the order of 15% to 20% of the supply voltage [6]. However, standard applications running under normal conditions do not exhibit voltage variations anywhere close to the worst-case margins [10]. Guarding against worst-case scenarios can significantly or disproportionally decrease performance; for instance, according to [16], a 20% voltage margin translates into a 33% frequency loss.

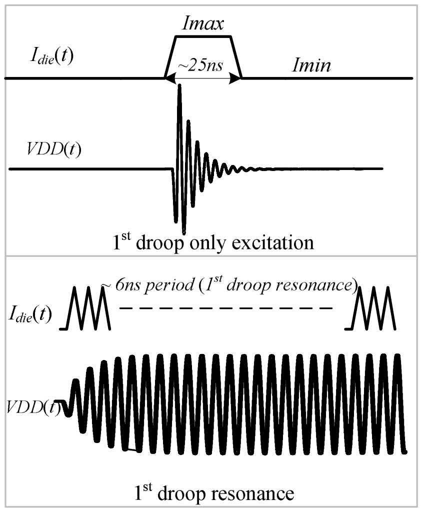


**Figure 1: Example of stressmarks to generate 1st droop excitation and 1st droop resonance.**

Worst-case voltage fluctuations occur when a high-power unit in the processor operates intermittently. These droops may be single-event or resonating droops, as illustrated in Figure 1. High di/dt stress occurs when a region of high-power instructions is followed by a region of low-power instructions [5][7][8][9][10]; if these instruction sequences repeat periodically, it may cause resonating droops. In general, resonant droops are more likely to induce failures than single-event droops because they are larger and repeat periodically.

The worst-case guardband of a system typically is determined using di/dt stressmarks, which are special programs designed to induce large di/dt voltage droops [5][7][8][9][10]. These stressmarks contain operations that utilize the high-power-consuming functional units/pipelines. If a workload does not have operations executing on the critical pipelines, then the system can be run with a lower-voltage guardband, which can be translated into a higher operating frequency.

In this paper, we investigate the use of a hardware-based throttling mechanism that limits the maximum issue rate of the floating-point (FP) pipeline. We conducted our studies with measurements on state-of-the-art x86 hardware in which FP/SSE instructions are the primary causes of droops, which explains our choice of FP units (FPUs) for throttling. The technique can be modified for architectures in which other units cause the worst droop. By enabling FP throttling, we guarantee lower workload-induced voltage droops, which enables the processor to operate with a smaller voltage guardband. We translate the reduced voltage guardband into increased operating frequency.

Our results show a significant performance improvement for non-FP-intensive programs, but some performance loss for FP-intensive programs due to restrictions on FP issue rate. In response to these findings, we present two algorithms to manage FP throttling dynamically and adjust the operating frequency to trade frequency for FP throughput to improve the performance of both FP-intensive and non-FP-intensive programs.

Figure 2 shows the general workflow of dynamic voltage guardband management introduced in this paper. Frequency sweeps using various di/dt stressmarks identify voltage margins and critical paths to determine the amount of frequency boost and microarchitecture units throttled. An algorithm that dynamically controls the throttling and boost mechanisms can improve both performance and energy.

The contributions of this paper are:

- analysis of an FP throttling mechanism on a state-of-the-art x86 processor,
- quantification of frequency boost, performance, and energy-delay product benefits made possible by FP throttling,
- analysis of the impact of FP throttling with multi-core execution, and
- new algorithms to manage FP throttling dynamically and an analysis of their benefits.

The rest of the paper presents our research. Section 2 discusses stressmarks and FP throttling. Section 3 describes our methodology for translating a savings in voltage margin into a frequency boost. Section 4 presents the dynamic FP throttling scheme. Sections 5 and 6 present our experimental methodology and results, respectively. Section 7 details the previous work in this area, and Section 8 concludes the paper.

## II. Workload-Induced Voltage Margins

### A. *Di/dt Stressmarks*

Specialized micro-benchmarks known as di/dt stressmarks are used to generate worst-case workload-induced voltage droops [7][8][9][10]. We use stressmarks as described in [10] to determine the voltage guardband necessary during normal operation and when

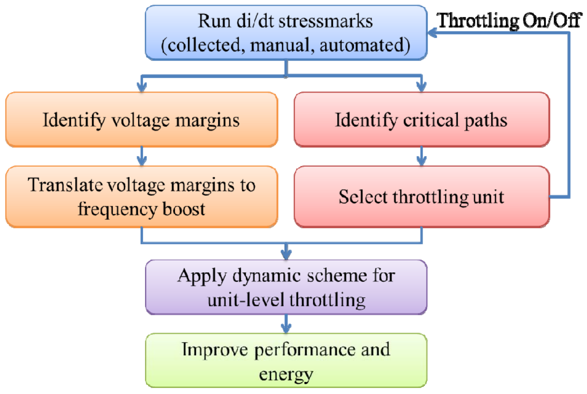

Run di/dt stressmarks (collected, manual, automated)
Throttling On/Off
Identify voltage margins
Identify critical paths
Translate voltage margins to frequency boost
Select throttling unit
Apply dynamic scheme for unit-level throttling
Improve performance and energy

**Figure 2. Dynamic voltage guardband management.**

FP throttling is enabled. We utilize three stressmarks: a manual stressmark (MS), a resonant manual stressmark (RMS) and a stressmark generated by an automatic framework (A-FpTh).

