Tribeca: Design for PVT Variations with Local Recovery and Fine-grained Adaptation

Meeta S. Gupta, Jude A. Rivers†, Pradip Bose†, Gu-Yeon Wei and David Brooks

School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138
†IBM T.J Watson Research Center Yorktown Heights, NY 10598
{meeta,guyeon,dbrooks}@eecs.harvard.edu, †{jarivers,pbose}@us.ibm.com

Abstract

With continued advances in CMOS technology, parameter variations are emerging as a major design challenge. Irregularities during the fabrication of a microprocessor and variations of voltage and temperature during its operation widen worst-case timing margins of the design—degrading performance significantly. Because runtime variations like supply voltage droops and temperature fluctuations depend on the activity signature of the processor's workload, there are several opportunities to improve performance by dynamically adapting margins. This paper explores the power-performance efficiency gains that result from designing for typical conditions while dynamically tuning frequency and voltage to accommodate the runtime behavior of workloads. Such a design depends on a fail-safe mechanism that allows it to protect against margin violations during adaptation; we evaluate several such mechanisms, and we propose a local recovery scheme that exploits spatial variation among the units of the processor. While a processor designed for worst-case conditions might only be capable of a frequency that is 75% of an ideal processor with no parameter variations, we show that a fine-grained global frequency tuning mechanism improves power-performance efficiency ($BIPS^3/W$) by 40% while operating at 91% of an ideal processor's frequency. Moreover, a per-unit voltage tuning mechanism aims to reduce the effect of within-die spatial variations to provide a 55% increase in power-performance efficiency. The benefits reported are clearly substantial in light of the $<$1% area overhead relative to existing global recovery mechanisms.

Categories and Subject Descriptors C.4 [*Performance of Systems*]: Reliability,availability and serviceability

General Terms Performance, Reliability

1. Introduction

Continued advancement of CMOS technologies provides the well-known benefits of device scaling. However, as feature sizes shrink and chip designers attempt to reduce supply voltage to meet power targets in large multi-core systems, parameter variations are becoming a serious problem. Parameter variations can be broadly classified into device variations incurred due to imperfections in the manufacturing process and environmental variations due to fluctuations in on-die temperature and supply voltage. Collectively, these PVT variations greatly impact the speed of circuits in a chip; delay paths may slow down or speed up due to these variations. The traditional approach to deal with parameter variations has been to over-design the processor based on the most pessimistic operating conditions to allow for worst-case variations.

As the gap between nominal and worst-case operating conditions in modern microprocessor designs grow, the performance impact of worst-case design are too large to ignore. Recognizing the large performance loss in such a design style, researchers have begun to propose architecture-level solutions that address worst-case conditions. However, almost all proposed solutions to date have focused on a single source of parameter variations at a time: temperature [28], voltage [10, 15, 22], or process [18, 25, 30]. Implicitly, many of these studies assume that the costs of individual sources of parameter variations and the benefits of the proposed schemes are orthogonal. However, there are many complex interactions between parameter variations that highly depend on the underlying microarchitecture, workloads, and operating conditions.

This paper seeks to understand these complex interactions and to propose new microarchitectural solutions cognizant of the combined characteristics of P, V, and T variations. We observe that simply adding up the margins from each source of variation can lead to an excessively conservative design. Moreover, these variations differ significantly in temporal and spatial scales. Process variations are static in nature, while voltage and temperature variations are highly sensitive to workload behavior, albeit at very different time scales. All sources of variation impact different parts of a microprocessor die in myriad ways with complex interactions. Microarchitectural techniques designed to mitigate parameter variations must clearly account for these differing characteristics.

This paper introduces a framework called *Tribeca* which considers the combined interaction of all the three sources of variations and proposes mechanisms for designing processors for typical operating conditions in the presence of PVT variations. Tribeca makes three primary contributions to the field of handling variations:

- We provide a detailed analysis of the combined effect of P, V, and T variations and demonstrate differences from naive approaches that treat each source in isolation. We further illustrate the spatial and temporal characteristics of parameter variations by combining best-known modeling frameworks for P, V, and T with a POWER6-like microprocessor performance simulator augmented with circuit-extracted power models.
- We propose a local recovery mechanism that provides low cost detection and rollback for timing-margin violations. This local recovery mechanism exploits spatial variation among units within the processor, and low cost recovery allows aggressive setting of design margins.

MICRO'09, December 12–16, 2009, New York, NY, USA.

- We explore fine-grained dynamic adaptation of processor frequency and voltage to exploit both temporal and spatial variation in delay across the processor. We show that previously proposed static or coarsely-dynamic mechanisms cannot sufficiently handle variation when including the effects of voltage variation. We propose adaptation mechanisms that seek to maximize power-performance efficiency in the presence of parameter variations.

Our results show solutions combining local recovery with a fine-resolution dynamic adaptation mechanism can maximize performance with minimal increase in power. A processor designed for worst-case operating conditions can only operate at 75% of the frequency of an ideal processor devoid of any variations. Deploying a fine-grained global frequency tuning mechanism increases the BIPS^3/W metric by 40%, while allowing operation at 91% of the ideal processor frequency. A per-unit voltage tuning mechanism takes advantage of spatial variations across the core to provide a 54% increase in BIPS^3/W.

This paper is organized as follows. Section 2 motivates why we must consider all types variations together and highlights the benefits of designing for typical, rather than worst-case, conditions. Section 3 then presents a detailed discussion of the characteristics of the variations, and it describes the proposed Tribeca framework for dealing with them. Our modeling methodology and simulator framework are presented in Section 4. A detailed evaluation of the various mechanisms proposed is presented in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2. Parameter Variations and Worst-Case Design

Parameter variations pose many challenges for CPU designers in technologies beyond 65nm [2]. The most critical sources of parameter (or PVT) variations are systematic and random variations in process characteristics across dies, supply voltage droop, and temperature variation. Parameter variations impact the worst-case delay of critical paths and, hence, directly affect performance and power. Variations also introduce spread in path delays across a chip both spatially (across different units in a chip) and temporally (as workload behavior causes voltage and temperature to fluctuate).

Designers typically account for parameter variations by imposing conservative margins that guard against *worst-case* variation characteristics to guarantee functional correctness of the system under all operating conditions. The size of these timing margins depends on how designers account for the impact of variations. Conservatively, designers may treat each source of variation independently and determine worst-case margins by simply summing the required margin for each source. This conservative approach ignores important interactions that can exacerbate variation effects. Hence, it is important to consider PVT variations in combination and not as independent entities. Section 2.1 evaluates this issue and motivates the need for design solutions that consider all sources of variation simultaneously. Section 2.2 motivates the need for solutions that accommodate the large gap between worst-case and nominal operating conditions.

2.1 Overlapping Margins

All sources of parameter variation lead to timing overhead and uncertainty, but each kind of variation has different characteristics. Process variations, which result from imperfections in the manufacturing process, are potentially large variations in device-level attributes such as the threshold voltage and gate length of transistors, with both systematic and random components. Voltage variations are closely related to workload-dependent activity and resulting current fluctuations on the order of 10s to 100s of cycles. These fluctuations cause droops in the power supply distribution network due to interaction with the parasitics of the supply network. Similarly, although at a much coarser time scale, different activity profiles also induce temperature differences across the chip.

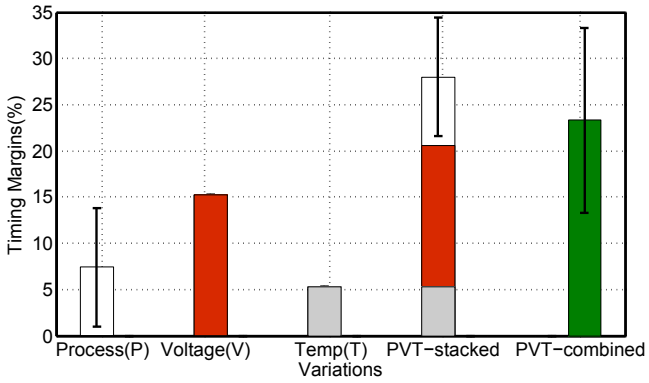


Figure 1: Impact of PVT variations on timing margins. Simple *stacking* leads to a larger mean and smaller spread in required timing margins as compared to the *combined* effects of all three.

Figure 1 illustrates the timing margins required when considering isolated and combined sources of variations. These timing margins are set by the worst-case delay path within a chip. The process parameters, gate length, threshold voltage, nominal supply voltage (V_{nom} = 1.15V), and nominal temperature (T_{nom} = 80C) are based on a 65nm technology node and ITRS specifications [13]. The first three bars (Process, Voltage, and Temperature) represent the timing margins required for each of the variations when considered in isolation. Process variation was evaluated across a batch of 100 chips by modeling and simulating both systematic and random effects. The bar represents the mean timing margin across all 100 chips. The error bars represent the maximum and minimum timing margins observed across the 100 chips. Voltage variations are evaluated by running our benchmark suite on a nominal chip with no process variation at T_{nom}. The bar represents the mean of the worst-case timing margin required for the benchmark suite considered, indicating a 15% timing margin required to accomodate voltage variations. Temperature variations are evaluated by determining the timing margins required for a chip with no process variation and operating at a temperature of 100C at V_{nom}. Temperature variations require 5% margins to ensure correctness due to worst-case temperature across the core.

