RECOVERY-DRIVEN DESIGN: EXPLOITING ERROR RESILIENCE IN DESIGN OF ENERGY-EFFICIENT PROCESSORS

BY

JOHN M. SARTORI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2010

Urbana, Illinois

Adviser:

Assistant Professor Rakesh Kumar

ABSTRACT

Conventional CAD methodologies optimize a processor module for correct operation and prohibit timing violations during nominal operation. We propose *recovery-driven design*, a design approach that optimizes a processor module for a target timing error rate instead of correct operation. The target error rate is chosen based on how many errors can be gainfully tolerated by a hardware or software error resilience mechanism. We show that significant power benefits are possible from a recovery-driven design approach that deliberately allows errors caused by voltage overscaling to occur during nominal operation, while relying on an error resilience technique to tolerate these errors. We present a detailed evaluation and analysis of such a design-level methodology that minimizes the power of a processor module for a target error rate. We show how this design-level methodology can be extended to design *recovery-driven processors* – processors that are optimized to take advantage of hardware or software error resilience. These may be single-core processors or *heterogeneously-reliable multi-core processors*, in which individual cores are optimized for different reliability targets. We also discuss a *gradual slack* recovery-driven design approach that optimizes for a range of error rates to create *soft processors* – processors that have graceful failure characteristics and the ability to trade throughput or output quality for additional energy savings over a range of error rates. We demonstrate significant power benefits over conventional design – 11.8% on average over all modules and error rate targets, and up to 29.1% for individual modules. Processor-level benefits are 19.0%, on average. Benefits increase when recovery-driven design is coupled with an error resilience mechanism or when the number of available voltage domains increases.

ACKNOWLEDGMENTS

Many thanks to Seokhyeong Kang, PhD Student at UC San Diego, for the long hours spent working together, and to Andrew B. Kahng, Professor of CSE and ECE at UC San Diego, for demanding excellence.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Conventional hardware is designed and optimized using techniques that aim to ensure correct operation of the hardware under different conditions. Conservative design techniques are aimed at ensuring correct hardware operation under worst-case conditions. Better-than-worst-case design techniques [1] save power by eliminating guardbands, but are still aimed at ensuring correct hardware operation under nominal conditions.

In this research, we ask the following question: *Should the availability of an error resilience mechanism change the way we approach hardware design and optimization?* I.e., given that mechanisms exist to tolerate hardware errors, should hardware continue to be designed for correct operation or should it be optimized for a target error rate even during nominal operation? To address this question, we propose and evaluate a novel approach to hardware design, called recovery-driven design. Rather than optimizing for correct operation, a recovery-driven design deliberately allows timing errors ([16],[8]) to occur during nominal operation, while relying on an error resilience mechanism to tolerate these errors. In other words, a recovery-driven design optimizes a circuit for a non-zero target error rate that can be gainfully tolerated by hardware [8] or software-based [16] error resilience. The expectation behind recovery-driven design is that the "underdesigned" hardware will have significantly lower power or higher performance than hardware optimized for correct operation. Also, because errors are now allowed, the design methodology can exploit workload-specific information (e.g, activity of timing paths, architecture-level criticality of timing errors, etc.) to further maximize the power / performance benefits of underdesign.

In this paper, we show that optimizing power for a target timing error rate for voltage overscaling-induced errors indeed results in significant power savings for similar levels of performance. We show that this is true when errors are detected and corrected by a hardware error tolerance mechanism [8]

or allowed to propagate to an error-tolerant application [6] where the errors manifest themselves as reduced performance or output quality [16]. Increasing the target error rate for a processor module increases the potential for power savings, since the module can be operated at a lower voltage. In practice, the target error rate is chosen such that an error recovery mechanism can correct the resulting errors and still reduce energy (after considering the error recovery overhead) for an acceptable degradation in performance or output quality. The power benefits of exploiting error resilience are maximized by redistributing timing slack from paths that cause very few errors to frequently-exercised paths that have the potential to cause many errors. This reduces the error rate at a given voltage, and hence reduces the minimum supply voltage and power for a target error rate.

This paper presents a detailed evaluation and analysis of a slack redistribution-based recovery-driven design methodology that minimizes the power of a processor module for a target error rate. Our cell sizing-based design-level methodology has been extended to create recovery-driven processors that are optimized for different target error rates or error-resilience mechanisms. Since some error resilience mechanisms (e.g. error-tolerant applications) require adaptation to multiple reliability targets, we have also extended our recovery-driven design approach to create *gradual slack* designs – designs that are optimized not for a single error rate, but instead, for a range of error rates. Such gradual slack designs (or *soft processors*) have the ability to trade performance or output quality for energy savings over a range of reliability targets. We make the following contributions in this paper.