MS is a stressmark that contains both 1st droop excitation and 1st droop resonance. Because it is not physically constructed to operate at a particular frequency, it is less sensitive to FP throttling. RMS is a stressmark specifically developed to induce 1st droop resonance, and it is tuned for the baseline operating frequency of 3 GHz. As discussed in [10], 1st droop resonance is highly sensitive to operating frequency, and RMS needs to be re-tuned every time operating frequencies change; MS is more tolerant of frequency changes. We use MS and RMS to analyze workload-induced voltage droops with and without FP throttling enabled. In addition, we use an automatic stressmark-generation tool with FP throttling enabled to generate a new stressmark (A-FpTh). The A-FpTh stressmark utilizes the integer (INT) pipeline to compensate for the FP pipeline throughput restrictions. This stressmark produces large voltage droops even when FP throttling is enabled.

In the next section, we show how MS, RMS, and A-FpTh, are used to determine the additional frequency boost possible with FP throttling.

### B. *Floating-point Activity*

We use the AMD processor code-named "Orochi" [3] as our experimental system. That processor has four "Bulldozer" modules, and each module consists of two cores. Each core contains an integer scheduler and an integer execution unit, and the FP scheduler and execution unit are shared between the module's two cores. The FPU has a maximum issue rate of four instructions per cycle (IPC) and the FP instructions can come from one or both cores. The FP scheduler handles most FP operations (FP loads go through the integer scheduler) and all SSE operations.

Given that FP and SSE operations are among the highest-power operations on this hardware, the high-power region of the stressmark contains a large percentage of these operations. Figure 3 shows the voltage droop and average power resulting from 1st droop resonant stressmarks that execute and commit four IPC during the high-power region of the stressmark. Each stressmark executes one (RMS-1FP), two (RMS-2FP), three (RMS-3FP), or four (RMS-4FP) FP pipeline operations per cycle, with the remaining operations coming from the integer pipeline. These benchmarks were created by modifying the RMS stressmark; RMS-4FP is the same as the RMS stressmark.

Power and voltage-droop values are shown relative to the RMS-4FP case. For this experiment, the second core in each module is inactive and the active core has full use of the FPU. Figure 3 shows the droop increases with the issue rate of FP operations, indicating that the FP pipeline issue rate is critical for inducing the largest workload-dependent voltage droop. This is because the power in the high-power region is correlated to the number of high-power FP operations executed per cycle. The greater the power in the high-power region of the stressmark, the larger the difference between the high- and low-power regions, and the larger the resulting voltage droop.

### C. *Floating-point Throttling*

As shown in Figure 3, the number of FP operations executed each cycle has a large impact on the voltage droop. If we can limit the maximum number of operations each cycle using the FP pipeline, we can also limit the maximum workload-induced voltage droop. The Bulldozer module has a hardware mechanism, known as FP throttling, to limit the number of FP operations issued per cycle. We configured it to restrict the maximum issue width from four to two, meaning that at most two FP pipeline operations can execute per cycle. Furthermore, for some FP operations that are restricted to certain execution pipes, at most one operation can take place on these pipes every four cycles.

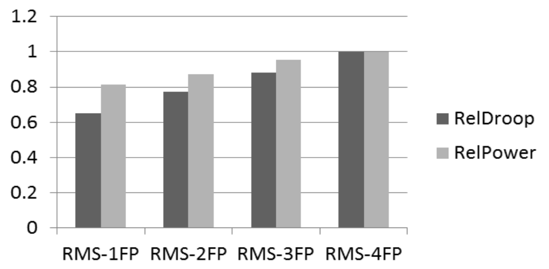


**Figure 3: Relative voltage droop and power with varying issue rate of FP operations.**

The FP throttling mechanism limits only the FP issue rate, and not the fetch or decode rate.

With FP throttling enabled, the worst-case droop for the stressmarks is reduced by 15% to 35%. The RMS stressmark uses the FP pipeline exclusively for the high-power region of the stressmark and FP throttling effectively cut its issue rate by at least one-half. This limits the power during the high-power region, and also changes the execution time of the high-power region so the stressmark no longer hits the resonance frequency of the system. Hence, the droops are significantly smaller than the resonance-induced droops prior to throttling.

## III. Determining Voltage Margins

To determine the potential frequency boost resulting from FP throttling, we use the voltage at failure as described in Kim et al. [10]. The voltage at failure determines the headroom available between the guaranteed safe point of operation and the point of failure. We determine the voltage at failure by keeping the frequency constant and reducing the supply voltage by decrements of 12.5 mV. For example, if a processor at 1.25 V and 3.0 GHz is guaranteed to run correctly but fails at 1.20 V and 3.0 GHz, then the voltage headroom between correct execution and failure is 50 mV.

To determine the maximum safe frequency of operation when workload-induced voltage droop is reduced via FP throttling, we must match the voltage headroom seen when FP throttling is disabled. To do this, we run the part with FP throttling enabled at higher frequencies in 100-MHz increments until the voltage headroom seen with FP throttling enabled and the higher frequency equals the voltage headroom seen with FP throttling disabled and the guaranteed safe frequency. Using the example in the preceding paragraph, if we see a voltage-headroom value of 50 mV when running at 3.3 GHz, then the frequency boost possible with FP throttling enabled is 300 MHz.

We implemented this methodology using the stressmarks MS, RMS, and A-FpTh with four threads and FP throttling enabled. As noted in [10], running one thread per module (four threads) generates the worst-case voltage droop due to the shared resources on a Bulldozer module. An additional complication occurs for resonant stressmarks as the frequency is increased. The resonant stressmarks (RMS) are tuned to the operating frequency of the part, so we need to re-tune the stressmark for every new frequency tested. We re-tuned RMS by removing instructions from the high- and low-power regions as the frequency increased. With A-FpTh, we generated a new automated stressmark for each new frequency.

