Designing a Processor From the Ground Up to Allow Voltage/Reliability Tradeoffs

Andrew B. Kahng[†+], Seokhyeong Kang[†], Rakesh Kumar[‡], John Sartori[‡]

[+]CSE and [†]ECE Departments
University of California, San Diego
La Jolla, CA 92093-0404

[‡]Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, IL 61801

***Abstract*—Current processor designs have a critical operating point that sets a hard limit on voltage scaling. Any scaling beyond the critical voltage results in exceeding the maximum allowable error rate, i.e., there are more timing errors than can be effectively and gainfully detected or corrected by an error-tolerance mechanism. This limits the effectiveness of voltage scaling as a knob for reliability/power tradeoffs.**

In this paper, we present *power-aware slack redistribution*, a novel design-level approach to allow voltage/reliability tradeoffs in processors. Techniques based on power-aware slack redistribution reapportion timing slack of the frequently-occurring, near-critical timing paths of a processor in a power- and area-efficient manner, such that we increase the range of voltages over which the incidence of operational (timing) errors is acceptable. This results in *soft architectures* - designs that fail gracefully, allowing us to perform reliability/power tradeoffs by reducing voltage up to the point that produces maximum allowable errors for our application. The goal of our optimization is to minimize the voltage at which a soft architecture encounters the maximum allowable error rate, thus maximizing the range over which voltage scaling is possible and minimizing power consumption for a given error rate. Our experiments demonstrate 23% power savings over the baseline design at an error rate of 1%. Observed power reductions are 29%, 29%, 19%, and 20% for error rates of 2%, 4%, 8%, and 16% respectively. Benefits are higher in the face of error recovery using Razor. Area overhead of our techniques is up to 2.7%.

I. Introduction

Traditionally, processors have been designed to always operate correctly, even when subjected to a worst-case combination of non-idealities. Conservative guardbands (in terms of voltage margins, for example) are incorporated into design constraints to ensure correct behavior in all possible scenarios. However, designing for a conservative operating point incurs considerable overhead, in terms of power and performance [5]. Overheads are worse for technologies with increased variations [26].

Several *better-than-worst-case* (BTWC) design approaches [1] have been recently proposed that allow tradeoffs between reliability and power/performance. Such approaches provide power/performance benefits by targeting average-case conditions, while an error detection/correction mechanism deals with errors in the worst-case. Razor [5], for example, is a well-known circuit-level technique to detect and correct timing errors due to frequency, temperature, and voltage variations. Razor detects timing violations by supplementing critical flip-flops with *shadow latches*. A shadow latch strobes the output of a logic stage at a fixed delay after the main flip-flop; if a timing violation occurs, the main flip-flop and shadow latch will have different values, signaling the need for correction. Correction involves recovery using the correct value(s) stored in the shadow latch(es). Similarly, system-level techniques such as Algorithmic Noise Tolerance [12] have proved effective in overcoming timing errors in specific domains. Such techniques allow timing errors due to frequency/voltage overscaling to propagate to the system or the application. The applications have algorithmic and/or cognitive noise tolerance and, therefore, perform application-level error correction. Application- or system-level error detection and correction is also assumed for recently proposed probabilistic SOCs [3] and stochastic processor architectures [33], [34] which are also classes of BTWC designs.

The effectiveness of BTWC techniques is limited, however, for high performance general-purpose microprocessors. This is because current general-purpose processor designs appear to have a *critical operating point* (see Figure 1) that sets a hard limit on voltage scaling [19]. The Critical Operating Point (COP) hypothesis [19], in the context of voltage scaling, states the following about large CMOS circuits (e.g., general-purpose microprocessors):

> There exists a critical operating voltage V_c for a fixed ambient temperature T, such that
>
> - Any voltage below V_c causes *massive* errors
> - Any voltage above V_c causes *no* voltage-induced timing errors
> - In practice, V_c is not a single point, but is confined to an extremely narrow range for a given ambient temperature, T_c

The hypothesis is based on the fact that a large number of timing paths in modern CMOS circuits are almost as long as the critical path. (In chip implementation, this is called the “wall of (critical) slack” in signoff timing reports.) This implies that timing errors, when they occur, are massive. The Critical Operating Point (COP) hypothesis suggests that any scaling beyond the critical voltage (*voltage overscaling*) will result in exceeding the maximum allowable error rate, rendering an error-tolerance mechanism ineffective. In other words, overscaling will lead to more timing errors than can be corrected by the error-tolerance mechanism.

While the experiments in [19] provided the basis for the

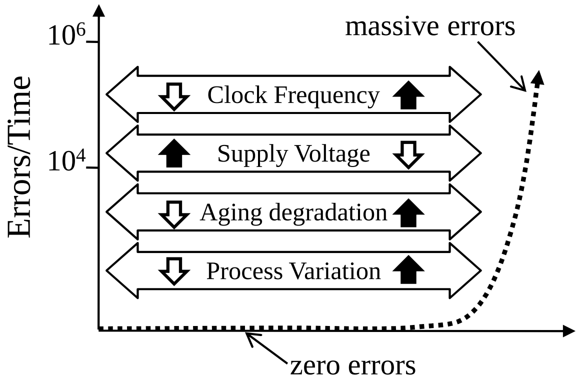


Fig. 1. Traditional designs exhibit a critical operating point. Scaling beyond this point results in catastrophic failure. (Critical Operating Point Hypothesis [19])

COP hypothesis, we confirm the critical operating point hypothesis in this paper for several modules of the *OpenSPARC T1* processor [27]. The existence of critical operating point behavior for modern processors limits the potential of voltage scaling as a knob for reliability vs. power tradeoffs, particularly those based on BTWC techniques.

The goal of this paper is to increase the effectiveness of BTWC designs for general-purpose microprocessors. We propose, *power-aware slack redistribution*, a novel design-level approach to allow voltage/reliability tradeoffs in processors. Techniques based on power-aware slack redistribution reapportion timing slack of the frequently occurring, near-critical paths in a processor design in a power- and area-efficient manner, so as to increase the range of voltages over which an error-tolerance mechanism encounters an acceptable number of timing errors. The end result is a *soft architecture*, i.e., a design that fails gracefully, allowing us to perform reliability-power tradeoffs by reducing voltage down to a point that produces the maximum allowable error rate that is appropriate for our application. This translates to significant processor power savings with a small degradation in application performance. The main contributions of our work are summarized as follows.

- We confirm through sampling that modules of the *Sun OpenSPARC T1* processor indeed demonstrate critical operating point behavior in the face of voltage scaling, even when extra timing slack has been garnered by running synthesis, placement and routing (SP&R) with tighter timing constraints than needed (Section III). We quantify the opportunity cost (in terms of potential power savings) due to the critical operating point behavior.
- We investigate *power-aware slack redistribution*, a novel approach to produce *soft architectures*, i.e., processor designs that fail gracefully, instead of catastrophically (Section IV). We also formulate the power-aware slack redistribution problem as a design optimization problem that allows more meaningful voltage-reliability tradeoffs (Section III).
- We propose post-layout cell sizing (followed by incremental placement and routing) as a technique for power-aware slack redistribution (Section IV-A). To the best of our knowledge, this is the first application of cell sizing to BTWC-driven slack redistribution.
- We show that power-aware slack redistribution techniques can extend the range over which voltage scaling is possible and reduce power consumption for a given error rate. Our experiments demonstrate 23% power savings over the baseline design at an error rate of 1%. Observed power reductions are 29%, 29%, 19%, and 20% for error rates of 2%, 4%, 8%, and 16% respectively (Section VI). Benefits are higher in the face of error recovery using Razor. Area overhead of our techniques is up to 2.7%.
- We show that power-aware slack redistribution can even result in increased throughput for a given voltage (Section VI-B). Benefits are due to reduced overhead of error recovery.
- Finally, we show that smoothening the critical wall through slack redistribution is more efficient than pushing the critical wall through tightly constrained SP&R. Benefits are both in terms of power and througput (Section VI).