Simply stacking the individual margins together (PVT-stacked) results in a simple, but conservative, approach to setting worst-case timing margins. In contrast, the figure also shows the resulting timing margins from simulating all sources of variations together. When PVT variation effects are combined (PVT-combined), the average margin required reduces, but the spread between maximum and minimum margins increases. This larger spread in margins primarily results from the interaction between voltage and process variations. Faster chips, consisting of transistors with lower threshold voltages, are less sensitive to voltage droops and can operate with tighter margins. On the other hand, the transistors with higher threshold voltages in slower chips are more sensitive to voltage droop and require larger margins. Hence, the spread between maximum and minimum margins increases. Runtime temperatures, typically being lower than applying a worst-case 100C penalty, result

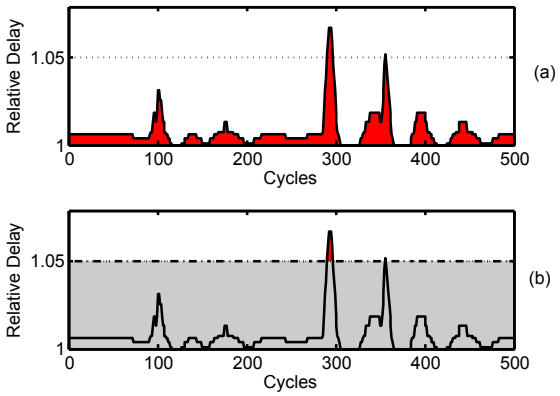


Figure 2: Example depicting performance loss incurred by providing timing margins (bottom) compared to ideal cycle-by-cycle frequency tracking (top).

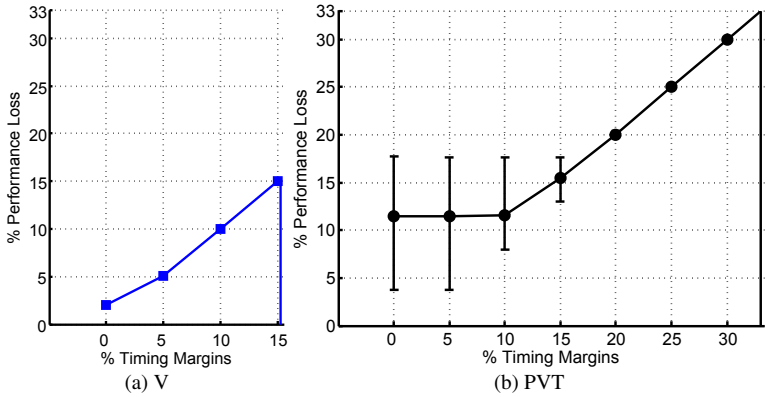


Figure 3: Need to design for the typical case. Performance loss at various timing margins is depicted for (a) voltage variations and (b) process-voltage-temperature variations.

in average margin reduction. The slowest chip, with the highest threshold voltages across the chip, exhibits lower leakage power to ameliorate thermal effects and slightly reduce the maximum required timing margin. Given that simply stacking margins misses important interactions found by considering PVT variations together, designers must address all sources of variations in combination and not as individual, orthogonal components.

2.2 Design for Typical Conditions

Conservative designs operate at the worst-case timing margin, ensuring robustness, but with performance loss due to lower processor frequency. Because worst-case conditions can be severe but infrequent, operating with conservative worst-case margins is costly. This section explores the widening gap between nominal and worst-case conditions assuming that a *fail-safe* mechanism can handle the infrequent worst-case scenarios.

Figure 2 shows a snapshot of circuit delay over 500 cycles for SPEC CPU2006 benchmark *h2ref* run on a chip with no process or temperature variations. In other words, voltage droops are solely responsible for the delay variations. There are infrequent large droops in voltage causing occasional increases in delay. To estimate the benefits of designing for the nominal case, we consider an ideal frequency tracking scenario that adjusts clock frequency according to cycle-by-cycle delay (akin to an asynchronous design). The performance loss of such a system corresponds to the area under the curve in Figure 2(top). We compare such a scenario to one that assumes a fixed timing margin of 5% and handles violations with a *fail-safe* mechanism shown in Figure 2(bottom). Margin violations incur additional performance penalties.

Figure 3 compares the performance loss of the ideal cycle-by-cycle tracking scheme (represented by 0% margins) with that of applying fixed timing margins while simulating the entire benchmark at the 65nm technology node. Figure 3(a) plots the performance loss corresponding to different timing margins in the presence of voltage variations. An ideal asynchronous design incurs 2% performance loss due to delay fluctuations. As timing margins increase up to the worst-case 15% level, performance loss is mostly due to the fixed margin as delay fluctuations are infrequent. Figure 3(b) presents a similar plot that considers the impact of PVT variations. The error bars again correspond to the spread due to process variations for 100 simulated chips. With 0% timing margins, the fastest chip exhibits a 4% loss in performance and the slowest chip incurs a much larger penalty of $\sim$17%. Applying larger fixed timing margins penalizes the fastest chips until the fastest and slowest chips suffer similar losses beyond 20%, where the margins again dictate performance loss. The plot extends to 33%, which represents a scenario that sets margins with respect to the worst-case delay across all chips. This analysis shows that the severe but infrequent nature of worst-case run-time conditions motivates design strategies that avoid running with timing margins based on worst-case conditions. Such strategies have the potential to recapture up to 20% of performance loss on average.

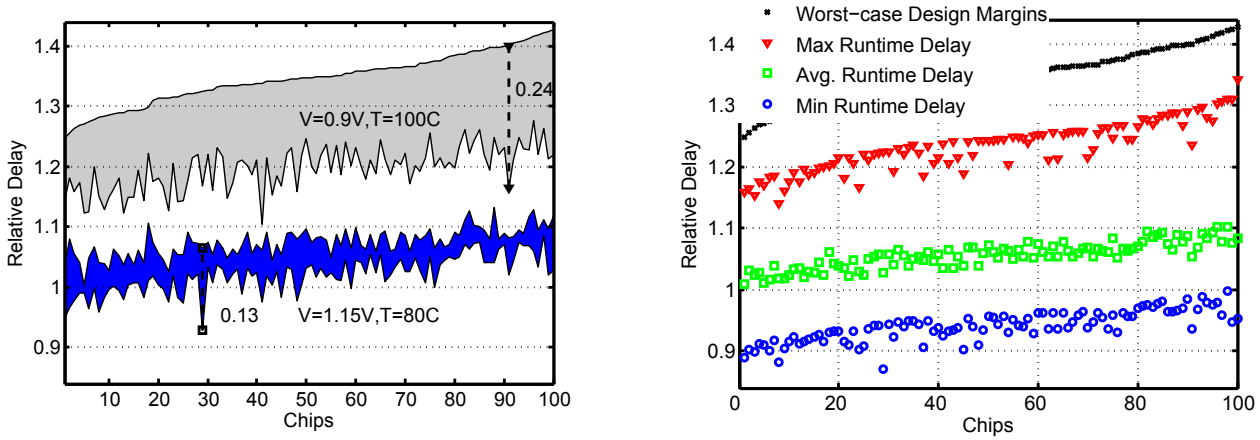
3. Tribeca: Processor Design for Typical Operating Conditions

With the growing gap between worst-case and typical-case delay in processors, it becomes essential to design for the typical case while ensuring correctness. Typical-case design can lead to timing violations when the length of a clock cycle is near the delay of the critical paths in the processor. Correct and reliable execution requires robust violation detection and a low-cost recovery mechanism. However, parameter variations lead to different behavior in different parts of the processor, suggesting that a global recovery mechanism may be wasteful. Section 3.1 gives an overview of our baseline microarchitecture, which includes a global recovery mechanism, represented by the recovery unit. Section 3.2 explores the spatial and temporal characteristics of the circuit delay for different units in the processor. Along with spatial and temporal variations, application variability emphasizes the need for adaptive mechanisms for coping with variations. Section 3.3 presents the Tribeca framework and explores the architectural support required to enable a low-cost solution that adapts processor frequency/voltage both spatially and temporally. The proposed distributed local recovery solution replaces the existing recovery unit in the baseline processor.

3.1 Baseline Microarchitecture

Our baseline microarchitecture resembles the published POWER6 processor core [17] .[1] The baseline core is modeled as shown in Figure 6(a) (excluding the shaded boxes). It consists of an instruc-

[1] The results reported in this paper do not claim to represent any real product. The POWER6 microarchitecture was chosen as it supports an on-chip recovery mechanism, which was essential to the particular ideas and attendant analysis reported in this paper.