Table 1 shows the frequency boost achievable with each stressmark. As noted earlier, because it is optimized for a throttled FPU, A-FpTh achieves the largest voltage droop with FP throttling and also determines the maximum safe frequency boost available with FP throttling: 400 MHz.

These results show two insights. First, the resonant stressmark must be tuned to the new frequencies. Second, we cannot rely blindly on existing stressmarks if we make notable changes to the FP issue rate, as is done with FP throttling. New stressmarks are needed to deal with the new FP issue rate. We use the frequency boost resulting from A-FpTh (400 MHz) to run the benchmarks at higher frequencies with FP throttling enabled.

**Table 1: Frequency boost possible with FP throttling.**

| Stressmark | Frequency Boost |
|---|---|
| MS | > 600 MHz |
| RMS | 600 MHz |
| A-FpTh | 400 MHz |

## IV. Dynamic FP Throttling

The frequency boost possible with FP throttling is beneficial for programs that do not contain a large number of operations that go through the FP pipeline. For FP- or SSE-intensive programs, however, the IPC loss resulting from reducing the FP pipeline issue rate from four to one or two IPC (depending on the pipeline on which they execute) may offset any performance benefits from increased frequency. As shown in Section 6, FP throttling can cause significant performance loss for some benchmarks; therefore, we also need a mechanism to enable and disable FP throttling dynamically. Such a mechanism would provide a frequency boost for programs with low FP pipeline usage and disable FP throttling for programs with high FP pipeline usage.

We want the dynamic FP throttling mechanism to be implemented in hardware so decisions can be made rapidly without interrupting the rest of the system. Unfortunately, such a mechanism is not available in current hardware, so we relied on performance counters and OS control to analyze our algorithms. However, the algorithms we developed can be easily implemented in future hardware.

We implemented the dynamic FP throttling mechanism using performance counters to determine the rate of ops being issued from the FP pipeline (Algorithm 1). We sample the counter every 10 ms. If the average FP pipeline issue rate exceeds a given threshold, then we disable FP throttling. Otherwise, we enable FP throttling and boost the

**Algorithm 1**: Dynamic scheme (Dyn) using FP_IPC

```
Given: FP_IPC_THRESHOLD, CONSECUTIVE,
       DECISION_INTERVAL, P0, and PB
 1: while TRUE do
 2:   for each core, C_i do
 3:     get fp_ipc; /* FP-IPC */
 4:     if (fp_ipc > FP_ IPC_THRESHOLD) then
 5:       if FP_throttling.enabled then
 6:         Frequency.adjust(P0);
 7:         FP_throttling.Off()
 8:       end if
 9:       fp_ipc_consecutive = 0;
10:     else /* fp_ipc <= FP_ IPC_THRESHOLD */
11:       if !FP_throttling.enabled then
12:         if (fp_ipc_consecutive > CONSECUTIVE) then
13:           FP_throttling.On();
14:           Frequency.adjust(PB);
15:         end if
16:       end if
17:       fp_ipc_consecutive = fp_ipc_consecutive + 1;
18:     end if
19:   end for
20:   Sleep.time(DECISION_INTERVAL);
21: end while
```

frequency. This simple mechanism makes a number of assumptions. First, it assumes that past behavior is indicative of future behavior. Second, it tracks FP issue rate at a fairly coarse level, assuming that FP activity does not change rapidly. Third, it does not consider that average values do not show the distribution of events and can hide large variances in the FP issue rate.

We chose this algorithm because it is simple to implement and not too intrusive at a 10-ms sampling rate. There is an overhead with sampling performance counters too frequently and with enabling and disabling throttling. Therefore, although programs might show fine-grained regions of high and low FP pipeline usage, our mechanism may not take advantage of them because of the overhead required to detect and manage these events. In general, we are trying to find reasonably large regions or phases of the program, and the algorithm chosen works well given our requirements. The algorithm could be expanded by collecting more details, such as determining the rate of change in the number of FP pipeline instructions issued or collecting data based on architectural events such as instruction stream misses or ITLB misses that indicate changes in program behavior. Although these are interesting ideas, they are beyond the scope of this paper, in which we introduce the general concept of dynamic voltage guardband management.

## V. Experimental Set-up

We use the AMD Orochi [3] processor with four Bulldozer modules as our experimental system. Each Bulldozer module contains two cores that share the front end, FPU, and L2 cache. We use SPEC CPU2006 benchmarks for our analysis. These benchmarks vary substantially in their use of FP and SSE instructions. SPEC CPU2006 benchmarks are compiled and highly optimized with GCC 4.6.2 and gfortran 4.6.2, which support the latest SSE instructions in Bulldozer. All the benchmarks and the stressmarks are run on Red Hat Enterprise Linux 6.

Performance is measured as total execution time from SPEC tools, and power is profiled using a National Instrument's data acquisition (DAQ) card (NI PCIe-6353), whose sampling rate is up to 1.2 million samples per second. A differential cable transfers multiple signals from the power supply lines on the motherboard to the DAQ card in the PC. Voltage droop is captured through a high-speed differential probe and a high-bandwidth oscilloscope.

## VI. Results

We ran all SPEC CPU2006 benchmarks with and without FP throttling. The baseline case disables FP throttling and uses a default 3-GHz frequency. With FP throttling enabled, we assume a frequency boost of 400 MHz based on our analysis in Section 3. As noted earlier, boosting frequency with FP throttling can help some benchmarks but hurt others. For benchmarks that do not utilize the FP pipe, we expect a maximum performance benefit of approximately 13.3% (3.4 GHz instead of 3.0 GHz) if the application is highly CPU-bound (i.e., highly sensitive to core performance). On the other hand, if the application is memory-bound, then we expect to see less improvement from a frequency boost. For FP- and/or SSE-intensive applications, FP throttling with frequency boosting may help or hurt performance. Overall performance will degrade if the IPC loss resulting from reduced FP bandwidth is greater than the performance benefit due to the increased frequency.