The rest of our paper is organized as follows. Section II discusses related work. Section III formulates the optimization problem to be solved by our techniques to produce soft architectural designs that degrade gracefully in face of voltage scaling. Section IV presents details of our power-aware slack redistribution techniques. Section V discusses additional methodological details of this research. Section VI presents analysis and results, including a quantification of potential processor power savings. Section VII concludes.

II. Related Work

A. Better-than-worst-case Designs

A number of better-than-worst case (BTWC) designs have been proposed in the past that save power by eliminating guardbands. For example, Razor [5] and ANT-based designs [12] allow BTWC operation by tolerating errors at the circuit and algorithm-level, respectively. Their benefits in the context of voltage scaling are limited, however, by the error rate at a given voltage and the corresponding error recovery overheads of the techniques.

Another class of BTWC designs uses “canary” circuits to detect when arrival at the critical point is imminent, thus revealing the extent of safe scaling. Delay line speed detectors [4] work by propagating a signal transition down a path that is slightly longer than the critical path of a circuit. Scaling is allowed to proceed until the point where the transition no longer reaches the end of the delay line before the clock period expires. While this circuit enables scaling, no scaling is allowed past the critical path delay plus a safety margin.

Another similar circuit technique uses multiple latches which strobe a signal in close succession to locate the critical operating point of a design. The third latch of a triple-latch monitor [14] is always assumed to capture the correct value, while the first two latches indicate how close the current operating point is to the critical point. Again, the effectiveness of the technique in the context of general-purpose processor designs will be limited by the critical operating point behavior of the processor.

B. Design-level Optimizations for Timing Speculation

Design-level optimizations have recently been proposed [7] to improve throughput of timing speculation architectures. The idea is to identify and optimize the most frequently-exercised dynamic paths in a design at the expense of the majority of the static paths, which are allowed to suffer infrequent timing errors. EVAL [18] is a technique that trades error rate for processor frequency by shifting, tilting, or reshaping the path delay distributions of the various functional units. As an application of EVAL, the BlueShift technique [7] identifies timing paths that are most-often violated and optimizes them using on-demand selective biasing and path constraint tuning. On-demand selective biasing (OSB) involves adding slack to the most frequently violated paths by forward body biasing some of their gates. Path constraint tuning (PCT) involves adding slack to paths by applying strong timing constraints on them.

There are four major differences between our work and the BlueShift work. First, the optimization problem being solved is different. The goal of the BlueShift work is to maximize the frequency for a given error rate, while the goal of this work is to minimize voltage for a given error rate. Second, our sensitivity functions are different. While BlueShift optimizations are agnostic of voltage-dependence of delay for various timing paths, our work involves optimizing paths / cells according to different functions, including switching activity, amount of negative slack, and response of path delay to voltage scaling. As the results in Section VI show, using the BlueShift sensitivity functions to minimize power may not be very effective. Third, our optimization techniques are different. While BlueShift uses OSB and PCT, we use cell sizing. Section VI compares the PCT method against our technique (cell sizing) and shows the limited effectiveness of PCT for power optimizations.

Fourth and finally, there is a significant difference between the optimization flow of our approach and the BlueShift approach. BlueShift uses repetitive gate level simulations to get path profiles after making iterative improvements. This may be impractical with large, modern SOC designs, as the number of post-sizing, layout, extraction simulation steps is often limited by runtime constraints. In contrast, our approach needs only one simulation of the gate-level netlist to obtain switching information for use in optimization. Moreover, this simulation does not need delay information (SDF). This expedites runtime.

C. Cell Sizing

In our work, we use post-layout cell resizing (or cell swapping) as a technique for redistributing timing slack in a design to create a gradual slack distribution. Previous works have typically proposed cell sizing or swapping as a technique for power or area recovery, subject to maintaining a target clock frequency. For example, in the context of leakage power reduction, previous works [6], [20], [8], [9], [10], [13] seek to minimize the (leakage-weighted) positive timing slack for non-critical cell instances (cell instances on non timing-critical paths) such that maximum leakage reduction is obtained without degrading overall circuit performance. Perhaps the closest work is by Ghosh *et al* [32] who also explored cell sizing for power. Unlike Ghosh's paper, our objective is to reduce error rate with minimum cell swaps. Also, we provide a method to find frequently exercised paths in general design cases – toggle information of cells from pre-simulation.

To the best of our knowledge, ours is the first work to use cell sizing for switching activity-aware 'gradual slack' redistribution in the BTWC context.

	Error Rate (%)					
Module	1.0V	0.9V	0.8V	0.7V	0.6V	0.5V
lsu_dctl	0.00	0.23	8.60	29.46	45.13	54.90
lsu_qctl1	0.00	5.94	10.85	16.99	16.56	37.53
lsu_stb_ctl	0.00	0.08	0.65	5.19	11.79	22.38
sparc_exu_div	0.00	0.15	0.23	0.35	0.49	1.10
sparc_exu_ecl	0.00	3.31	10.97	87.08	88.93	73.03
sparc_ifu_dec	0.00	0.08	0.87	7.09	15.22	20.48
sparc_ifu_errdp	0.00	0.00	0.00	0.00	0.00	9.21
sparc_ifu_fcl	0.00	10.56	22.25	50.04	55.06	56.95
spu_ctl	0.00	0.00	0.00	1.30	2.96	35.53
tlu_mmu_ctl	0.00	0.01	0.02	0.06	0.14	0.19

TABLE I
TIMING VIOLATIONS IN VARIOUS T1 MODULES AT DIFFERENT INPUT VOLTAGES

III. THE DESIGN OPTIMIZATION PROBLEM

Before we detail the need and design-level techniques for power-aware slack redistribution, we discuss how present processor designs are limited in their ability to allow voltage vs. reliability tradeoffs. We then formally state the corresponding optimization problem.

Table I shows how timing errors increase for ten selected modules of the *OpenSPARC T1* processor [27] when voltage is decreased (methodological details are in Section V). Overwhelmingly, the modules demonstrate critical operating point behavior, i.e., for each module, the error rate increases dramatically when the voltage is scaled beyond a certain critical voltage value, and there is only a small range of voltages where the error rate is low. Some modules, like tlu_mmu_ctl, have more timing slack than others before reaching the critical point, and Table I does not cover a wide enough range of voltages to demonstrate the critical module behavior.

Table I also shows that after a certain voltage, the error rates may exceed a given target error threshold. A target threshold may represent the maximum allowable error rate for a given error tolerance mechanism. For example, an error rate of up to 1% may be allowable for traditional Razor-based designs [5], while maximum allowable error rates may be higher for algorithmic noise tolerance (ANT) techniques. If Razor is to be used for error recovery in the lsu_qctl1 module, for example, voltage cannot be scaled below 1.0V. Also, since modules follow a critical operating point behavior rather than a gradual degradation in reliability, switching to an error recovery technique that can tolerate 2% errors is not possible for lsu_qctl1. This effect is even more pronounced for modules like sparc_ifu_fcl, where the increase in error rate is even more drastic.

The goal of our design optimization, therefore, is to simultaneously minimize the voltages at which each given maximum allowable error rate is observed, thus maximizing the range over which voltage scaling is possible. Formally, the optimization problem can be stated as follows.

Given: a set of error rates $e_1, e_2, \cdots, e_n$ ($e_i < e_{i+1}, 1 < i < n$).