(a) Spatial: Spread of worst-case delay between different units. Bottom: spread due to process variations alone. Top: increased spatial spread at worst-case conditions.

(b) Temporal: Spread of delay due to run-time behavior. The maximum, average, and minimum delay over all 16 benchmarks and 5 units per chip (a total of 80 points/ chip).

Figure 4: Spatial and temporal slack. Differences across processor units and between application phases the designers can exploit.

tion fetch unit (IFU), instruction dispatch unit (IDU), two fixed-point (FXU), floating-point (FPU) and two load-store (LSU) units. In addition it has a recovery unit (RU), which performs checkpoint and recovery. It contains the data representing processor state (protected by ECC), so that the state of the processor can be restored when an error condition is detected. Issue in the POWER6 core is strictly in-order (per thread), and execution pipelines include delay stages designed to relieve some of the potential performance impact of this in-order issue policy. The FXU pipeline has register file access delayed by two cycles, for example, to provide the proper load-use delay timing for cases where a load hits in the L1 cache, thereby allowing the load and its dependent use to occupy the same group of issued instructions (i.e., to issue into the execution pipelines on the same cycle). Instructions are not directly issued into the FPUs, but instead into a simple issue queue, which allows some out-of-order execution of FPU instructions, but also decouples the instruction dispatch/issue unit from the longer delays of the floating-point unit. In a similar way, some execution pipelines include delay stages after their active execution intervals to properly balance the pipelines, and to ensure that instruction writeback occurs in the architected order.

3.2 Characteristics of Variations

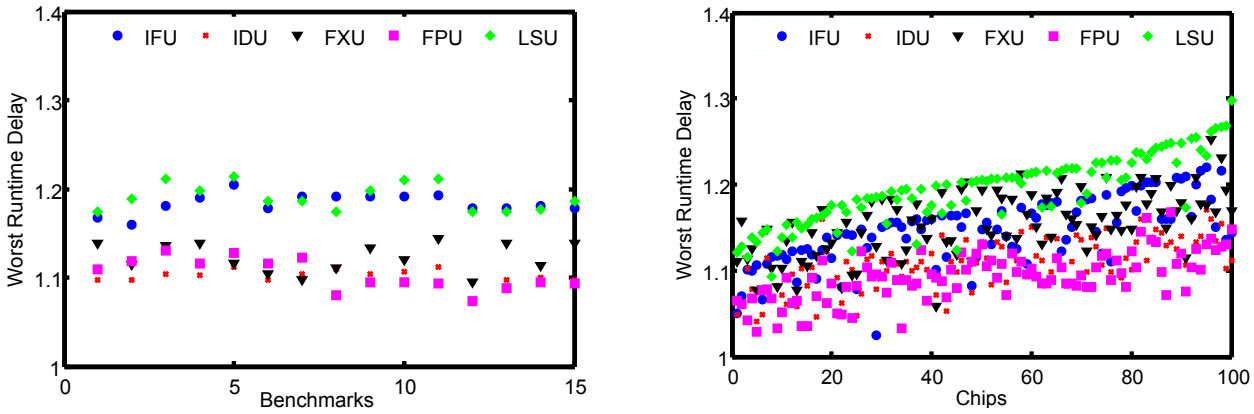
Parameter variations lead to different behavior in different parts of the core. Process variation introduces static differences between various units, amplified by run-time variations. Delay characteristics vary both temporally and spatially, across the blocks of the processor core. Traditionally, the worst-case design point (worst-case voltage droop, worst-case temperature), based on synthetic activity patterns that attempt to induce these worst-case events, determines the timing margins. However, there are substantial amounts of spatial, temporal, and application-level slack designers can exploit. This section explores the spatial and runtime characteristics of circuit delay across a processor along with the variability within and across applications. Details of the processor and the PVT modeling framework are deferred to Section 4.

Variations delay different units of a processor by different amounts. We call this *spatial slack*. Systematic variation in process parameters like threshold voltage, gate length, and oxide thickness creates differences across the core. Different software phases load different functional units to a greater or lesser degree. For example, the FXU has both high power density and high activity fluctuations, leading to larger temperature spikes and deeper voltage droops than those of the FPU.

Figure 4(a) shows the difference between the worst-case circuit delay for different units under two different operating conditions, a nominal operating condition of 1.15V at 80C and a worst-case design point of 0.9V and 100C. A maximum slack of relative delay of 13 percentage points is available under nominal operating conditions. This slack increases to 24 percentage points when the operating conditions are based on the worst-case design points (though for a different chip). Variation in process parameters changes the voltage-delay or temperature-delay relationships of circuits, widening the gap between units as conditions worsen. The variation in delay between units suggests using a recovery solution that can take advantage of the differences in delay behavior across the processor, rather than a global recovery mechanism.

The runtime behavior of workloads also exposes *temporal slack*. Figure 4(b) shows the maximum run-time delay across the benchmark suite for 100 simulated chips (generated with process variations as described in Section 4), along with the delay for worst-case design margins (V=0.9V, T=100C). For nearly every chip, there is a slack of 10 percentage points between the worst-case delay and the maximum runtime delay seen in benchmarks. The average run-time delay provides an even greater temporal slack of almost 20 percentage points. A tuning solution that adapts to the run-time behavior of the application can exploit this temporal slack.

Our simulations show significant amounts of *application variability*. Unlike process variations, voltage and temperature variations are closely coupled to workload characteristics, which vary from application to application and from phase to phase within an application. Figure 5(a) plots the variation of delay across major units for different benchmarks. Within a benchmark, the units have different delay profiles and the relative order of units differs between benchmarks. Figure 5(b) shows the variation of *omne*, a SPEC CPU2006 benchmark, across different chips. The load-store unit (LSU) has the worst behavior for a majority of the chips, owing to large swings in activity followed by long stall periods. The



(a) Across Benchmarks

(b) Across Chips

Figure 5: Worst runtime circuit delay variation for various units with respect to (a) different applications for a single chip and (b) running the SPEC CPU2006 benchmark *omne* with different chips.

variation in the relative order of units between different chips is primarily the result of within-die process variations. Previously proposed static tuning schemes [18] which primarily focus on dealing with process variations cannot adapt to the workload characteristics, hence, fail to adapt to both inter- and intra-application variability. Designing for all the three sources of variations requires dynamic adaptations mechanisms which can effectively adapt to the differences within and across applications. The tuning mechanisms can differ in the temporal granularity of tuning (coarse vs fine) and spatial granularity (global vs local). Section 3.3 presents our proposed tuning solution.

3.3 The Tribeca Framework

This section explores the architectural support required to enable the performance and power benefits of typical-case design margins. Our solution has two parts: 1) low-cost and robust mechanisms to detect and recover from timing violations, to ensure correct and reliable execution; and 2) spatially and temporally fine-grained tuning mechanisms that allow the chip to balance design margins across workloads and blocks.

Existing processors have begun to implement global recovery (GR) mechanisms to handle errors, such as the recovery unit (RU) in POWER6 [19]. While these schemes have initially been proposed to handle infrequent radiation-induced soft errors, they can also be used to handle timing errors. The high frequency of occurrence of timing errors requires a low cost recovery solution. Moreover, spatial differences across the different units require a mechanism which can exploit the differences in error-rates of different units. GR schemes provide a coarse-grained recovery mechanism which over-penalizes parts of the processor that do not experience timing violations. Hence, we propose a fully distributed local recovery mechanism (LR) that is cognizant of inter-unit variability and reduces overall recovery cost in the presence of violations. A fully distributed local recovery mechanism aims to eliminate the global recovery mechanism. However, a fully distributed local-recovery mechanism entails overheads, particularly for the front end of the pipeline. We also propose another flavor of local recovery: partial local recovery (PLR) which augments global recovery with local recovery for the execution units.