### A. *Performance with FP Throttling*

Figure 4 shows the performance results with FP throttling enabled. All results are gathered while running one thread. Performance is measured as run-time and is shown relative to the baseline case (`Base-3G`) with no FP throttling and no frequency boost (frequency of 3.0 GHz). For `St-FpThr-3G`, FP throttling is enabled but the processor is still operating at 3 GHz to show the impact of FP throttling on IPC. `St-FpThr-3.4G` represents results with FP throttling enabled and a 400-MHz frequency boost. Both `St-FpThr-3G` and `St-FpThr-3.4G` use a static scheme in which FP throttling is always enabled. `Dyn-FpThr-3.4G` is a dynamic scheme that decides on the fly whether to enable FP throttling and frequency boosting. The dynamic scheme recognizes which applications benefit from FP throttling in addition to adjusting for phase behavior within an application. The dynamic scheme is analyzed in more detail in Section 6.C.

FP throttling without boosting does not affect CINT2006 benchmarks with the exception of a slight performance loss for *omnetpp* and *xalancbmk*. However, the performance loss on CFP2006 benchmarks is significant: up to 38% for *calculix*. These results are not surprising given the nature of the benchmarks. With the frequency boost possible with FP throttling, we see a performance increase of up to 15% in the CINT2006 benchmarks. The FP benchmarks also improve relative to the case of FP throttling with no frequency boosting (`St-FpThr-3G`), but many of them suffer relative to the baseline case.

There are some interesting points to note about the results. First, the performance improvement on some CINT2006 benchmarks (*perl*, *bzip2*, *sjeng*) is greater than the expected maximum of ~13.3% given the frequency boost. The Bulldozer module contains complex, out-of-order cores, and the increase in core frequency changes pipeline behavior. The boosted frequency is for the Bulldozer cores only, but the northbridge and memory continue to operate at the same frequency; hence, we are increasing the operating frequency as well as creating small changes in application IPC. We cannot predict whether

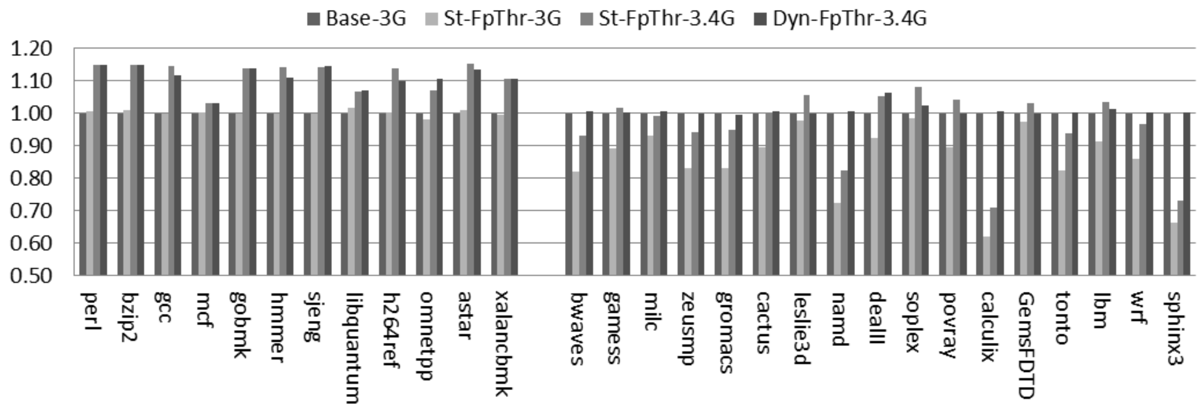


**Figure 4: Relative performance impact of FP throttling with and without frequency boost.**

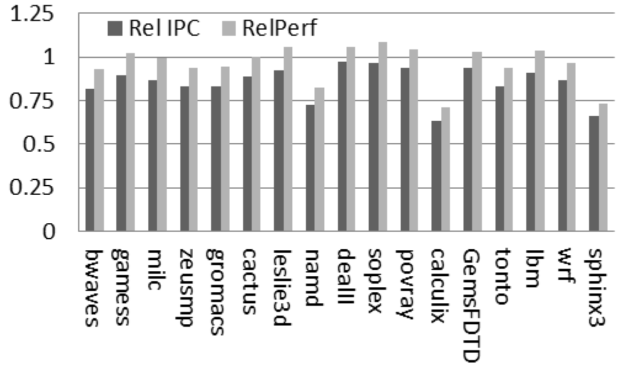


**Figure 5: IPC and performance for St-FpThr-3.4G relative to Base-3G.**

IPC will increase or decrease with a frequency boost. We measured IPC when running the CINT2006 benchmarks, and noticed small improvements in IPC for the cases when the performance exceeds 13.3%.

The second interesting aspect of the results is that some CFP2006 benchmarks improve with FP throttling and frequency boosting by up to 8% (*soplex*). St-FpThr-3.4G generally increases performance relative to the baseline when the performance loss resulting from FP throttling is not significant. As noted earlier, if the performance improvement resulting from a higher frequency is greater than the IPC loss from FP throttling, performance will improve. Figure 5 shows the relative IPC and performance for CFP2006 benchmarks. The data is for St-FpThr-3.4G relative to the Base-3G case. When the application's IPC is not significantly affected by FP throttling (relative IPC close to 1), then the resulting performance improves with FP throttling.