Find: Minimize $V_{i,k}$, where $V_{i,k}$ is the voltage at which the error rate is no more than e_i for design k.

Subject to: (1) For all i and k, $V_{i,k} \geq V_{(i+1),k}$; (2) $\exists$ K, s.t., for all i and k, $V_{i,K} \leq V_{i,k}$ where K is the optimized design.

The above formulation results in designs that allow voltage/reliability tradeoffs up to a point where the error rate is e_n. Design K is the optimal design.

Note that the above optimization can be performed in two different ways: (1) reduce the voltage value at which a module exhibits critical operating point behavior, or (2) optimize the module to eliminate the critical operating point behavior (i.e., there is now a gradual degradation in module reliability). In this paper, we focus on the latter, for the following two reasons. First, a soft architecture with a gradual degradation in reliability with voltage scaling would allow us to perform reliability/power tradeoffs by reducing voltage down to a point that produces maximum allowable errors for a given error tolerance mechanism, i.e., maximum number of timing errors than can be effectively and gainfully corrected by a given error-tolerance mechanism. This will allow us to maximize the power savings for a given error tolerance mechanism. Second, a soft architecture allows one to use a different error tolerance mechanism at different voltages, allowing deeper voltage scaling, since an appropriate mechanism can be selected based on observed error rate. Figure 2 demonstrates the goal of the optimization problem.

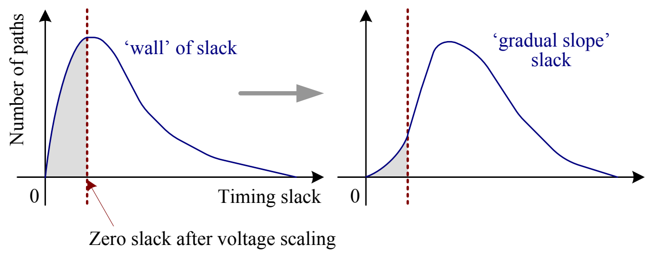


Fig. 2. The goal of the gradual slack optimization is to transform a slack distribution with critical wall behavior into one with a gradual failure characteristic.

We propose to solve this optimization problem based on power-aware slack redistribution, which is discussed in detail in the next section.

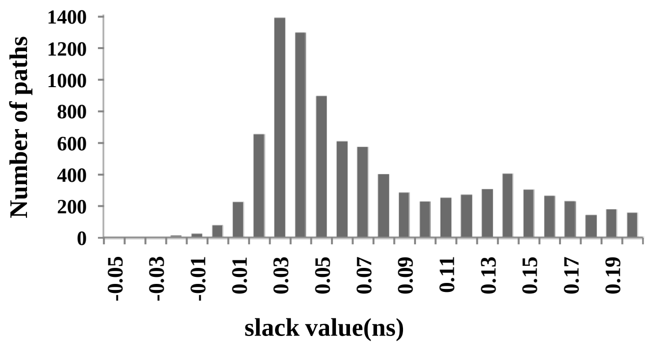


Fig. 3. Traditional design flow for high performance designs results in a critical operating point, with the slack of all paths bunched around the critical point. Scaling past the critical point results in a massive onset of timing violations.

IV. Power-aware Slack Distribution for Power/Reliability Tradeoffs

To understand the critical operating point behavior shown in Table I that our processor modules demonstrate, we generated the timing slack distributions for the various timing paths for these modules. Figure 3 shows the distribution for one of the processor modules we studied (*sparc_ifu_fcl*). As the figure shows, timing slack for the vast majority of paths of the design is close to that of the critical path. We observed the same behavior for other modules as well. This explains their critical operating point behavior – since path slack is bunched together for the majority of paths, scaling voltage past the critical path results in a massive onset of timing violations. The critical operating point of these individual modules causes the processor to show the same behavior.

The hypothesis of this work is that a power-aware redistribution of timing slack for a processor design can allow for better optimization of power efficiency at different error rates. If the onset of timing violations is made gradual, rather than the traditional wall, the range over which processor voltage scaling is possible (given error rate constraints) can be extended (see Figure 2), affording increased power savings for higher allowable error rates. In the following section, we present a technique that performs cell swapping to redistribute slack, producing a gradual, power-aware slack distribution for the timing paths of a processor.

A. Power-aware Slack Redistribution Using Cell Swap Method

Our slack distribution optimizer, implemented in C++, performs cell swapping with *Synopsys PrimeTime vB-2008.12-SP2* [29], using a *Tcl* socket interface. The optimization algorithm alters the timing slack distribution of a processor design to make the distribution more gradual.

In the traditional processor design methodology, all negative slack paths identified in static timing analysis must be optimized. In our approach, however, we focus our optimization on the frequently exercised paths with a negative timing slack at a target voltage, to minimize error rates in the face of voltage overscaling. This selective optimization targets benefits in both the performance and power of the design.

Specifically, our algorithm finds a target voltage corresponding to a specific error rate, and optimizes the frequently exercised paths intensively. In order to minimize power consumption after scaling the operating voltage, our slack optimizer must improve error rates while minimizing cell swaps. If we set an overly aggressive operating voltage point and optimize all negative slack paths at the target point, the optimizer can perform unnecessary swaps so that power consumption increases excessively.

To eliminate these unnecessary cell swaps, the target voltage for the slack optimizer must be selected correctly. In our approach, the slack optimizer finds a target voltage after estimating error rates at each operating voltage. At the initially selected voltage, the optimizer performs cell swaps to improve timing slack. After timing optimization at the initially selected voltage, the target voltage is scaled to the new lower voltage until the target error rate is reached. This heuristic optimizes paths and scales the voltage iteratively. With this approach, we can avoid excessive cell swaps while improving error rates.

For the appropriate voltage selection, the slack optimizer needs to forecast error rates without functional simulations. To estimate the error rates, we use toggle information for the data pins of flip-flops that have negative slack. The toggle information consists of two kinds of toggles – those from negative slack paths and those from positive slack paths. In the error rate calculation, only the toggles of negative slack paths should be considered. More details on our methodology for error rate estimation can be found in [35].

After finding a target voltage, the slack optimization algorithm collects negative slack paths by tracing backward from flip-flop cells using a depth–first search (DFS) algorithm. Collected paths are optimized, with priority given to frequently exercised paths. Our algorithm swaps cells in the paths with other library cells that have the same functionality. However, it cannot restore a previously swapped cell in order to recover to a previous configuration. Therefore, prioritizing the order of optimization is crucial. The priority is decided according to the switching activity of a path, defined as the minimum toggle rate of all cells in the path.

Cell swapping is performed on all cells in a path. If a cell has already been touched during optimization, the optimizer skips this cell. After performing a swap, the optimizer checks the timing slack of the path and rejects any move that makes the slack worse. If the path slack is improved, the optimizer checks the timing of connected fan-in and fan-out cells which have been touched previously in the optimization. Path information and initial slack are known for these cells, so we can check whether their slack is improved or degraded by the move. When there is no timing degradation in the connected neighboring cells, the cell change is finally accepted.

The cell swapping algorithm is iterated several times on a path until the path slack becomes positive or no further swaps are made. After finishing the target path optimization, affected cells are marked to prohibit further changes during optimization of other paths. The optimization is then performed on the next path in prioritized order. In this paper, the target voltage is scaled repeatedly by 0.01V until the error rate exceeds a target error rate. Then, the algorithm optimizes critical paths at the target voltage. If the power consumption is not reduced after the voltage scaling, the latest swaps are restored, and the optimizer is terminated.