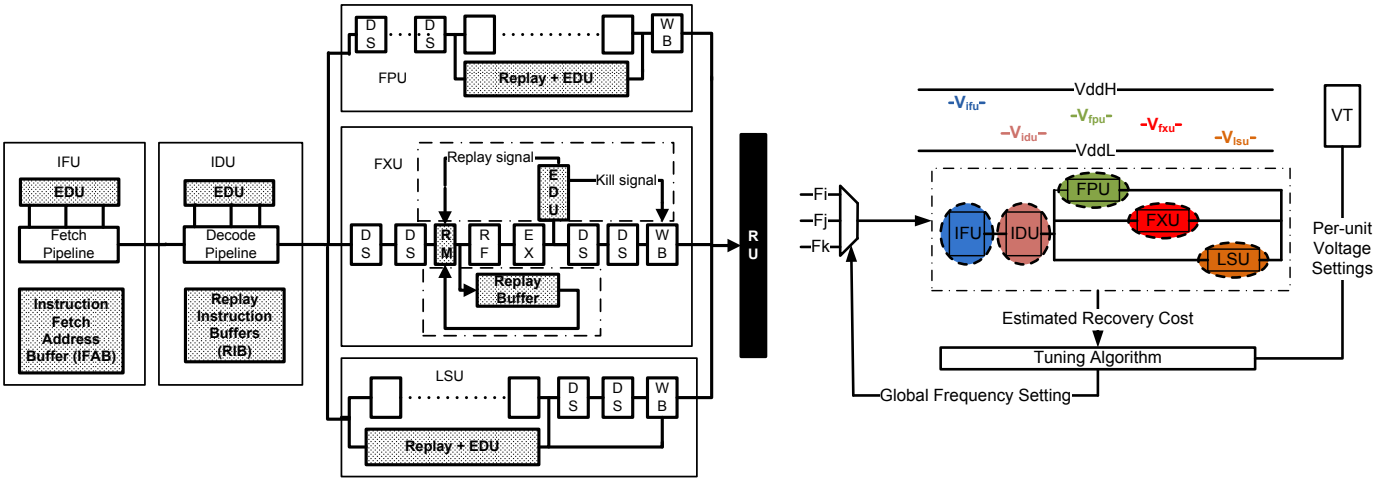
As discussed in Section 3.2, workload and unit-level variability highlight the need for adaptive solutions that can reduce required timing margins. Many algorithms have been proposed to statically or dynamically adjust voltage and/or frequency settings to address temperature and process variations [25, 30]. However, all previously proposed mechanisms have been applied at coarse temporal granularity—either statically, after post-fabrication test, or dynamically, at the OS scheduling interval (a few milliseconds). We show that adaptation at finer-resolutions can enhance the performance gains. The remainder of this section discusses details of the local recovery and fine-grained adaptation mechanisms.

3.3.1 Local Recovery

The recovery unit in POWER6 [19] is an example of a global checkpoint-recovery mechanism in a shipping microprocessor. The RU contains ECC-protected data representing the state of the processor, allowing recovery of internal processor state when an error condition is detected. This recovery process is coarse-grained and hence relatively costly, requiring a full pipeline flush and restart. We propose to replace the RU's global checkpoint-recovery mechanism with a distributed error detection and recovery mechanism. Local detection and recovery can better handle spatial variability across the processor, and it minimizes the recovery cost of violations. A key requirement of a local recovery mechanism is timely detection of violations to prevent propagation of corrupted state across unit boundaries. The shaded logic in Figure 6(a) highlights the additional hardware required for implementing a local recovery mechanism, described below.

Error detection unit (EDU): To maintain correct semantics and ensure that a corrupt instruction does not propagate from the unit in violation to the rest of the pipeline, error detection is performed before the transition boundaries between units. Many server-class microprocessors (including the POWER6) now provide error detection mechanisms distributed throughout the pipeline, often in the form of parity check, ECC, and residue codes [19], and we rely on these for local error detection. Additional circuit-level schemes such as Razor latches can provide supplemental error detection [7, 32, 33], and it is likely that due to rising error rates these schemes will become more pervasive in future microprocessors.

When the EDU detects a violation, it triggers a recovery mechanism that simultaneously flushes the local pipeline and initiates a replay mechanism described below. For each execution pipeline in the processor, the detection unit follows the execution stage and



(a) Baseline microarchitecture with support for local recovery mechanism.

(b) Support for fine-grained frequency/voltage tuning

Figure 6: Proposed local recovery and fine-grained tuning mechanisms. (a) depicts the Baseline microarchitecture with architectural support for local recovery mechanism . Additional hardware is shown by the shaded boxes. Details of the recovery mechanism for the execution units are presented for the fixed-point unit (FXU), and remaining units have the same logic represented as *Replay + EDU* logic. (DS: delay stages; RF: register file access; EX: execute; EDU: Error detection unit; RM: replay mux.). The recovery unit is replaced by our distributed recovery mechanism. (b) highlights the additional support required for a fine-grained frequency (voltage) tuning mechanism.

precedes the delay and writeback stages. In the POWER6 pipeline, all units except the FPU have several delay buffering stages following execution completion to ensure in-order writeback. EDU can take advantage of these stages and perform error detection off the critical path, in most cases with multiple stages before writeback. Writebacks for the FPU pipeline may be delayed depending on the timing of the EDU for FPU operations. Inter-pipeline communication (e.g., Load-to-FXU dependencies) requires flushing both pipelines if the forwarding unit reports a violation, although this is not required for inter-pipeline dependencies through the register file. When an error is detected, the EDU sends a *replay signal* to the replay logic and a *kill signal* to the writeback stages of the execution pipeline. This prevents propagation of corrupted state, hence protecting the architected state of the processor (the register files and memory units). The architected register file must be ECC-protected.

Replay buffer for the execution units: Error detection must trigger a recovery mechanism. The unit in error needs to replay all operations performed on the internal, temporary state corresponding to the in-flight instructions in the unit's pipeline. Replay buffers, often used in processors to provide speculative execution of long latency or load instructions [20], provide recovery for each execution pipeline. Recovery, in the presence of timing-margin violations, can use a similar replay mechanism. The replay buffer stores the source and destination registers and the operation field for each instruction in the execution pipeline. The EDU sends a replay signal to the *replay mux*, which then feeds instructions into the execution pipeline from the replay buffer. A *stop* signal is sent to the scheduler, which halts scheduling of instructions to the recovering execution units; however, it is possible to continue scheduling non-dependent instructions in the remaining execution pipelines. Normal scheduling resumes after all instructions are replayed. Figure 6(a) depicts the details of the replay logic coupled with the EDU for the FXU; other units have similar logic, shown as a composite box (*Replay+EDU*).

Violations detected in the processor trigger a global throttling mechanism, which operates the processor in a low-frequency mode (at half the frequency of the processor). This throttling mechanism ensures that local restart will make forward progress by providing timing slack in the logic. Our simulations show that a slow-restart period of ten cycles is sufficient to guarantee forward progress.

Recovering the front end: Unlike the execution units, the front-end units of the processor (fetch and decode) do not require replay buffers. Instead the front end needs to maintain the instruction stream fed into the back end to enable recovery, the required state being proportional to the depth of the front-end. First, the fetch unit must save the instruction address buffer (PC-chain) to recover the fetched instruction sequence. This is saved in the *instruction fetch address buffers* (IFAB). The size of IFAB is proportional to the pipeline depth of the IFU. Second, the IDU maintains a copy of the instruction buffers (*replay instruction buffers* (RIB)) to enable recovery of the fetched instructions written into the instruction buffers. The instruction buffers can receive eight instructions per thread every cycle. The RIB size is 8 times the number of stages in the IDU. It is important to note that all recovery state buffers (in both the front end and back end) require ECC protection since for functional correctness these buffers must not be corrupted.

Implementation overhead: The baseline processor is designed with conservative margins and operates at 75% of ideal frequency. Global recovery adds error detection mechanisms and centralized recovery state. For a POWER6-style design, the total overhead for detection supported by global recovery is estimated to be at least 15% (based on designer input and publicly available papers).

Our proposed local recovery mechanism requires additional ECC-protected replay state, but can reuse much from an existing global recovery design, e.g. existing recovery state is distributed across local pipelines. For the LSU, much of the local replay logic and state may already exist in most processors to enable speculative execution for load instructions. The FXU pipeline requires a two-stage replay buffer (RF and EX stages), while the FPU pipeline re-

quires an 8-stage replay buffer due to the longer execution pipeline. In contrast to the back end, front-end units have higher replay state overhead since the IFU and IDU units deal with a wide stream of instruction and decode bits. We estimate that the RIB requires 8 instructions per pipeline stage in the IDU, leading to a 48-entry buffer per thread. The additional state required for LR is less than 0.5KB ($<$1% of the core), and the front end accounts for 75% of the implementation overhead. We estimated added latch bits conservatively, using actual low-level design data for the POWER6 core.

Partial local recovery: As discussed above, much of the additional state overhead exists in the front end of the pipeline, with wide and relatively long stages. This motivates us to explore a scheme we call *partial local recovery* in which we provide global recovery for the front-end units and local recovery for the back-end units (FXU, FPU and LSU). In this scheme, violations in the front end initiate global recovery using the RU. However, violations in the execution units initiate local recovery using the local replay logic as explained above. This partial local recovery mechanism exploits the spatial variations of the execution units only, although based on our unit-level analysis, we find that the back-end units are noisier and hence more susceptible to timing violations.

3.3.2 Dynamic Adaptation Mechanism

The activity profile of an application varies with the program phase, directly impacting the voltage and temperature variation across the core. Previous work has considered frequency and/or voltage adaptation, but with a focus on static, post-fabrication test-time tuning to target process variation [18], or coarse-grained tuning to target process and thermal parameters [25, 30]. The smaller time constants associated with voltage variations, coupled with the observation that voltage noise heavily depends on application characteristics [23], imply that solutions will need finer temporal resolution to target the combined effect of PVT-variations. We do not seek to adapt at the granularity of actual voltage noise, with periods on the on the order of 10s to 100s of cycles. Rather, we find that an adaptation interval of a few thousand cycles strikes the right balance between sensitivity to workload phases and acceptable recovery overhead.