## B. *Energy Efficiency*

Figure 6 shows the energy-delay product (E*D) for all applications. The results are shown relative to the baseline case (Base-3G), and values lower than 1.0 mean better efficiency. We expect dynamic power dissipation in the core to scale linearly with frequency. However, the leakage power should not vary significantly because voltage remains constant between the baseline case and the boosted case with FP throttling enabled. The only reason leakage power might increase is due to the increase in temperature resulting from higher dynamic power dissipation.

The results show that E*D improves for all CINT2006 benchmarks. This is expected given that the performance improvements for these benchmarks more than compensate for the linear increase in power. The results for CFP2006 benchmarks are more mixed. There is significant increase in E*D for benchmarks with large performance losses (*namd*, *calculix*, *sphinx3*). However, benchmarks that improved in performance with FP throttling and a 400-MHz boost also show a small decrease in the E*D product. The results for Dyn-FpThr-3.4G will be discussed further in Section 6.C.

## C. *Dynamic Scheme*

The data in Figure 4 clearly shows the need for a dynamic scheme to determine when to trade FP throughput for a frequency increase. In this section, we discuss the methodology and metrics used to implement a dynamic FP throttling scheme.

Figures 4 and 6 also show results for the dynamic scheme (Dyn-FpThr-3.4G) implemented using performance counters to measure the FP IPC over a sampling period. The scheme is simple: If FP throttling is currently enabled and FP IPC is greater than a predetermined threshold, disable FP throttling and run at 3 GHz; if FP throttling is disabled and the FP IPC is less than the predetermined threshold for three sampling periods in a row, then enable FP throttling and boost the frequency to 3.4 GHz.

To determine the optimal threshold at which to disable FP throttling, we measured the average number of instructions per cycle executed in the FP pipeline (FP-IPC) for the base case (Base-3G). FP-IPC includes traditional FP operations and, among others, SSE operations. The FP-IPC results for all SPEC CPU2006 benchmarks are shown in Figure 7. As expected, the FP-IPC rate in the CFP2006 benchmarks is significantly higher than that of CINT2006 benchmarks.

The FP pipeline can sustain four IPC with optimized code, but FP-IPC values in Figure 7 are all significantly less than the maximum. There are many reasons for the large discrepancy between maximum and measured FP-IPC. First, each core has a commit width of four IPC, which means that there are other ops -- most likely loads and stores -- required to support a high issue rate in the FP pipe. Second, although the FPU can execute four IPC, not every execution unit can execute every operation; there is a limitation on the combinations of operations that can execute per cycle. Third, there are data dependencies that restrict the instruction-level parallelism (ILP) of the application. Hence, it is unlikely that the average issue rate will reach anywhere near the maximum sustainable issue rate. However, as seen by the results with FP throttling enabled, reducing the issue rate detrimentally affects performance by reducing the FP throughput during the critical stages of the application. We used a very conservative FP-IPC threshold of 0.1 to disable FP throttling

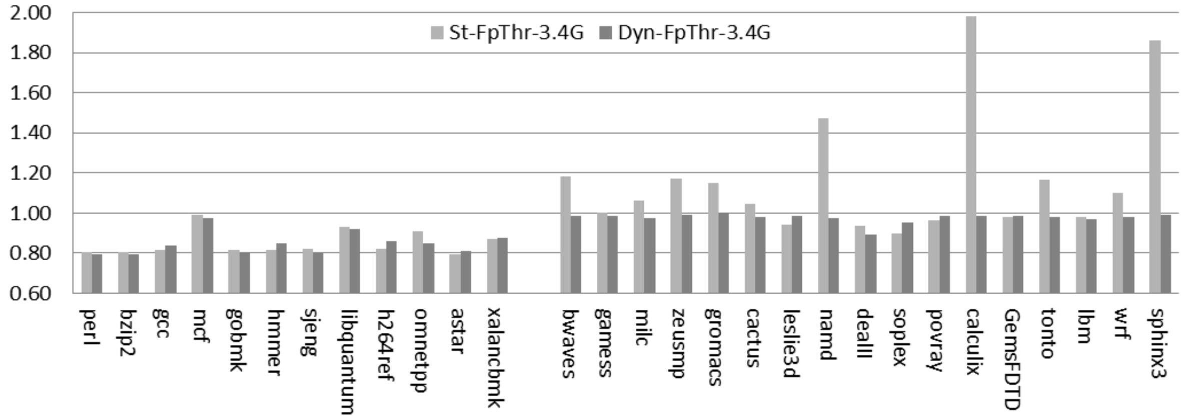


**Figure 6: Relative energy-delay product (E*D) with static and dynamic schemes.**

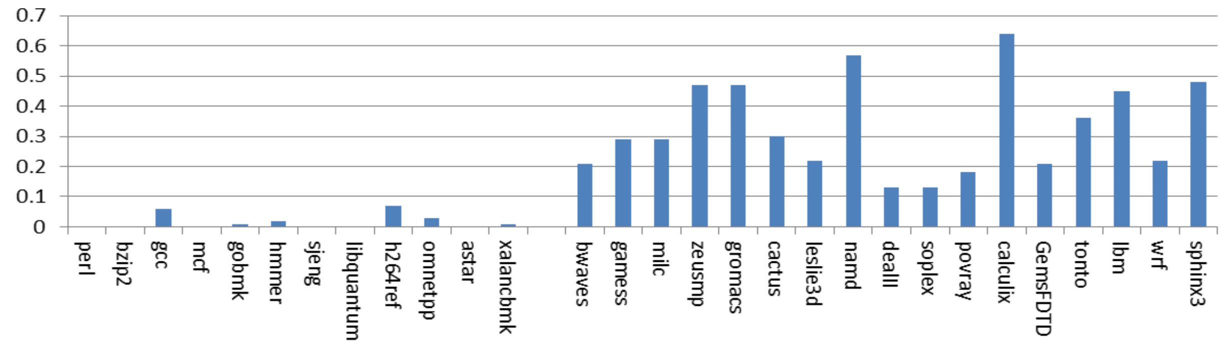


**Figure 7: Average FP-IPC for benchmarks for Base-3G case.**

aggressively when necessary. The goal is to improve all the CINT2006 cases without incurring a performance loss in the CFP2006 benchmarks.