- To the best of our knowledge, we present the first design flow for power minimization that deliberately allows errors under nominal conditions. We demonstrate that such a design flow can result in power savings of 11.8%, on average over all modules and error rate targets, and up to 29.1% for individual modules.

- We explore the heuristic choices and tradeoffs that are fundamental to the optimization quality of slack redistribution-based, recovery-driven designs. We evaluate choices for path priority and traversal during optimization, optimization radius, accuracy of path selection, error budget utilization, starting netlist, voltage step size granularity, and iterative

optimization in terms of their effects on the optimization result, heuristic runtime, and sensitivity to target error rate.

- To support the proposed recovery-driven design flow, we present a fast and accurate technique for post-layout activity and error rate estimation. We use collected functional information to redistribute slack efficiently in a circuit and significantly extend the range of voltage scaling for a target error rate.

- We extend our recovery-driven design methodology to create *recovery-driven processors* (processors that are optimized for different target error rates or error recovery mechanisms) and *soft processors* (processors that are optimized for efficiency over a range of target error rates). We demonstrate the power and energy benefits of such processor designs.

- We demonstrate that the power benefits of recovery-driven processors and soft processors increase when a hardware or software-based error resilience mechanism is used. We consider Razor [8] and application-level noise tolerance [31] as examples and show additional energy reductions of 19% and 20% with respect to the best correctness-optimized processors that exploit the same error resilience mechanisms.

- We make a case for *heterogeneously-reliable multi-core processors* by demonstrating that a heterogeneous processor can achieve substantial power benefits over homogeneous processors for a diverse set of applications. An example heterogeneously-reliable dual-core CMP has an energy-delay product (EDP) benefit of 16% over the best homogeneous CMP design for a workload consisting of both error-tolerant and error-intolerant applications. Benefits are 26% and 32% for homogeneous workloads with error tolerance and homogeneous workloads with no error tolerance, respectively.

CHAPTER 2

RELATED WORK

2.1 Design-Level Optimization

Previous design-level optimizations for error-tolerant designs ([12], [11]) identify and optimize critical paths that are frequently exercised during operation. *BlueShift* [12] identifies the most frequently violated timing paths during gate-level simulation, and optimizes the paths iteratively until the error rate is below the target. *BlueShift* uses two methods to add slack to the frequently-exercised paths – forward body biasing of selected gates and application of tighter timing constraints to the frequently-exercised paths.

Our work differs from *BlueShift* in objective, approach, and scope of optimization. Our objective is to minimize power, while *BlueShift*'s objective is to improve performance. Consequently, we use sensitivity functions that are voltage-aware. Also, *BlueShift* requires iterative gate-level simulation and relayout, making the approach time-consuming and impractical for large SOC designs. Furthermore, while *BlueShift* optimizes only the post-synthesis circuit over many layout iterations, our recovery-driven design techniques perform both activity-guided post-synthesis and post-layout optimizations in a single pass to enhance energy efficiency.

CRISTA [11] isolates critical paths with Shannon-expansion-based partitioning. After partitioning, *CRISTA* downsizes cells on the critical path and upsizes cells on the non-critical paths: critical paths are made slower while non-critical paths are made faster. When a critical path is excited, the corresponding operation takes two cycles.

CRISTA changes the structure of the original circuit and also requires circuit-specific design to isolate critical paths. Since we do not change the original circuit structure, our techniques are more general in nature and can be applied more easily to a wider range of circuits.

2.2 Sensitivity-Based Cell Sizing

Our methodology relies on cell sizing for slack distribution. Sensitivity-based downsizing approaches have been proposed in [10], [24], [25], [15], [13], and [14]. *TILOS* [10] proposes a heuristic that sizes transistors iteratively, according to the sensitivity of the critical path delay to the transistor sizes, in order to find an optimum (with maximum delay reduction / transistor width increase). Equation (2.1) shows the sensitivity function of *TILOS*. ΔL and ΔD represent the change in leakage and delay for a resized transistor. The techniques proposed in [25] use the same sensitivity function as $TILOS$.

$$Sensitivity = \Delta L / \Delta D \tag{2.1}$$

For the cell sizing in [15], all cells are sorted in decreasing order of $\Delta L \times S$ (Equation (2.2)), where ΔL is the improvement in leakage after a cell is replaced with its less leaky variant, and S is its timing slack after the replacement has been made.