We can also reduce leakage power by using the cell swapping method. This power reduction stage can be added at the end of the previous slack optimization. This procedure is the opposite of the previous slack optimizing algorithm. In the slack optimization, we choose cells in highly exercised, negative slack paths, and perform cell swapping to reduce delay. In the power reduction procedure, however, the cells in rarely exercised paths are chosen and swapped to reduce the power consumption while leaving error rate unaffected. Figure 4 shows the complete optimization process. More details can be found in [35].

Note that the slack redistribution technique described above considers voltage-dependence of delay. This creates timing paths with fewer timing errors at a given voltage. Since this optimization is performed throughout the processor, the processor itself will show reduced error rate at a given input voltage.

Note also that the above optimization is replacing cells on selected paths with faster but larger cells, so the new design may be larger. We believe, and the results in Section VI confirm, that the number of cells that need to be replaced with larger cells is small (as only a few paths are interesting), and therefore, there are net power savings for a given error rate at reduced voltages.

Finally, note that optimization is performed only on frequently occurring, near-critical paths. Since, a vast majority of timing paths are left untouched, we expect the area overhead of our technique to be small (Section VI confirms that the overhead is no more than 2.7%).

B. *Tightly Constrained SP&R*

Another potential approach for creating a gradual slack distribution is to perform traditional SP&R with an aggressive timing constraint. In this case, some paths will not meet the timing constraint, and the resulting slack distribution will be more gradual. Our baseline SP&R targets a frequency of 0.8 GHz. For the tightly constrained SP&R, we use a target frequency of 1.2 GHz and use the resulting design as a point of comparison for our gradual slack designs.

C. *BlueShift PCT*

We also compare our power-aware slack redistribution strategy against a BlueShift-like strategy for optimizing timing paths. BlueShift [7], as discussed in Section II, focuses on frequency overscaling for increased throughput benefits over traditional processor designs. It chooses paths to optimize based on the frequency of timing violations encountered during simulation. It iteratively optimizes the paths with the most timing violations until error rate targets are achieved.

We implement a BlueShift-like CAD flow as a point of comparison for our techniques. Our BlueShift implementation

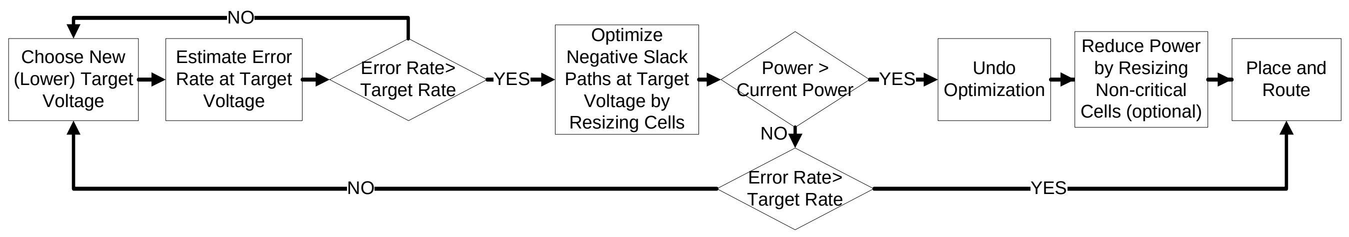

Choose New (Lower) Target Voltage
Estimate Error Rate at Target Voltage
Error Rate> Target Rate
NO
YES
Optimize Negative Slack Paths at Target Voltage by Resizing Cells
Power > Current Power
YES
NO
Undo Optimization
Reduce Power by Resizing Non-critical Cells (optional)
Place and Route
Error Rate> Target Rate
NO
YES

Fig. 4. Flowchart for the Slack Optimizer.

chooses paths to optimize based on the highest product of switching activity and negative slack. These paths will incur the most timing violations in the face of voltage scaling. For the selected paths, we specify tighter constraints using the *set_max_delay* command during P&R in *Cadence SoC Encounter*. We add the list to the *Synopsys Design Constraints (SDC)* file and apply 10%, 25%, 50% and 100% SDC iteratively.

V. Methodology

The goal of the paper is to show that power-aware slack redistribution can break the critical wall of a processor and create gracefully degrading processor designs that can operate at a much lower voltages before reaching the threshold error rate.

Our methodology for showing the above has two parts – a design-level methodology to characterize how timing error rate changes with voltage and an architecture-level methodology to estimate processor power and performance when the proposed design-level techniques are applied.

A. Design-level Methodology

Figure 5 shows our methodology flow diagram for the proposed slack optimizer. The optimizer selects paths for optimization based on switching activity and timing slack under voltage scaling. In order to find the frequently exercised paths, we use the switching activity interchange format (SAIF) file, which describes toggling frequency of each net and cell in the gate-level netlist. We perform gate-level simulation to produce a value change dump (VCD) file and convert the VCD to SAIF using *Synopsys PrimeTime-Px* [29]. To find timing slack and power values at the specific voltages, we prepare *Synopsys Liberty* (.lib) files for each voltage point – from 1.00V to 0.50V in 0.01V increments – using *Cadence SignalStorm TSI61* [30].

We use the *OpenSPARC T1* processor [27] to test our optimization framework and to gather switching information to be used by the optimizer. T1 is a chip multithreaded processor from Sun consisting of eight CPU cores where each core is 4-way multithreaded. The processor implements Sun's SPARC V9 instruction set. Since full system, gate-level simulation of this complex design would require an unreasonable amount of time, we instead sample modules from throughout the T1 for use in testing. We select modules used by a related work [7] to allow for more accurate comparisons. We also select additional

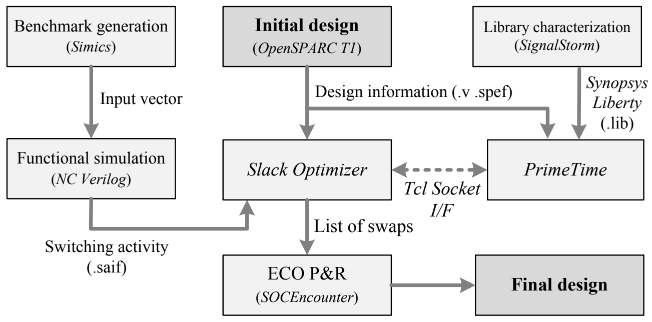

Benchmark generation (Simics)
Initial design (OpenSPARC T1)
Library characterization (SignalStorm)
Input vector
Design information (.v .spef)
Synopsys Liberty (.lib)
Functional simulation (NC Verilog)
Slack Optimizer
PrimeTime
Tcl Socket I/F
Switching activity (.saif)
List of swaps
ECO P&R (SOCEncounter)
Final design

Fig. 5. CAD flow incorporating the slack optimizer to create a design with a gradual slack distribution.

modules from different regions of the processor to obtain a more representative characterization of the processor.

Table II describes the selected modules and provides characterization in terms of cell count, area, and worst case negative slack for two design points implemented for different target frequencies. The slack optimizer targets frequently exercised paths for the first group of designs, which are implemented with a moderate clock frequency (0.8 GHz). The second group of designs, implemented for an aggressive clock frequency (1.2 GHz) are used as a comparison point against designs optimized for gradual slack. Note that the maximum range of critical delay difference between any two modules is 0.17ns, and the average deviation from the timing target is 0.068ns. This shows that the design is balanced with roughly equal critical timing paths.