Figure 4 shows that the dynamic scheme eliminates the performance losses in CFP2006 benchmarks while retaining most of the performance gains of the CINT2006 benchmarks. Performance losses are no longer seen for *namd*, *calculix*, and *sphinx3*. However, the dynamic scheme also reduced the benefits in some benchmarks such as *h264ref*, *povray*, and *leslie3d*. The dynamic scheme also had slightly worse E*D numbers (Figure 6) for the same benchmarks although it improves the average E*D numbers across all benchmarks.

## D. Multi-core Execution

The results so far have focused on single-threaded runs in which one thread runs on one core in a Bulldozer module. The AMD Orochi processor, however, is capable of running eight threads. Multi-core execution adds additional complexity to the problem. First, the L2 cache, the fetch unit, and the FP pipeline are shared between two cores in a Bulldozer module. This sharing can result in contention between threads, making them less capable of benefiting from a frequency boost. Second, depending on the homogeneity of the threads and how they execute, the two threads in a Bulldozer module may have differing requirements. Although each Bulldozer module can run at different frequencies, the two cores in a module must run at the same frequency.

We performed eight-threaded SPECrate runs and collected relative performance information (Figure 8). The numbers for the static and dynamic schemes are shown relative to Base-3G. As before, CINT2006 benchmarks benefit the most from FP throttling and boosting while CFP2006 benchmarks suffer the most. However, the dynamic scheme is able to detect the application phases that benefit from FP throttling and boosting to eliminate the performance loss on CFP2006 benchmarks.

There are some differences in the results from the single-thread (1T) runs. The number of CFP2006 benchmarks that benefit from FP throttling and frequency boosting is not as large as the 1T case because the FP pipeline is shared between two cores in a Bulldozer module. Although some CFP2006 benchmarks are not as sensitive to FP-IPC, the problem gets exacerbated with the shared FP pipeline; as a result, the utilization of a shared FP pipeline increases when multiple threads run in the same module.

## E. Improved Dynamic Scheme

The dynamic scheme using FP-IPC improved overall results, but there were some cases when the static scheme outperformed the dynamic scheme. The dynamic scheme is very simple and examines only one metric: FP-IPC. Thrashing between FP throttling-enabled and -disabled states will occur if the FP-IPC of the application hovers around the predetermined threshold value; this is the case for *h264ref*. In addition, an FP-IPC higher than the threshold does not always indicate that the application performance is dependent on FP throughput. For instance, a high overall IPC may indicate a large amount of ILP in the program, and FP-IPC is not as critical to performance; *dealII* and *povray* show this characteristic.

We modified the dynamic scheme based on those observations by examining not just the FP-IPC but also the ratio of FP-IPC to overall IPC (Algorithm 2). The new scheme (Dyn2-FpThr-3.4G) is shown in Figure 9 for a sub-set of interesting benchmarks from SPEC CPU2006. The static and both dynamic schemes are shown relative to Base-3G. The new dynamic scheme (Dyn2-FpThr-3.4G) improves the performance and the energy-delay product of most of the benchmarks and brings them back up to the static levels (St-FpThr-3.4G) in many cases.

Figure 10 shows the percent of time FP throttling was enabled with the two different dynamic schemes. In many cases, the improved dynamic scheme alternates between throttled and non-throttled states. In *dealII* for instance, by spending a small portion of the total run-time with throttling disabled, the Dyn2-FpThr-3.4G scheme is able to improve on the static scheme. In *h264ref* and *povray*, the Dyn2-FpThr-3.4G scheme enables FP throttling for nearly the entire run and improves performance when compared to Dyn-FpThr-3.4G. These results show that the dynamic scheme is able to detect which applications require FP throttling and which do not as well as to determine phases within an application that can take advantage of throttling.

## VII. Previous Work

A number of previous papers explored how to reduce the voltage guardband of the system to achieve better performance. The work that comes closest in terms of a hardware implementation is the work by Lefurgy et al. [11], which addressed actively monitoring and managing the voltage guardband based on the use of a critical-path monitor (CPM). The CPM monitors the critical pathways in the chip and increases the voltage guardband if the CPM detects potential errors. Although the CPM is a very effective mechanism, it requires additional hardware, monitoring mechanisms, and tuning of the CPM to detect and correct possible errors. Our technique, by contrast, is very simple to implement and manage and requires only a characterization effort to determine the frequency boost possible with FP throttling.