$$Sensitivity = \Delta L \times S \tag{2.2}$$

The techniques proposed in [13] and [14] use sensitivity-based *downsizing* (i.e., begin with all nominal cell variants and replace cells on non-critical paths with long channel-length variants) heuristics for leakage optimization. In their heuristics, they defined the sensitivity associated with cell instance.

$$Sensitivity = \Delta L / \Delta S \tag{2.3}$$

In Equation (2.3), ΔS represents the slack change of a given cell instance after downsizing. ΔL indicates the leakage change of cell instance after downsizing. The sensitivities are computed for all cell instances. The heuristics of [13],[14] select a cell with the largest sensitivity and perform downsizing with a logically equivalent cell. If there is no timing violation in incremental STA, this move is accepted and saved.

Our work uses cell sizing in a novel context – as a mechanism to optimize hardware for non-zero error rates.

CHAPTER 3

HEURISTIC DESIGN

3.1 Motivation

The goal of recovery-driven design in context of voltage overscaling can be stated formally as follows. Given an initial netlist N_0, a set of cell libraries characterized for allowable operating voltages, toggle rates for the toggled paths in the netlist, and a target error rate ER_{target}, produce the optimized netlist $N_{V_{opt}}$ and operating voltage V_{opt} that minimize the total power consumption $W_{V_{opt}}$ of the circuit, such that the error rate of the optimized netlist does not exceed ER_{target}. Figure 3.1 demonstrates the goal.

In this paper, we present a cell sizing-based design methodology that relies on efficient redistribution of timing slack from infrequently-exercised critical paths to frequently-exercised paths to reduce the error rate at a given voltage, allowing a reduction in voltage for a given target error rate.

3.2 An Abstract Heuristic for Power Minimization

Our heuristic for slack redistribution-based power minimization uses a two-pronged approach – extended voltage scaling through cell upsizing on critical and frequently-exercised circuit paths (*OptimizePaths*), and leakage power reduction achieved by downsizing cells in non-critical and infrequently-exercised paths (*ReducePower*). The heuristic searches for the combination of the two techniques that results in the lowest total power consumption for the circuit, by performing path optimization and power reduction at each voltage step and then choosing the operating power at which minimum power is observed.

Figure 3.2 illustrates the evolution of the circuit path slack distribution throughout the stages of the power minimization procedure. Each iteration

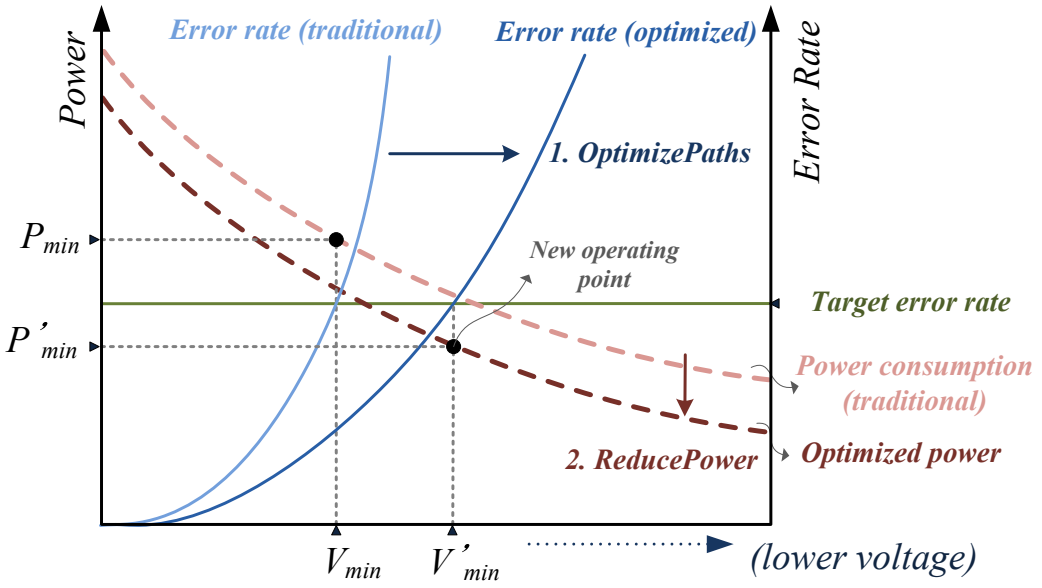

Figure 3.1: Our recovery-driven design optimization redistributes slack from infrequently-exercised paths to frequently-exercised paths and performs cell downsizing for average-case conditions. These optimizations reduce the power consumption of a circuit and extend the range that voltage can be scaled before a target error rate is exceeded. The combination of these factors produces a design with significantly reduced power consumption.