We evaluate two fine-grained tuning mechanisms to deal with parameter variations: global, core-wide frequency tuning and local, per-unit voltage tuning. The global frequency tuning mechanism aims to increase performance at some cost in power. Per-unit voltage tuning aims to maximize the collective power-performance metric (BIPS^3/W) [4].

Frequency tuning algorithm: A processor designed for a fixed frequency of operation misses opportunities to run at higher frequency with little or no penalty. Recent work has demonstrated fine-grained frequency adaptation using muxes in the clock tree network to select different clock frequencies at fine granularity [31]. We evaluate a similar frequency adaptation mechanism at a resolution of 10K cycles. Our frequency tuning algorithm selects a frequency, f_i, that maximizes BIPS, as defined

$$BIPS(f_i) = \frac{f_i}{CPI_{original} + CPI_{recovery}(f_i)} \qquad (1)$$

$CPI_{original}$ is the number of cycles per instruction in normal execution without timing violations and $CPI_{recovery}$ is the number of extra cycles per instruction incurred to recover from timing violations.

Ideally, a frequency tuning mechanism should be able to adapt to any frequency required; however, that is not a feasible implementation. Instead we consider frequency steps of 1% of the baseline frequency. Responding to changes at fine resolution requires a method for changing frequency quickly without waiting for a PLL to re-lock. We assume multiple PLLs running at independent frequencies and a multiplexer that chooses among them in a single cycle [31]. Our analysis shows that the maximum frequency jump between consecutive windows is $\pm 3\%$ of the current frequency choice. This suggests using 7 PLLs for frequency adaptation, which is costly to implement. Our analysis shows very little sensitivity to the frequency steps, a 3% frequency step can achieve gains similar to those of a 1% frequency step, and three PLLs are sufficient.

Figure 6(b) highlights the required support for enabling fine-grained global frequency tuning[2] . We assume a single clock and frequency domain. The tuning algorithm determines the optimal frequency (that maximizes BIPS) from the three available frequency choices. Fast frequency switching is enabled by the three PLLs locked at different frequencies muxed onto one clock network (similar to the implementation presented in [31]).

Voltage tuning algorithm: Device tuning techniques seek to optimize delay by modifying the threshold voltage or supply voltage. For example, adaptive body biasing adjusts the threshold voltage for individual blocks of transistors compensate for fluctuations in threshold voltages arising from the manufacturing process [30]. Another recently proposed scheme, voltage interpolation, provides the ability for different groups of logic gates within a block to select between two static supply voltages (high and low) to accommodate gates that deviate from nominal delays [18]. In both of these approaches, devices are partitioned into groups or blocks at design time, but the block-level tuning occurs after fabrication and during test. We envision extending these approaches for run-time tuning that can occur every 10K cycles.

The choice of voltage affects both the dynamic power consumption (quadratic relationship) and leakage power (linearly related to supply voltage), so it affects total power consumption. Our voltage tuning mechanism chooses the voltage for each unit to maximize the overall BIPS^3/W of the core, where BIPS is calculated using Equation 1. Again, since an exhaustive search for the appropriate voltage setting is not practical, we limit our search space to $\pm 1\%$ of the current voltage setting, which translates to three possible choices for each of the units. At every decision interval, a voltage setting for each unit (five units in our framework) is chosen from three possible settings for each unit, requiring an evaluation of a total of 3^5 settings. We apply a *hill-climbing* [24] heuristic to reduce overhead associated with converging to an optimal setting.

Figure 6(b) also highlights the support required for a fine-grained voltage tuning mechanism[2]. The tuning algorithm determines the voltage setting for each of the five units using the algorithm described above. Using two static voltage supplies (VddH and VddL), our voltage tuning mechanism can provide different per-unit voltages, similar to the static voltage interpolation scheme described in [18]. Similar to the frequency tuning mechanism, we assume a single clock/frequency domain, however, we do not tune the frequency but only the per-unit voltage.

Monitoring delay: Both tuning mechanisms (frequency and voltage) need to estimate possible violations for different voltage or frequency settings being evaluated. We assume the presence of delay-skitter circuits for each unit of the processor to estimate expected timing violations at different voltage and frequency settings. Whenever the delay of a critical path exceeds the clock period being evaluated, a violation is flagged. Delay-skitter circuits are "canary" circuits that can capture local voltage and temperature conditions.

[2] The assumed mechanisms are not reflective of what is available or possible in the real POWER6 family products.

Parameter	Equation	Parameter	Equation
Delay of a Gate	delay $\propto \frac{L_{eff}V}{\mu(V-V_{th})^{\alpha}}$	Voltage	$V = V_{dd} - V_{drop}$ where $V_{drop} = Z\left(\frac{P_{total}}{V_{dd}}\right)$
Leakage Power	$P_{leakage} \propto VT^2 e^{-qV_{th}/kT}$	Threshold Voltage	$V_{th} = V_{th0} + k_1(T - T_0)$
Dynamic Power	$P_{dynamic} \propto CV^2 f$	Mobility	$\mu \propto T^{-1.5}$
Total Power	$P_{total} = P_{dynamic} + P_{leakage}$		
Package Resistance or Inductance	$R(L)_{unit} = R(L)_l \frac{A_{core}}{A_{unit}}$	Package Capacitance	$C_{unit} = C_l \frac{A_{unit}}{A_{core}}$

Table 1: Equations used in modeling variations

They are used in the POWER6 core for diagnostic measurement of timing uncertainties due to various sources of variation (PLL jitter, clock distribution skew, supply noise, or device variation) [8]. Their resolution is limited to one FO1 inverter delay, a resolution of 5-8 ps [8]. Our baseline core has a frequency of 4.7GHz, or a clock period of 212 ps. Thus we can use the existing delay-skitter circuits to estimate the impact of 3-4% tuning steps. Our analysis shows less than 1% loss in performance gain when operating at steps of 3% or 4% as opposed to 1% frequency steps. An alternative delay monitoring circuit could rely on time-to-digital converters rather than skitter circuits if finer resolution is needed [11].

Predicting voltage or frequency settings: In the above discussion, we assume complete knowledge of the delay behavior of applications (which determines the violation profile at any given setting) across different voltage and frequency settings for a given decision window. However, this is an upper-bound approach, with complete oracle knowledge of the present window. Operating at fine resolutions of 10K cycles does not provide room for exploration phases [25]. Keeping this in mind, we use a simple predictor approach based on the values monitored for the previous window. We deploy a simple last-value predictor (LVP), which chooses the settings of the current window based on information about timing violations, throughput, and power dissipation gathered in previous windows for the different settings being considered. We compared our light-weight predictor with an oracle predictor, which has complete information for each decision window. Our results show very good prediction accuracy; the LVP predictor yields gains that are within 2% of those obtained with the oracle predictor.

Choice of resolution: Previous work on processor tuning either tuned statically at design time or used coarse tuning intervals on the order of 100 million cycles. Such coarse resolutions work well when considering either process variations alone, or those plus temperature, but they are not sufficient when adapting for a combination of all parameter variations (PVT). A coarse resolution (e.g, a resolution of 10M cycles) has the effect of filtering out high-frequency variations, leading to conservative choices and missing out opportunities to run at higher frequencies.

4. Experimental Framework

This section presents the overall experimental framework, which consists of the models used to understand the effects of different parameter variations and the power and performance simulator.

4.1 Modeling Variations

An accurate model of parameter variations is a key aspect of this paper. We model the three main sources of variations (PVT) to evaluate their effects on processor power and performance. The delay of a gate is a function of process variation plus runtime variations (voltage and temperature). Process variation affects threshold voltage and effective gate length of a gate; voltage variation affects supply voltage; and temperature variation affects leakage power and threshold voltage.

Modeling process variations: Process variation mainly affects the threshold voltage (V_{th}) and effective gate length (L_{eff}) of the transistors. These two parameters in turn affect the delay of the gate (Table 1) and the leakage power of the gate ($P_{leakage}$). In this work, we capture the effect of within-die variations using the VARIUS model [26] which generates different V_{th} and L_{eff} values for each unit shown in Figure 6(a). Each individual experiment uses a batch of 100 chips that have different V_{th} (and L_{eff}) maps generated with the same σ (standard deviation), μ (mean), and ϕ (correlation range). We assume that the random and systematic components have equal variances. Gate length has a correlation range close to half of the chip's width. Since the systematic component of V_{th} variation directly depends on the gate length variation, we assume $\phi = 0.5$ for V_{th}.

Modeling voltage variations: Sudden current swings due to activity fluctuations in processors, when coupled with parasitic resistances and inductances in the power delivery subsystem, give rise to large voltage swings. A decrease in supply voltage leads to an increase in the delay of the gates. Voltage also impacts both the dynamic power and leakage power of the system.