A number of papers dealt with mitigating voltage droops using software techniques [4][5][16]. These techniques recognize the existence of repetitive code with high di/dt transition activity and dampen or eliminate this activity through software techniques. Software mitigation of noise does not guarantee elimination of all errors. In fact, the software techniques learn from errors detected by

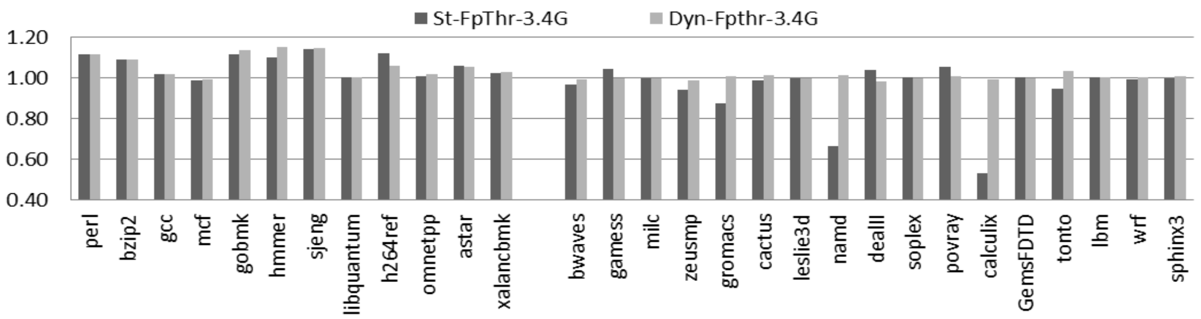


**Figure 8: Relative performance with eight-threaded execution and FP throttling.**

hardware (such as a CPM) and adjust the software only after errors are detected. Our FP throttling mechanism avoids errors altogether by suppressing the structures that generate the largest voltage droops.

Other work examined using hardware techniques to manage high droops [12][14][15]. Some of the work focused on using hardware to detect a resonant droop about to build and suppressing the droop before it reaches peak droop value, while others focused on mechanisms to dampen the difference between the high- and low-power regions by techniques such as throttling issue rates or staged activation and deactivation of clock-gated units. All these techniques address the issue of di/dt noise. However, we are the first to characterize the impact of FP throttling, translate that characterization to a frequency increase, and present results with static and dynamic schemes showing the benefits of the performance increase with FP throttling and frequency boosting.

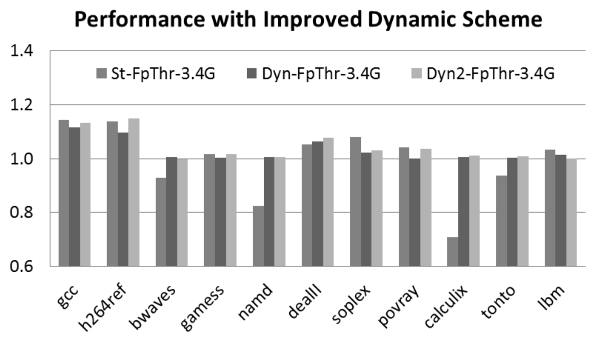
Another body of work explored detecting and mitigating errors via circuit techniques [1][2]. The research using Razor systems assumed that errors will occur and inserts redundancy in latches. Although effective, Razor requires significant new hardware and a completely different design methodology that fundamentally changes the way processors are designed. The FP-throttling technique, on the other hand, works well with existing systems in which the FPU is a large contributor to the voltage droop in the system.

Finally, there are other methods to boost processor frequency [12][17]. Frequency-boosting techniques such as AMD Turbo CORE or Intel Turbo Boost are in use in state-of-the-art systems from AMD and Intel. AMD Turbo CORE allows the chip to run at a higher frequency than that visible to software based on the availability of power headroom. Once the power headroom is depleted, the application returns to a lower-power, lower-performance DVFS state until power headroom is once again available.

AMD Turbo CORE is available on the hardware we used for our

**Algorithm 2**: Improved dynamic scheme (Dyn2) using FP_IPC and FP_RATIO

```
Given: FP_IPC_THRESHOLD, FP_RATIO_THRESHOLD,
       CONSECUTIVE, DECISION_INTERVAL, P0, and PB
 1: while TRUE do
 2:   for each core, C_i do
 3:     get fp_ipc;   /* FP-IPC */
 4:     get fp_ratio; /* ratio: FP-IPC to IPC */
 5:     if fp_ipc > FP_IPC_THRESHOLD then
 6:       if fp_ratio > FP_RATIO_THRESHOLD then
 7:         if FP_throttling.enabled then
 8:           Frequency.adjust(P0);
 9:           FP_throttling.Off()
10:         end if
11:         fp_mac_consecutive = 0;
12:       else /* fp_ratio <= FP_RATIO_THRESHOLD */
13:         if fp_ratio_consecutive > CONSECUTIVE then
14:           if !FP_throttling.enabled then
15:             FP_throttling.On();
16:             Frequency.adjust(PB);
17:           end if
18:         end if
19:         fp_ratio_consecutive = fp_ratio_consecutive + 1;
20:       end if
21:       fp_ipc_consecutive = 0;
22:     else /* fp_ipc <= FP_IPC_THRESHOLD */
23:       if fp_ipc_consecutive > CONSECUTIVE then
24:         if !FP_throttling.enabled then
25:           FP_throttling.On();
26:           Frequency.adjust(PB);
27:         end if
28:       end if
29:       fp_ipc_consecutive = fp_ipc_consecutive + 1;
30:     end if
31:   end for
32:   Sleep.time(DECISION_INTERVAL);
33: end while
```