For the selected modules, we perform gate-level simulation using test vectors gathered from full system, RTL simulation of a benchmark test set (details in Section V-A). Before gathering test vectors in the RTL simulation, we fast-forward each benchmark 1 billion instructions using Simics [16] Niagara. Simics is a full-system simulator used to run unmodified production binaries on the target hardware at high-performance speeds. Simics Niagara simulates the T1 processor. After fast-forwarding in Simics, the architectural state is transferred to the *OpenSPARC* RTL using the *CMU Transplant* tool [17]. The Transplant tool provides the capability for simulating portions of full-system workloads on the *OpenSPARC* RTL. The key idea is to "transplant architectural register and memory state from full-system functional simulators such as Simics to the RTL model. This process allows RTL simulation for workloads such as operating systems and databases that are otherwise

TABLE II
TIMING AND AREA CHARACTERISTICS OF TARGET MODULES.

Module	Stage	# of F/Fs	Description	SP&R with 0.8 GHz target			SP&R with 1.2 GHz target		
				Cell count	Area(um^2)	WNS(ns)	Cell count	Area(um^2)	WNS(ns)
lsu_dctl	MEM	672	L1 Dcache Control	4537	13850	0.067	4766	16387	-0.357
lsu_qctl1	MEM	372	LDST Queue Control	2485	7964	0.082	2740	9454	-0.230
lsu_stb_ctl	MEM	115	ST Buffer Control	854	2453	0.136	989	3443	-0.127
sparc_exu_div	EX	544	Integer Division	4809	14189	0.000	5135	16601	-0.384
sparc_exu_ecl	EX	351	Execution Unit Control Logic	2302	7089	0.103	2553	8703	-0.072
sparc_ifu_dec	FD	42	Instruction Decode	802	1737	0.183	892	2526	-0.048
sparc_ifu_errdp	FD	589	Error Datapath	4184	12972	0.145	4768	14497	-0.279
sparc_ifu_fcl	FD	280	L1 Icache and PC Control	2431	6457	0.113	2759	8719	-0.128
spu_ctl	SPU	430	Stream Processing Unit Control	3341	9853	0.102	3481	11662	-0.111
tlu_mmu_ctl	MEM	262	MMU Control	1701	5113	0.117	1788	6271	-0.286

too slow to simulate or require resources (e.g., I/O) that are not modeled in RTL." [17] In our workflow, the architectural transplant allows us to quickly seek to an interesting point in an application and transfer the state of the processor to the RTL simulator for more detailed simulation and test vector capture.

Switching activity gathered from gate-level simulation, along with design information such as timing slack and library cell characterization, are fed to the *Slack Optimizer* by *Synopsys PrimeTime (PT)* through a *Tcl Socket* interface. In order to obtain the timing slack and switching activity of critical paths, the optimizer accesses *PT* continuously during the optimization process. After optimization, the modified design is implemented using the Engineering Change Order (ECO) layout function of *Cadence SoC Encounter v7.1* [31].

Module designs are implemented with the 65GP library (65nm) using the traditional ASIC flow – synthesis with *Synopsys Design Compiler vY-2006.06-SP5* [28] and layout with *Cadence SoC Encounter*. In order to capture the voltage scaling effect on circuit behavior, we generate *Synopsys Liberty* libraries at several operating voltage levels. To expedite library characterization, we implement our testcases using a restricted library of 63 commonly used cells (62 combinational cells and 1 sequential cell). Reducing the library to essential, commonly used cells reduces the runtime of the optimizer and ensures compliance with the library characterization tool – *Cadence SignalStorm TSI61*.

We optimize module implementations in Table II (0.8 GHz) with the slack optimizer and check the error rate through gate-level simulation. Error rate is estimated by counting the timing failure cycles encountered during simulation. For precise estimation, we use a SCAN-like test, wherein the test vectors specify the value of each primary input and internal flip-flop for each cycle. This prevents erroneous signals from propagating to other registers and resulting in pessimistic error rates. In order to emulate the SCAN test, we connect all register output ports to the primary input ports, allowing full control of module state.

Note that while the goal of our design-level methodology is fidelity of results. the above methodology uses pessimistic STA and real P&R results making it relatively robust to process variation. Estimating the extent of robustness is a subject of future work.

B. Architecture-level Methodology

We use SMTSIM [21] integrated with Wattch [22] to simulate a processor whose parameters are in Table III. The simulator reports performance and power numbers at different voltages. All our evaluations are done using benchmarks in Table IV. These benchmarks were chosen to maximize diversity. We base our out-of-order processor microarchitecture model on the MIPS R10000 [23].

To get a processor-wide error rate at a given frequency and voltage, we first add up the error rates from all the *OpenSPARC* modules in Table II and then scale up the sum based on area such that it includes all modules that we believe can be optimized. The error rate of a module that has not been characterized is assumed to vary linearly with area. This is the same methodology as used in [7]. We believe that all modules can be optimized using our techniques other than array structures like registers and caches where all timing paths are equally likely to be exercised. Such modules are assumed to run, for a given voltage, at the highest frequency that produces no timing errors (870 MHz). Once the processor-wide error rate is calculated, we can use our simulator to estimate the throughput and power impact of errors for a given error recovery overhead.

We use a similar methodology to get processor-wide power numbers. To get a dynamic power estimate, we scale the dynamic power numbers reported by Wattch for the optimizable components by the ratio of total module power for an optimization technique over total module power for the baseline design, as reported by *Synopsys PrimeTime*. For the non-optimizable components, the Wattch numbers are scaled based on the maximum frequency that these components can run at without producing timing errors. For static power estimation, we use the ratio of dynamic and static module power for an optimization technique, as reported by *PrimeTime*, to determine static power for a given dynamic power determined using the above methodology.

When calculating processor power consumption with Razor error recovery in place, we scale the flip-flop power reported by *PrimeTime* to account for the increased power consumption of Razor flip-flops. Razor flip-flops consume higher power during normal operation and also introduce a power overhead when recovering from an error. We use the processor error rate, as formulated above, in conjunction with the rates of power consumption during normal operation and error recovery [5] to calculate the power overhead of Razor and determine total processor power consumption.

All our application simulations are done for 1 billion cycles after fast-forwarding to a Simpoint [24].

Property	Value
L1 Icache	16KB, 4-way, 1 cyc
L1 Dcache	16KB, 4-way, 1 cyc
L2	2MB, 8 way, 8 cyc
Execution	2-way OO
RegFile	72 (int), 72 (FP)
Branch Predictor	gshare (8K entries)
Memory Access	315 cyc

TABLE III
PROCESSOR SPECIFICATIONS.

VI. Analysis and Results

In this section, we present the results of our study, demonstrating that power-aware slack redistribution can extend the range of allowable voltage scaling, resulting in significant power benefits for a given error rate. We first demonstrate the benefits for individual processor modules. Then, we demonstrate processor-wide benefits and characterize the effect of power-aware slack redistribution on performance.

A. Voltage/Reliability Tradeoffs for Processor Modules

In these experiments, we use 10 submodules of the *OpenSPARC T1* processor [27] to test our optimization framework. We estimate error rates through gate-level simulations with different voltage libraries. Power consumption is also calculated for each operating voltage. We run experiments for four implementation cases at an operating frequency of 0.87GHz. This is the highest frequency at which no module produces timing errors.

1) Traditional P&R with a loose clock frequency target (0.8GHz)
2) Traditional P&R with a tight clock frequency target (1.2GHz)
3) *BlueShift PCT* [7]
4) Slack optimizer

Our slack optimizer includes the power reduction post-processing stage.

Figure 6 compares the power consumptions of the various design techniques at several target error rates (0.25%, 0.5%, 1%, 2%, 4%, 8% and 16%). Different error recovery mechanisms have different overheads of recovery. Thus, each mechanism can gainfully tolerate a different level of errors. One of the benefits of gradual slack designs is that they minimize the incidence of maximum acceptable error rates over a range of voltages, allowing for tradeoffs between power and error rate based on what is acceptable for a given application and recovery technique.