Since voltage variations are strongly coupled to the characteristics of the underlying power delivery subsystem, it is important to have good models for processor activity, power consumption, and the power delivery subsystem. Most of the earlier work that seeks to address voltage noise at the architectural level uses a simplified second-order lumped model [12], which captures the mid-frequency response of the system. However, such a model fails to capture within-die spatial variation of voltage. While detailed grid models have also been proposed and used, the large number of nodes leads to prohibitively high simulation times. As a compromise, we replace the simplified lumped model with a per-unit distributed power grid to capture block-level voltage fluctuations and spatial interactions. By appropriately scaling R, L, and C of each unit's power grid with respect to area, this model enables relatively fast simulations while maintaining high accuracy that closely matches a detailed grid model. For example, the R, L, and C for any unit will scale as given in Table 1, where A_{unit} represents the area of the unit and A_{core} represents the area of the core. R_l, L_l, and C_l correspond to values found in a lumped model.

Lastly, it is important to note that voltage-dependent delay variations differ between memory-dominated units (IFU and LSU) and logic-dominated units (IDU, FXU and FPU). Large wire capacitance and weak devices in memory delay paths lead to steeper delay vs. voltage curves compared to logic gates. This is consistent with the trends presented for a 45nm Intel process [21].

Modeling temperature variations: Temperature affects the delay of gates via its impact on mobility (μ) and threshold voltage. As temperature increases, carrier mobility degrades (Table 1) and

Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BRU, 1 CRU
Branch predictor	16K-entry BHT, 2 bits/entry
Instruction buffer	64 entries for each thread
Decode width	8
Dispatch/complete width	7
L1 D cache	64 KB, 8-way, line size is 128 bytes
L1 I cache	64 KB, 4-way, line size is 128 bytes
L2 cache	4 MB, 8-way, line size is 128 bytes
L3 cache	16 MB, 16-way, line size is 128 bytes

(a) Simulator parameters.

Tech: 65nm ; Frequency (worst-variations): 4.7GHz ; Core Size: 7x7 mm^2
Process Parameters
Number of Chips per experiment: 100
V_{th} : 150mV, at V_{dd}=1.15V, T=80C ; ϕ: 1cm ; α: 1.3
V_{th}'s σ/μ: 0.09 ($\sigma_{ran}/\mu = \sigma_{sys}/\mu = 0.064$)
L_{eff}'s σ/μ: 0.045 ($\sigma_{ran}/\mu = \sigma_s ys/\mu = 0.032$)
Voltage and Temperature Modeling
V_{DD}: 1.15V
Resonance Frequency: 100MHz; Peak Impedance: $12m\Omega$
$P_{leakage}$ (at 80C): 19.6W; Thermal Samples: 10K cycles

(b) Process, Voltage and Temperature Parameters

Figure 7: Simulated processor and variation parameters.

circuitry slows down [16]. While threshold voltage decreases with increasing temperature, mobility degradation dominates at nominal supply voltages with sufficiently high gate overdrive. Variation in temperature has a slower time constant (on the order of a few tens of thousands of cycles) as compared to changes in voltage (on the order of a few cycles). To model temperature effects, we rely on HotSpot models [28] and focus on temperature transients to accurately capture workload dependencies. Operating temperature from HotSpot feeds back into the power and delay models to accurately account for its runtime effects. Across the limited operating temperature range we assume (80C to 100C), temperature fluctuations have smaller impact on delay than voltage droops.

4.2 Performance and Power Simulator

Power density plays an important role in understanding the behavior of different units in the context of voltage and temperature variations. We use a detailed cycle-accurate PowerPC simulator framework to model a microarchitecture that resembles the published POWER6 processor core [17]. The simulator supports both single- and multi-threaded (SMT) execution modes, and supports a multi-level private cache hierarchy (with a memory model). However, it is primarily a uniprocessor simulator and thus does not model other chip-level elements; nor does it support a multi-core simulation environment. The high-level microarchitecture parameters are listed in the table shown in Figure 7(a). The simulator was coupled with an activity-based power model to obtain cycle-by-cycle per-unit power consumption. The power model is based on circuit-extracted power estimates and accurately models both dynamic and leakage power. We assume 15% of the latches cannot be clock-gated. The simulated workloads are a subset of the SPEC CPU2006 benchmark suite along with one TPC-C benchmark. The traces used in this study were sampled from the full reference input set to obtain 100 million instructions per benchmark [14]. Systematic validation was performed to compare the sampled traces against the full traces to ensure accurate representation.

5. Evaluation

The "better than worst-case" strategy strives to improve power-performance efficiency of the system. We present results for the BIPS^3/W metric, which is a voltage-invariant power-performance efficiency metric for processors [4]. Our results are presented with respect to either an *ideal* processor with no parameter variations or a *baseline* processor with worst-case margins (operating voltage of 0.9V and temperature of 100C). The baseline processor operates at 75% frequency of the ideal processor. When summary results are presented, they include the average across all 100 chips simulated to model process variations and the entire benchmark suite. We evaluate different combinations of recovery (global, local, and partial local), frequency tuning (coarse and fine resolution), and voltage tuning schemes(fine and static). We present all results with the last value predictor described in Section 3.3.2, and discuss overhead relative to an oracle predictor.

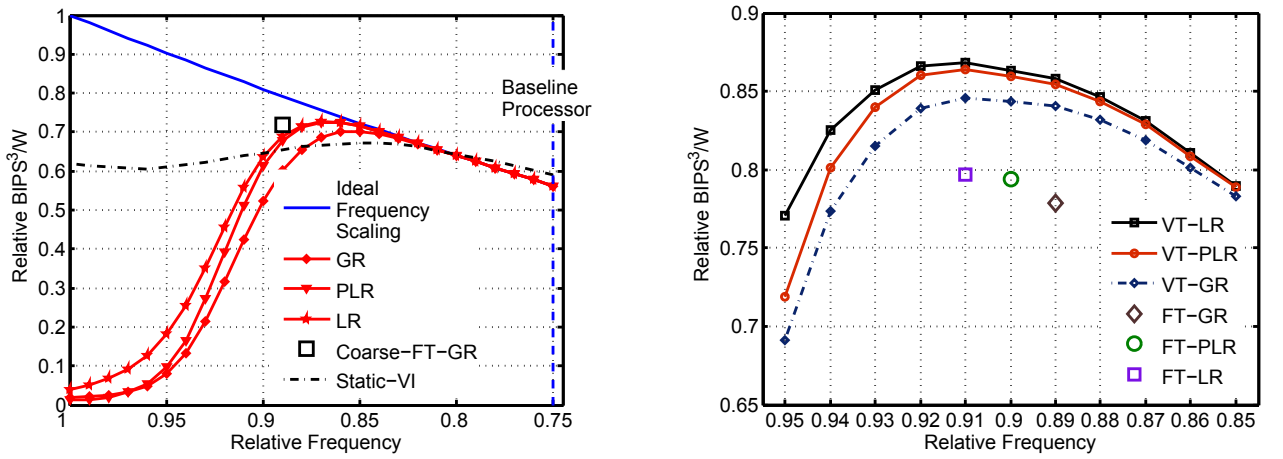
5.1 Evaluation of Static Schemes

Designing a system for typical conditions requires a low cost recovery mechanism to guarantee correctness in the presence of violations. Figure 8(a) presents the resulting relative BIPS^3/W obtained by deploying different flavors of recovery mechanisms while operating at frequencies greater than the frequency of a baseline processor. The plot shows a linear increase in BIPS^3/W for the recovery-only schemes (GR, PLR, LR) until benefits start decreasing with further increases in frequency. This drop in performance occurs due to rapid increases in the cost of recovering from violations, nullifying the gains obtained by running at a higher frequency.

Recovery mechanisms enable the processor to operate at aggressive frequencies by guaranteeing correctness in the presence of violations, but they should not overwhelm the gains of operating with reduced margins. Providing a global recovery mechanism (GR) enables the processor to maximize BIPS^3/W while running at 86% of the *ideal processor*, with a 12% frequency increase over a baseline processor. However, the global mechanism fails to exploit the spatial variations between various units. The local recovery mechanism (LR) can take advantage of the spatial variation in delay characteristics of various units leading to a further increase in the BIPS^3/W obtainable at a frequency 87% of an ideal processor. There are slight differences between the LR and partial local recovery (PLR) mechanisms up to 90% relative frequency. This is a result of execution units being predominantly in error at frequencies lower than 90% leading to similar recovery costs for LR and PLR schemes. However, as frequency increases beyond 90% all of the units start experiencing larger numbers of violations, resulting in penalties similar to a global recovery mechanism for the PLR scheme. An LR scheme is able to achieve a BIPS^3/W increase of 29% over a baseline processor. This is because an LR scheme has the benefit of a lower recovery penalty even in the worst-case scenario of all units in error, as the local penalty is determined by the maximum penalty of any unit at any given instance of recovery. Moreover, an LR scheme is able to exploit the spatial differences in the error-rates across different units.