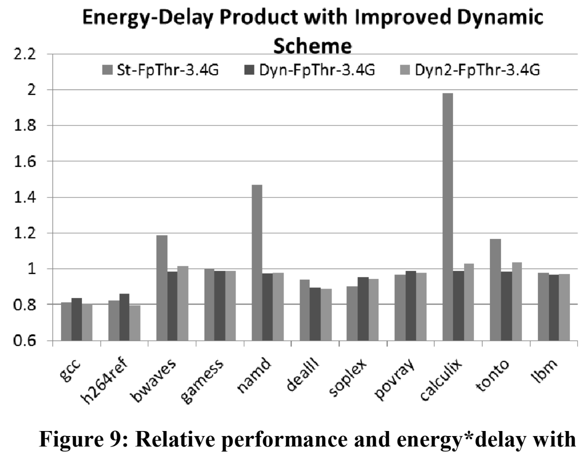


**Figure 9: Relative performance and energy*delay with improved dynamic scheme (Dyn2-FpThr-3.4G).**

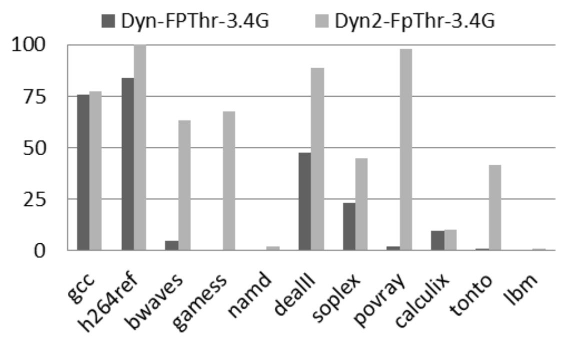


**Figure 10: Percent time spent with FP throttling enabled.**

experiments. There is one major difference between the frequency boost possible with AMD Turbo CORE and that resulting from FP throttling. With FP throttling, the processor can boost the frequency without an increase in voltage, resulting in a linear increase in power for a potentially linear increase in performance. AMD Turbo CORE, on the other hand, requires an increase in both frequency and voltage, resulting in a cubic increase in power. Hence, it is not as efficient a method for boosting performance. However, it does not incur any IPC loss due to FP throttling. AMD Turbo CORE was disabled for the analysis presented in this paper to study the impact of FP throttling; however, combining the two techniques offers interesting research opportunities in the future.

## VIII. Conclusions

This paper addressed decreasing voltage guardbands to increase performance in existing processors. We used a hardware technique to limit the issue rate in the floating-point scheduler that resulted in a reduction in the worst-case voltage droop. We used a number of existing and newly generated stressmarks to translate the reduction in voltage guardband into a frequency boost and showed the impact this has on the SPEC CPU2006 benchmarks. Based on our evaluation, we developed dynamic schemes to detect when to trade FP throughput for a frequency increase. We implemented two dynamic schemes in firmware and showed they can improve the performance of several benchmarks without sacrificing the performance of others.

## Acknowledgments

Lizy Kurian John and Youngtaek Kim are supported partially by NSF grants 1117895 and 1218474 and by AMD unrestricted research funding. The views and opinions in this paper do not represent those of the National Science Foundation. Youngtaek Kim completed the work for this paper while he was a PhD student at The University of Texas at Austin and a graduate intern at Advanced Micro Devices, Inc.

## References

[1] D. Ernst, N.S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Proceedings of the IEEE Micro Symposium*, 2003.

[2] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. Kim, and K. Flautner, "RAZOR: circuit-level correction of timing errors for low power operation," *IEEE Micro*, 2004.

[3] T. Fischer et al., "Design solutions for the Bulldozer 32nm SOI 2-core processor module in an 8-core CPU," in *Proceedings of IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2011.

[4] M. Gupta, K. Rangan, M. Smith, G. Wei, and D. Brooks, "Towards a software approach to mitigate voltage emergencies," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

[5] K. Hazelwood and D. Brooks, "Eliminating voltage emergencies via microarchitectural voltage control feedback and dynamic optimization," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 2004.

[6] N. James, P. Restle, J. Friedrich, B. Houtt, and B. McCredie, "Comparison of split-versus connected-core supplies in the POWER6 microprocessor," in *Proceedings of the International Solid State and Circuits Conference (ISSCC)*, 2007.

[7] R. Joseph, D. Brooks, and M. Martonosi, "Control techniques to eliminate voltage emergencies in high performance processors," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-9)*, 2003.

[8] A. Joshi, L. Eeckhout, L. John, and C. Isen, "Automated microprocessor stressmark generation," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA-14)*, 2008.

[9] M. Ketkar and E. Chiprout, "A microarchitecture-based framework for pre- and post-silicon power delivery analysis," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42)*, 2009.

[10] Y. Kim, L. John, S. Pant, S. Manne, and M. Schulte, W. Bircher, and M. Govindan, "AUDIT: Stress-Testing the Automatic Way," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*, 2012.

[11] C. Lefurgy, A. Drake, M. Floyd, M. Allen-Ware, B. Brock, J. Tiemo, and J. Carter, "Active management of timing guardband to save energy in POWER7," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)*, 2011.

[12] S. Nussbaum, "AMD Trinity APU," in *Hotchips*, 2012.

[13] M. Pant, P. Pant, D. Willis, and V. Tiwari, "Architectural solution for the inductive noise problem due to clock-gating," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 1999.

[14] M. Powell and T. Vijaykumar, "Exploiting resonant behavior to reduce inductive noise," in *Proceedings of the 31st International Symposium on Computer Architecture (ISCA-31)*, 2004.

[15] M. Powell and T. Vijaykumar, "Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise," in *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 2003.

[16] V. Reddi, S. Kanev, W. Kim, S. Campanoni, M. Smith, G.Y. Wei, and D. Brooks, "Voltage Smoothing: Characterizing and Mitigating Voltage Noise in Production Processors via Software-guided Thread Scheduling," in *Proceedings of the International Symposium on Microarchitecture (MICRO-43)*, 2010.

[17] V. Reddi, M. Gupta, K. Rangan, S. Campanoni, G. Holloway, M. Smith, G.Y. Wei, and D. Brooks, "Voltage Noise: Why It's Bad, and What To Do About It," in *Proceedings of IEEE Workshop on Silicon Errors in Logic – System Effects (SELSE)*, March, 2009.

[18] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weisman, "Power management architectures of the Intel microarchitecture code-named Sandy Bridge," *IEEE Micro*, 2012.