Although one proposed benefit of slack redistribution is power efficiency, path slack optimization can also incur costs in terms of power, due to cell resizing. We observed an average area overhead of 20.3% for tightly constrained P&R, 6.5% for BlueShift, and 2.7% for the slack optimizer (the power reduction process changes cells with smaller one, but there is no area reduction since we used ECO P&R to conserve the timing). Techniques like tightly constrained P&R and BlueShift do not consider the power implications of their optimizations and in many cases the power overhead of optimization outweighs the power benefit of voltage scaling, as evidenced in Figure 6. Power-aware slack redistribution, on the other hand, does well to reduce power consumption at the target error rates for the diverse set of processor modules, in spite of the slight area overhead (2.7%). In fact, only 5% cells were swapped, on average.

B. Processor-wide Voltage/Reliability Tradeoffs

We now examine the effectiveness of our slack redistribution techniques for a full processor design. Figure 7 illustrates how timing violations and power consumption vary with voltage for processors designed from the ground up, using the design methodologies described in Section IV.

The top graph in Figure 7 demonstrates the benefits of gradual slack design in extending the range of voltage scaling by reducing the error rate for a given voltage. While the power-aware slack redistribution techniques show substantial reductions in error rate, they do not always produce the lowest error rate for a given voltage. However, techniques with comparable error rates have much higher power and area overheads, as demonstrated by the bottom graph in Figure 7. Note that the power graph (the top graph) does not assume any error recovery overhead, so some designs have more errors than others, as evidenced in the error graph (the bottom graph).

The extended voltage scaling afforded by gradual slack designs allows for power reductions at a given target error rate. Figures 8 and 9 show processor power consumption at different error rates, demonstrating the potential of power-aware slack redistribution to significantly reduce power consumption. The power-aware techniques result in a superior design for two reasons.

- The slack optimizer extends the range of voltage scaling by reshaping the slack distribution of the optimized modules. This translates into more power savings for the same error rate when compared to the other techniques, since slack optimized designs can be operated at lower voltages while achieving the same error rate.
- The slack optimizer makes cost-effective optimizations. Power savings due to aggressive voltage scaling afforded by the slack optimizer outweigh the power overhead of slack optimization. This is not surprising since the area overhead of our techniques is limited to 2.7%. This also

Benchmark	Description	Benchmark	Description
ammp	Computational Chemistry	mcf	Combinatorial Optimization
applu	Parabolic / Elliptic Partial Differential Equations	mgrid	Multi-grid Solver: 3D Potential Field
art	Image Recognition / Neural Networks	parser	Word Processing
bzip2	Compression	swim	Shallow Water Modeling
crafty	Game Playing: Chess	twolf	Place and Route Simulator
eon	Computer Visualization	vortex-2	Object-oriented Database
equake	Seismic Wave Propagation Simulation	vpr	FPGA Circuit Placement and Routing
wupwise	Physics/ Quantum Chromodynamics		

TABLE IV
BENCHMARKS.

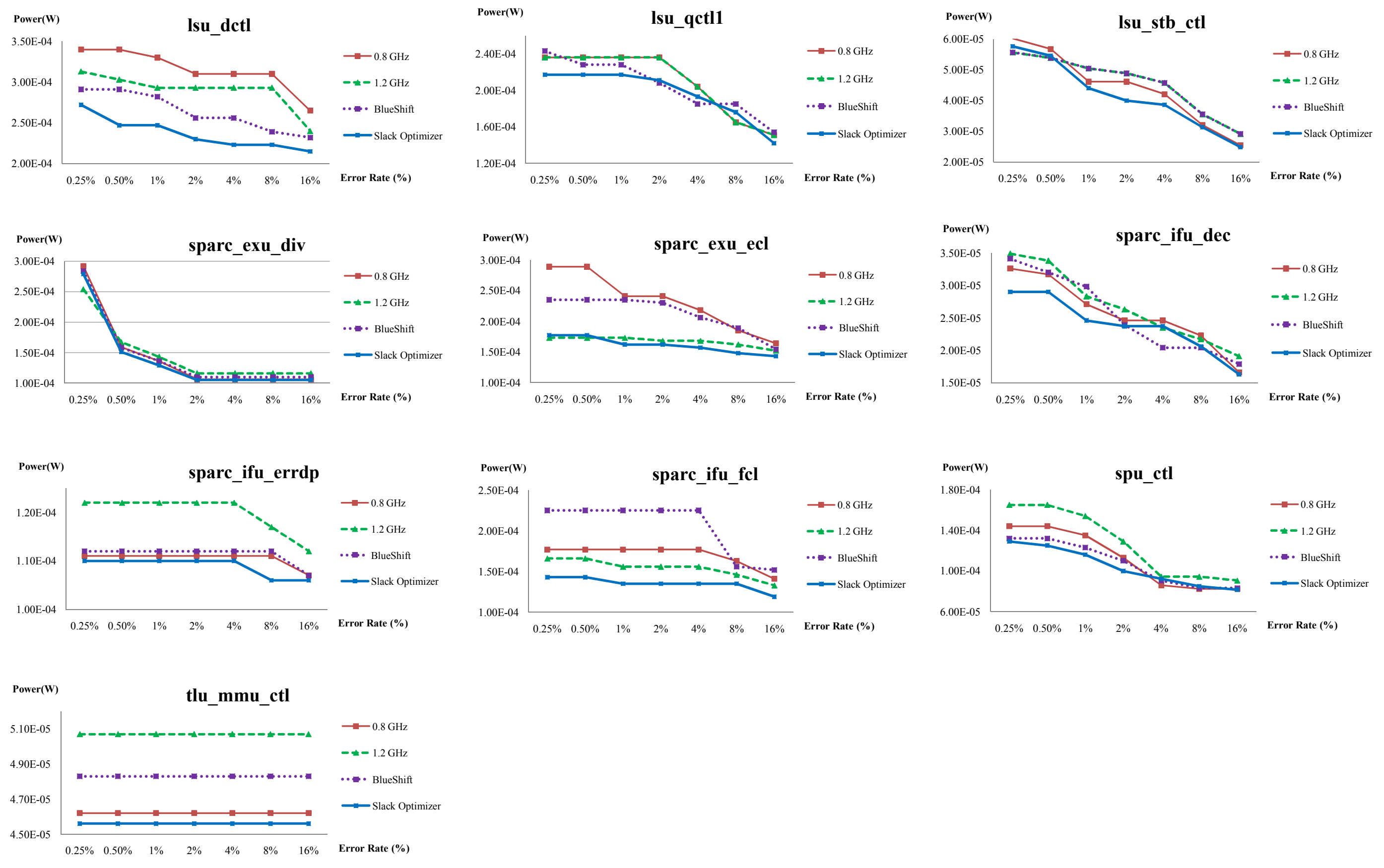


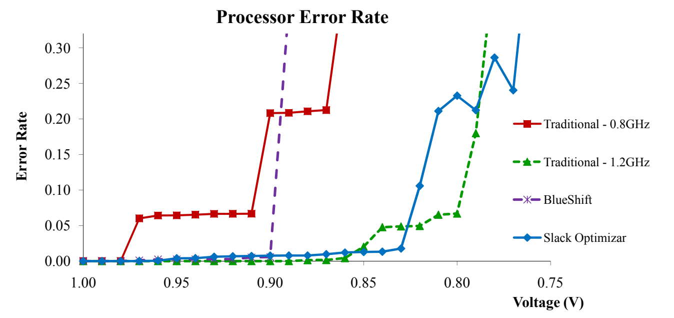
Fig. 6. Power consumption of each design technique at various target error rates (operating frequency is 0.87 GHz).

explains why some techniques that achieve lower error rates than the power-aware slack optimizer for some voltages still consume more power for a given error rate.