5.2 Evaluation of Tuning Mechanisms

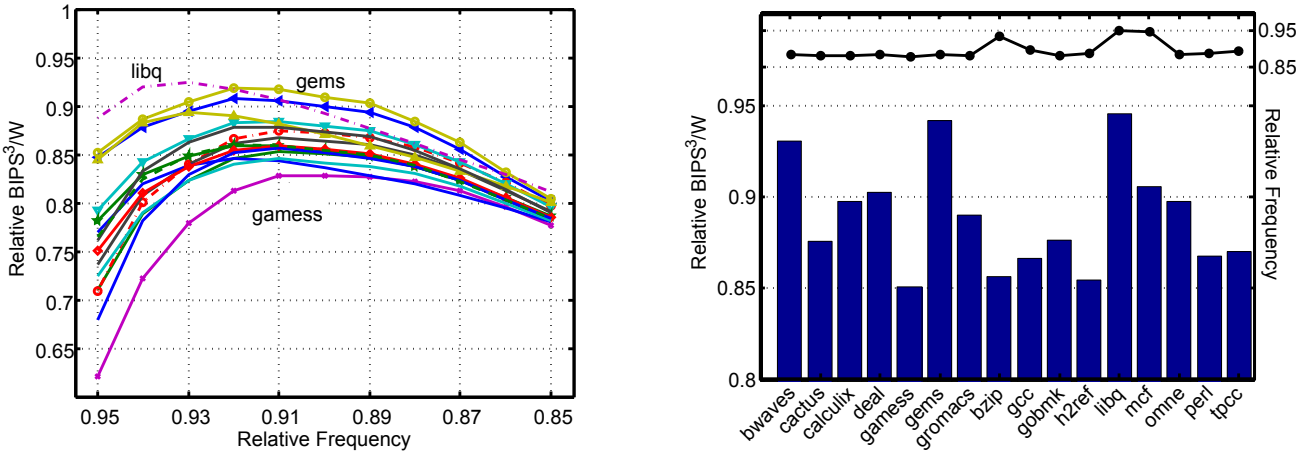
The delay characteristics of cores can change temporally according to workload characteristics. The temporal characteristics affect the spatial characteristics of circuit delay, since activity patterns cause both spatial and temporal variation of circuit delay across the core. Previous schemes that adapt for variations have focused on dealing with process and temperature variation by adapting the fre-



(a) Recovery Schemes

(b) Tuning Schemes

Figure 8: Recovery and tuning schemes. (a) BIPS^3/W (relative to an ideal processor) vs. operating frequency. Ideal frequency scaling represents a frequency scaling scheme with no recovery cost. (b) Relative BIPS^3/W metric for fine-grained tuning. The frequency tuning mechanisms are depicted at the mean of the frequencies chosen by the tuning algorithm.



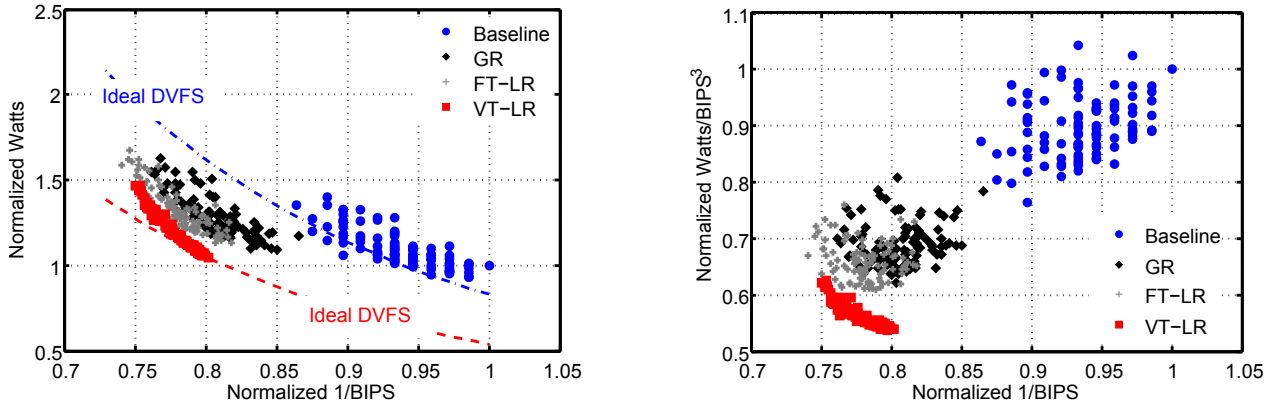
(a) Variation of BIPS^3/W across benchmarks.

(b) Optimal BIPS^3/W and frequency per benchmark.

Figure 9: Application sensitivity for VT-LR scheme.

quency, supply voltage, or body-bias settings at a very coarse granularities [25], on the order of a few milliseconds, or statically tune the parameters per chip at test time [18]. Figure 8(a) also shows the gains in BIPS^3/W obtained through two previously proposed schemes: tuning the frequency at coarse granularity (Coarse-FT-GR [25]) and tuning the voltage statically (Static-VI, representative of a voltage-interpolation mechanism [18]). A coarse frequency tuning technique enables a slight increase (1%) in BIPS^3/W over a global recovery mechanism, but the tuning benefits of this scheme are limited by voltage noise. In contrast, the static voltage tuning mechanism aims to choose voltage settings for individual units based on a static profile of the processor across several benchmarks, which leads a 20% increase in BIPS^3/W over a baseline processor. However, the static voltage tuning mechanism does not assume the presence of a recovery mechanism and hence must conservatively choose its voltage settings to guarantee correctness across the benchmark suite. This leads to a significant increase in the overall power of the core, and hence a flattened BIPS^3/W metric with increasing frequency.

Figure 8(b) presents the results for fine-grained tuning mechanisms that we propose in this work. The fine-grained frequency tuning mechanism (FT) effectively adapts the frequency of the processor based on workload behavior leading to an effective increase of attainable processor frequency to 91% of an ideal processor for a local recovery mechanism and 89% for the global recovery mechanism, with a slight difference in the BIPS^3/W metric. The fine-grained voltage tuning mechanism equipped with global recovery (VT-GR) is able to achieve a further 6% increase over the FT-LR scheme. Similarly, applying the voltage tuning mechanism with different flavors of local recovery leads to a 9% (8% for VT-PLR) increase in the BIPS^3/W over the FT-LR mechanism. The local recovery mechanism gives the best voltage tuning results, achieving a 55% increase in BIPS^3/W over the baseline processor. The last value predictor results are within 2% of that of an oracle predictor



(a) Power-BIPS Efficiency

(b) BIPS^3/W Efficiency

Figure 10: Tightening the power-performance spread. This figure presents the evaluation of Power Efficiency and BIPS^3/W metric across all chips, normalized to the worst-chip from the *baseline* cluster.

indicating reasonable prediction estimates. Voltage tuning presents a per-unit tuning knob as compared to the global frequency tuning of frequency adaptation, and hence is able to adapt both spatially and temporally. Coupled with a local recovery mechanism, voltage tuning can provide the best power-performance gains.

We further explore the sensitivity of voltage tuning with local recovery (VT-LR) to different applications in Figure 9. Figure 9(a) shows that most of the benchmarks exhibit similar BIPS^3/W-to-frequency behavior. Notable outliers are *libq, gems* and *gamess*. Low-IPC applications *libq* and *gems* incur long stall periods, and the low-activity periods provide greater opportunity to reduce voltage without impacting performance. On the other hand, *gamess* is a high-IPC benchmark with high activity and many violations due to large swings, which provides fewer opportunities to adapt the voltage. Figure 9(b) depicts the optimal relative BIPS^3/W metric for each benchmark and also the frequency that maximizes the BIPS^3/W metric. This can be viewed as the best achievable gains in the presence of a per-unit voltage tuning mechanism, coupled with an application-level global frequency tuning mechanism. We observe 58% gains on an average for a global frequency, local voltage tuning mechanism.

Our proposed local recovery coupled with fine-grained tuning approaches allows designers to tighten the power-performance spread, enabling more efficient speed-binning. Figure 10(a) compares the power-BIPS efficiency of three of the solutions to that of a chip designed with worst-case margins (represented by the *baseline* cluster). The power and BIPS are both normalized to a chip with the minimum frequency when designed for worst-case conditions. The ideal dynamic-voltage and frequency scaling curve has been plotted for a median chip with no tuning mechanism and for the same chip assuming a local voltage-tuning mechanism. An increase in BIPS (or decrease in 1/BIPS) can be observed as recovery mechanisms and tuning mechanisms are applied. A 25% increase in power translates to a 23% decrease in the 1/BIPS metric (a 30% increase in performance) for a system equipped with local voltage tuning, versus a 13-to-7% decrease in 1/BIPS (7-to-14% increase in performance) for the baseline cluster of 100 chips simulated. Importantly, we also find that the spread of chips significantly decreases after applying voltage tuning. The baseline chips have a spread of 16% of BIPS and 50% of power. The tuned chips have a lower spread in both performance and power, with a 7% spread in BIPS and 40% in power.