Note that the benefits are smaller relative to BlueShift for a small number of very low non-zero error rates, due to the aggressiveness of the BlueShift technique. However, power-aware slack optimization creates a gradual slack distribution, whereas BlueShift optimizes target paths heavily at the expense of other timing paths in the design. After 0.9V, the error rate of the BlueShift design shoots up, while the error rate of the slack optimized design continues to increase gradually, causing the two curves to diverge on the power vs. error rate plane

While the power graph in Figure 7 shows the effect of voltage scaling on processor power consumption as well as the relative ordering of techniques in terms of their power overhead, it does not assume any error recovery overhead. Figure 10 shows how processor power consumption varies with voltage scaling when Razor is used to detect and correct errors. This graph incorporates power overheads due to the increased power consumption of Razor flip-flops during normal operation and the power overhead of recovery incurred when Razor detects and corrects an error.

Figure 10 demonstrates that power-aware slack redistribution achieves the lowest power of any technique and extends the range over which voltage scaling and Razor correction are feasible. Note that in the baseline design (P&R with a frequency target of 0.8GHz), minimum power with Razor



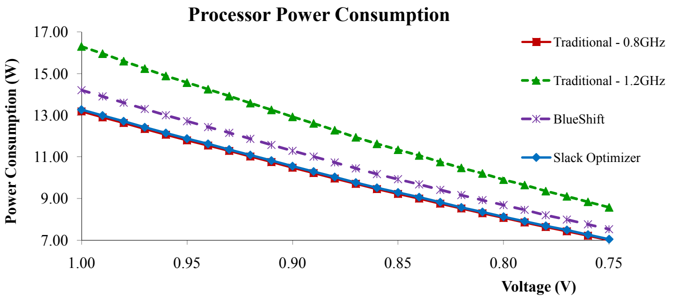


Fig. 7. Gradual slack designs have fewer errors than traditional processor designs as voltage is scaled down (operating frequency is 0.87 GHz). Although the power-aware slack redistribution techniques do not always result in the fewest timing violations for a particular voltage (top), techniques that have comparable error rates also have much higher power overheads (bottom).

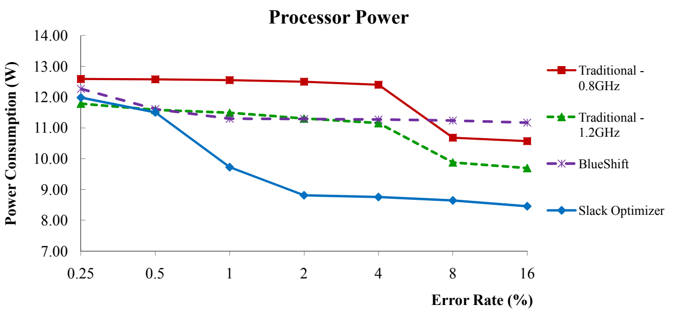


Fig. 8. The power-aware slack redistribution techniques result in the lowest power consumption over the entire range of error rates. Power reductions can be attributed to efficient error elimination strategies that enable extended voltage scaling without adding significant area and power overhead.

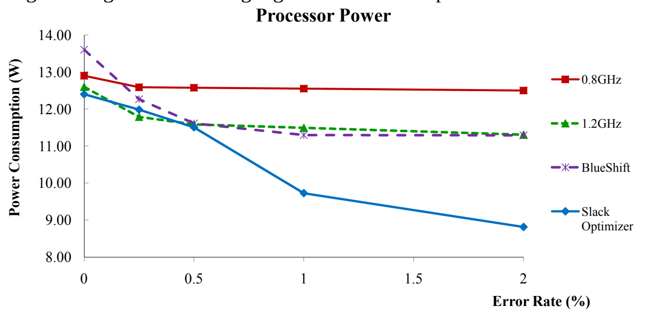


Fig. 9. Power vs. error rate for error rates less than 2%. The slack optimizer has lower power than BlueShift for an error rate of 0% because the power-aware slack redistribution algorithm results in a design with less area overhead than the BlueShift approach. For high voltages (very low error rates), BlueShift results in a slightly lower error rates than the slack optimizer, since it optimizes cells more aggressively. At higher error rates, the slack optimizer performs better due to gradual slack distribution.

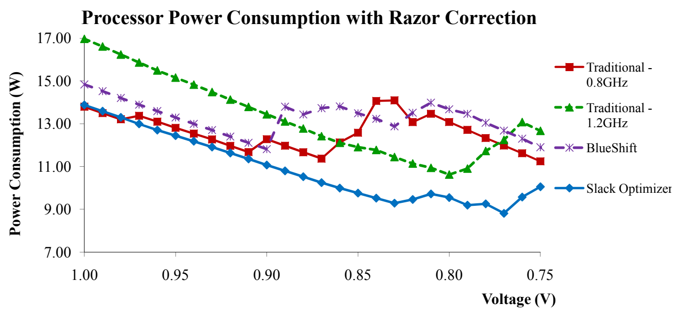


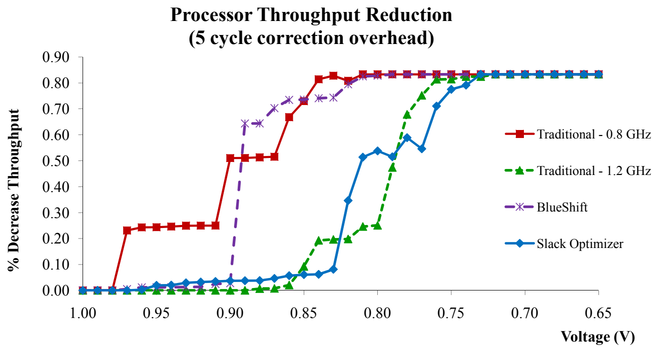
Fig. 10. When Razor is used to detect and correct timing errors, power increases due to increased power consumption in Razor flip-flops as well as the power costs associated with error detection and correction. Even so, power-aware slack redistribution minimizes power consumption in the face of errors by reducing timing violations that trigger error recoveries in an area- and power-efficient manner.

(11.4W) is achieved at 0.87V. For the slack optimizer, the overhead of error recovery is reduced due to reduced number of timing violations. So, the minimum power (9.0W) is achieved only at 0.77V. This represents an *additional* reduction in total power of 21% and a total power reduction of 35% with respect to the baseline design. Note that slack optimization results in the minimum power design over the entire range of voltages, even at near-nominal voltages, where the error rate is low.

Sub-critical voltage operation also incurs a performance penalty when error recovery is considered. Although throughput does not suffer directly, since frequency is not scaled down along with voltage, each error recovery technique has an associated performance overhead represented by the time it takes to correct an error and restore normal operation. For Razor, this overhead is about 5 cycles from error detection to error correction. Figure 11 shows the average effect of error recovery overhead on processor throughput for our workload of SPEC benchmarks as voltage is scaled down on each processor design. The results show that the slack redistribution-based techniques often have higher throughput than their traditional counterparts. This is due to smaller aggregate error recovery overhead in most voltage ranges. Even in the voltage ranges where throughput degradation is higher than tightly constrained SP&R, there is a power efficiency win due to the reasons described above.