Figure 10(b) presents another view of the data, evaluating the normalized W/BIPS^3 metric with respect to a normalized 1/BIPS metric (normalized relative to the worst of the chips designed for worst-case margins). Since the baseline cluster is designed for worst-case operating conditions, without any tuning mechanism, a large spread in the cluster across both W/BIPS^3 and 1/BIPS can be observed. However, a shift in the cluster is observed for the system augmented with a global recovery mechanism to enable typical-case design. A frequency tuning mechanism with local recovery causes a further shift in the cluster to the left, indicating a potential to increase overall performance with minimal increase in power. Finally, the per-unit voltage tuning (VT-LR) mechanism is able to adapt each chip in the presence of variations and tightens the cluster, indicating a run-time customizable design to maximize performance with minimal power overhead.

6. Related Work

Designers have started analyzing ways of dealing with designing for the typical conditions. Several techniques have been proposed for detection and mitigation of timing errors. In Razor [7], the authors propose a circuit-level mechanism to dynamically detect and correct timing failures by augmenting critical flip-flops in the microprocessor pipeline with shadow latches. These shadow latches rely on a delayed clock to provide additional timing margins and enable detection of speed-path failures. A recent work by Bowman et al [3] presents an implementation of timing-error detection and correction circuits which help in eliminating V_{dd} and temperature guardbands as well as maximize throughput. DIVA also provides a method to dynamically detect and recover from transient errors [1,5,34]. This scheme relies on a checker processor that runs in parallel with the main out-of-order core, checking results prior to committing the instructions. DIVA requires duplicate functional units (e.g., INT/FPU/SSE units) that consume additional power and area resources. This paper also relies on these error-detection techniques to detect timing-violations. However, these error-detection techniques trigger a global recovery mechanism [6, 9, 10, 27]. We are tackling the high frequency nature of temporal variations coupled with spatial variation across the processor, so we need a local recovery scheme. Micro-rollback [29] presents an approach to buffering state at various stages/logic in the pipeline to offload error-detection and correction from the critical path. However, the

recovery scheme is global in nature. On the contrary this paper presents a distributed local recovery mechanism.

Post-fabrication test-time [18] tuning of voltages using a voltage interpolation scheme has been proposed, mainly targeting variability due to process variations. Researchers are also exploring the opportunities of dynamically adapting processor parameters by exploiting workload variability. Tschanz et al [31] explore schemes to dynamically adapt various combinations of frequency, V_{dd} and body bias to changes in temperature, supply noises, and transistor aging, to maximize average performance or improve energy efficiency. A prototype containing a TCP offload accelerator core is implemented. EVAL [25] presents a high-dimensional dynamic adaptation technique using a machine learning algorithm for maximizing performance and minimizing power in the presence of parameter variations. The adaptation mechanism mainly addresses process+temperature variations, and does not account for high-frequency voltage changes. Hence, the adaptation scheme is at a coarse-granularity of 120ms.

7. Conclusion

Parameter variations threaten to significantly degrade performance benefits offered by technology scaling. Previous solutions for dealing with variations treat each source of variation (process, voltage, and temperature) independently, ignoring potential interactions between them. This paper presents a novel framework, Tribeca, which aims to better understand the complex interactions between the different sources of variations to enable design of efficient microarchitectural solutions for dealing with them. We make three notable contributions. First, we present a detailed characterization of spatial and temporal properties of all three variations combined together. Second, we propose a distributed local recovery mechanism that exploits the spatial characteristics of variations to provide low-cost recovery from timing violations. Third, we show how to design effectively for typical-case operating conditions. To achieve this we explore fine-grained voltage or frequency tuning to adapt to workload behavior. A fine-grained, per-unit voltage tuning technique provides the best power-performance efficiency, achieving a 55% increase in power-performance efficiency along with a 21% increase in processor frequency, compared to a processor designed for worst-case timing margins.

8. Acknowledgments

We are thankful to John-David Wellman for making the POWER6-like simulator available for instrumentation and Hans Jacobson for his help in embedding the power model. We are also grateful to Glenn Holloway and the anonymous reviewers for their comments and suggestions. This work is supported by National Science Foundation grants CSR-0720566, CCF-0702344 and CCF-0448313 and DARPA agreement HR0011-07-9-002. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or DARPA.

References

[1] T. M. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *MICRO 32*, 1999.

[2] S. Borkar et al. Parameter variations and impact on circuits and microarchitecture. In *DAC*, June 2003.

[3] K. A. Bowman et al. Energy-efficient and metastability-immune timing-error detection and instruction replay-based recovery circuits for dynamic variation tolerance. In *ISSCC*, 2008.

[4] D. Brooks et al. Poweraware microarchitecture: Design and modeling challenges for nextgeneration microprocessors. *IEEE Micro*, 2000.

[5] S. Chatterjee, C. Weaver, and T. Austin. Efficient Checker Processor Design. In *MICRO*, 2000.

[6] K. Constantinides et al. Bulletproof: A defect-tolerant CMP switch architecture. In *HPCA*, 2006.

[7] D. Ernst et al. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *MICRO*, 2003.

[8] R. Franch et al. On-chip timing uncertainty measurements on IBM microprocessors. In *IEEE International Test Conference*, Oct. 2007.

[9] B. Greskamp and J. Torrellas. Paceline: Improving single-thread performance in nanoscale CMPs through core overclocking. In *PACT*, 2007.

[10] M. S. Gupta et al. DeCoR: A Delayed Commit and Rollback Mechanism for Handling Inductive Noise in Processors. In *HPCA*, 2008.

[11] V. Gutnik and A. Chandrakasan. On-chip picosecond time measurement. In *Symposium on VLSI Circuits*, 2000.

[12] D. Herell and B. Becker. Modelling of Power Distribution Systems for High-Performance Microprocessors. In *IEEE TAP*, 1999.

[13] International Technology Roadmap for Semiconductors. Process integration, devices and structures, 2007.

[14] V. Iyengar, L. Trevillyan, and P. Bose. Representative Traces for Processor Models with Infinite Cache. In *HPCA*, 1996.

[15] R. Joseph et al. Control Techniques to Eliminate Voltage Emergencies in High Performance Processors. In *HPCA*, 2003.

[16] K. Kanda et al. Design impact of positive temperature dependence on drain current in sub- 1-V CMOS VLSIs. *JSSC*, 36.

[17] H. Q. Le et al. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51(6), 2007.

[18] X. Liang et al. ReVIVaL: Variation Tolerant Architecture Using Voltage Interpolation and Variable Latency. In *ISCA*, 2008.

[19] M. Mack, W. Sauer, S. Swaney, and B. Mealy. IBM POWER6 reliability. *IBM Journal of Research and Development*, 51(6), 2007.

[20] A. A. Merchant, D. J. Sagger, and D. D. Boggs. Computer processor with a replay system. United States Patent 6,163,838, Dec. 2000.

[21] K. Mistry et al. A 45nm logic technology with high-k+metal gate transistors, strained silicon, 9 Cu interconnect layers, 193nm dry patterning, and 100% pb-free packaging. In *IEDM*, 2007.

[22] M. D. Powell and T. N. Vijaykumar. Pipeline muffling and a priori current ramping: architectural techniques to reduce high-frequency inductive noise. In *ISLPED*, 2003.

[23] V. J. Reddi et al. Voltage Emergency Prediction: A Signature-Based Approach To Reducing Voltage Emergencies. In *HPCA*, 2009.

[24] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[25] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas. EVAL: Utilizing Processors with Variation-Induced Timing Errors. In *MICRO*, 2008.

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

[27] S. Shyam et al. Ultra Low-Cost Defect Protection for Microprocessor Pipelines. In *ASPLOS*, 2006.

[28] K. Skadron et al. Temperature-aware microarchitecture. In *ISCA*, 2003.

[29] Y. Tamir and M. Tremblay. High-performance fault tolerant vlsi systems using micro rollback. *IEEE TOC*, 39(4), 1990.

[30] R. Teodorescu et al. Mitigating Parameter Variation with Dynamic Fine-Grain Body Biasing. In *MICRO*, 2007.

[31] J. Tschanz et al. Adaptive frequency and biasing techniques for tolerance to dynamic temperature-voltage variations and aging. In *ISSCC*, 2007.

[32] A. Uht. Achieving typical delays in synchronous systems via timing error toleration. Electrical and Computer Engineering Tech Report 032000-0100, University of Rhode Island, March 2000.

[33] X. Vera, O. Unsal, and A. Gonzalez. X-Pipe: An Adaptive Resilient Microarchitecture for Parameter Variations. In *ASGI*, 2006.

[34] C. Weaver and T. M. Austin. A Fault Tolerant Approach to Microprocessor Design. In *DSN*, 2001.