VII. Summary and Conclusion

In this paper, we proposed power-aware slack redistribution, a novel approach to enable extended voltage/reliability tradeoffs in processors. Our techniques reapportion timing slack in a power- and area-efficient manner, such that we increase the range of voltages over which the incidence of operational (timing) errors is acceptable. This results in soft architectures, i.e., designs that fail gracefully, allowing us to perform reliability/power tradeoffs by reducing voltage down to the point that produces maximum allowable errors for our application without inducing catastrophic failure. We demonstrated the benefits of such designs in terms of power efficiency and extended range of voltage scaling before encountering a target error rate. Our experiments demonstrate 23% power savings



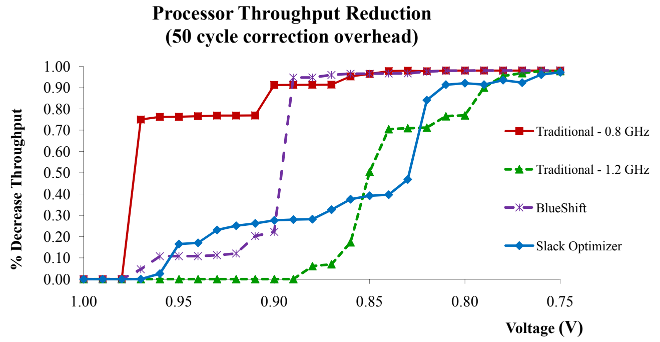


Fig. 11. The reduced error recovery overheads afforded by gradual slack designs result in throughput gains over the baseline design. Throughput degradation for the baseline design spikes due to a massive onset of errors. For gradual slack designs, throughput degradation increases gradually as voltage is scaled down.

over the baseline design at an error rate of 1%. Observed power reductions are 29%, 29%, 19%, and 20% for error rates of 2%, 4%, 8%, and 16% respectively. Benefits are higher in the face of error recovery using Razor. Area overhead of our techniques is up to 2.7%.

VIII. Acknowledgments

Rakesh Kumar would like to acknowledge support by Intel, NSF, and GSRC. Feedback from Janak Patel, Naresh Shanbhag, Steve Lumetta, Sanjay Patel, and anonymous reviewers also helped improve this paper.

References

[1] T. Austin, V. Bertacco, D. Blaauw and T. Mudge, "Oppotunities and Challenges for Better Than Worst-Case Design", *Proc. Asia South Pacific Design Automation*, 2005, pp. 2–7.

[2] T. D. Burd, S. Member, T. A. Pering, A. J. Stratakos and R. W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System", *IEEE Journal of Solid-State Circuits*, 35(11), 2000, pp. 1571–1580.

[3] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem and B. Seshasayee, "Ultra-Efficient (Embedded) SOC Architectures Based on Probabilistic CMOS (PCMOS) Technology", *Proc. Design Automation and Test in Europe*, 2006, pp. 1110-1115.

[4] S. Dhar, D. Maksimović and B. Kranzen, "Closed-Loop Adaptive Voltage Scaling Controller for Standard-Cell Asics", *IEEE/ACM Proc. International Symposium on Low Power Electronics and Design*, 2002, pp.103–107.

[5] 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 Lowpower Pipeline Based on Circuit-Level Timing Speculation", *IEEE/ACM Proc. International Symposium on Microarchitecture*, 2003, pp. 7–18.

[6] J. P. Fishburn and A. E. Dunlop, "Tilos: A Polynomial Programming Approach to Transistor Sizing", *IEEE/ACM Proc. International Conference on Computer-Aided Design*, 1985, pp. 326–328.

[7] B. Greskamp, L. Wan, W. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen and C. Zilles, "Blueshift: Designing Processors for Timing Speculation from the Ground Up", *International Symposium on High Performance Computer Architecture*, 2009, pp. 213–224.

[8] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Selective Gate-length Biasing for Cost-Effective Runtime Leakage Control", *IEEE/ACM Proc. of Design Automation Conference*, 2004, pp. 327–330.

[9] P. Gupta, A. Kahng and P. Sharma, "A Practical Transistor-Level Dual Threshold Voltage Assignment Methodology", *Proc. of International Symposium on Quality Electronic Design*, 2005, pp. 421–426.

[10] P. Gupta, A. Kahng, P. Sharma and D. Sylvester, "Gate-Length Biasing for Runtime-Leakage Control", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(8), 2006, pp. 1475–1485.

[11] V. Gutnik and A. Chandrakasan, "An Efficient Controller for Variable Supply-Voltage Low Power Processing", *IEEE Proc. Symposium on VLSI Circuits*, 1996, pp. 158–159.

[12] R. Hegde and N. R. Shanbhag, "Energy-Efficient Signal Processing via Algorithmic Noise-Tolerance", *Proc. International Symposium on Low Power Electronics and Design*, 1999, pp. 30–35.

[13] K. Jeong, A. B. Kahng and H. Yao, "Revisiting the Linear Programming Framework for Leakage Power vs. Performance Optimization", *Proc. Proc. of International Symposium on Quality Electronic Design*, 2009, pp. 127–134.

[14] T. Kehl, "Hardware Self-Tuning and Circuit Performance Monitoring", *IEEE International Conference on Computer Deisgn*, 1993, pp. 188–192.

[15] S. Narayanan, G. Lyle, R. Kumar and D. Jones, "Testing the Critical Operating Point (COP) Hypothesis using FPGA Emulation of Timing Errors in Over-Scaled Soft-Processors", *IEEE Workshop on Silicon Errors in Logic - System Effects*, 2009.

[16] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt and B. Werner, "Simics: A Full System Simulation Platform", *Computer*, 35(2), 2002, pp. 50–58.

[17] E. Chung and J. Smolens, *OpenSPARC T1: Architectural Transplants.* http://transplant.sunsource.net/.

[18] S. Sarangi, B. Greskamp, A. Tiwari and J. Torrellas, "EVAL: Utilizing Processors with Variation-Induced Timing Errors", *IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 423–434.

[19] J. Patel, "CMOS Process Variations: A Critical Operation Point Hypothesis", *Online Presentation*, 2008.

[20] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda and D. Blaauw, "Duet: An Accurate Leakage Estimation and Optimization Tool for Dual-Vt Circuits", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 10(2), 2002, pp. 79–90.

[21] D.M. Tullsen, "Simulation and Modeling of a Simultaneous Multithreading Processor", *22nd Annual Computer Measurement Group Conference*, December 2006.

[22] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", *International Symposium on Computer Architecture*, June 2000.

[23] K. Yeager, "The MIPS R10000 Superscalar Microprocessor", *IEEE Micro*, Vol 16, Number 2, 1996.

[24] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, "Automatically Characterizing Large Scale Program Behavior", *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[25] Intel Corporation, "Enhanced Intel Speed Step Technology for the Intel Pentium M Processor", *Microproceesors White Papers*, 2004.

[26] *International Technology Roadmap for Semiconductors 2008 Update.*, http://www.itrs.net/.

[27] *Sun OpenSPARC Project.*, http://www.sun.com/processors/opensparc/.

[28] *Synopsys Design Compiler User's Manual.* http://www.synopsys.com/.

[29] *Synopsys PrimeTime User's Manual.* http://www.synopsys.com/.

[30] *Cadence SignalStorm User's Manual.* http://www.cadence.com/.

[31] *Cadence SOCEncounter User's Manual.* http://www.cadence.com/.

[32] S. Ghosh and K. Roy, "Crista: An Integrated Technique for Voltage-Overscaling and Error Resiliency", March 2008.

[33] R. Kumar,"Stochastic Processors", *NSF Workshop on Science of Power Management*, March 2009.

[34] S. Narayanan, J. Sartori, R. Kumar and D. Jones, "Scalable Stochastic Processors", to appear in *Design and Test in Europe (DATE)*, March 2010.

[35] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Slack Redistribution for Graceful Degradation Under Voltage Overscaling", to appear in *Proc. Asia and South Pacific Design Automation Conf.*, January 2010.