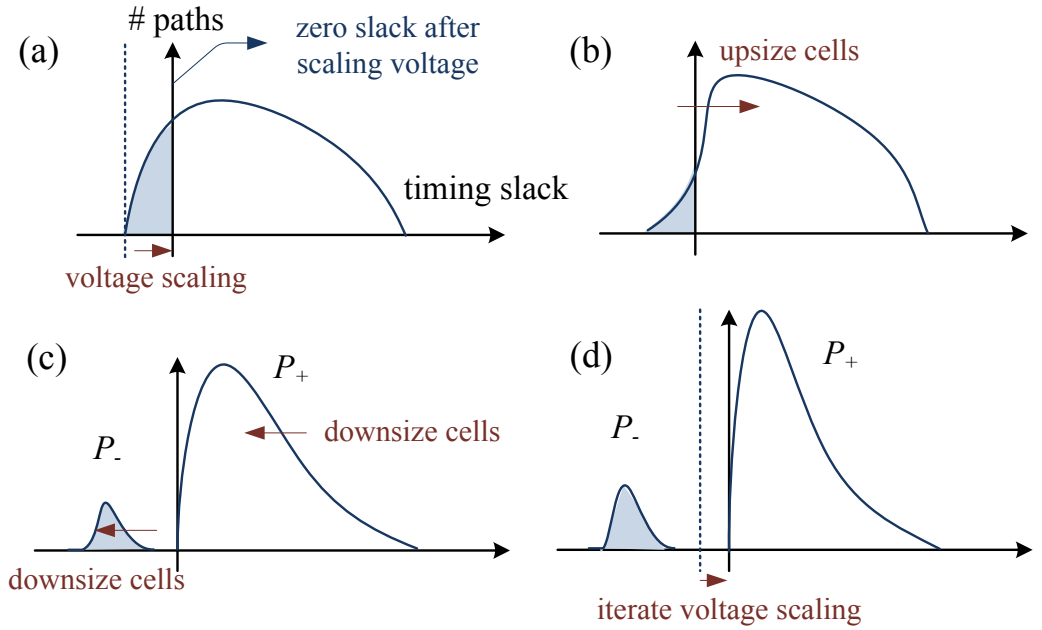

Figure 3.2: The power minimization heuristic reshapes the path slack distribution by redistributing slack from paths that rarely toggle to paths that toggle frequently.

begins as voltage is scaled down by one step (a). After partitioning the paths into sets containing positive and negative-slack paths, *OptimizePaths* attempts to reduce the error rate by increasing timing slack on negative-slack paths (b). Next, the heuristic allocates the error rate budget by selecting paths to be added to the set of negative-slack paths, and downsizes cells to achieve area / power reduction while respecting the partition between positive- and negative-slack paths (c). This cycle is repeated over the range of voltages to find the minimum power netlist and corresponding voltage (d). In Figure 3.2, P_+ is a set of paths that must have non-negative slack after power reduction, and P_- is a set of paths that are allowed to have negative slack.

Figure 3.3 presents the algorithmic flow of our power minimization heuristic, which couples path optimization to extend the range of voltage scaling (*OptimizePaths*) with area minimization to achieve power reduction (*ReducePower*).

3.3 Heuristic Procedures

Path Optimization. The goal of the path optimization procedure (*OptimizePaths*) presented in Algorithm 1 is to minimize the error rate at a voltage level by transforming negative slack paths into non-negative-slack paths. This is accomplished by performing cell swaps that upsize cells in the negative slack paths and increase the path slack. Negative slack paths with maximum toggle rates are selected first during optimization, since they have the most potential to reduce the error rate if converted into positive-slack paths.

When a path is targeted for optimization, upsizing cell swaps are attempted on all cells in the path to increase slack as much as possible until non-negative path slack is achieved.[1] Once a cell has been visited during optimization, it is marked to prevent degradation of timing slack on any path that the cell is on. Before accepting a cell swap, path slack is checked for all paths that the cell or any visited fanin / fanout cell is on. If the swap caused a decrease in slack for any such path, the move is rejected, and the original cell is restored.

[1]We consider only setup timing slack, since hold violations can typically be fixed by inserting hold buffers in a later step.

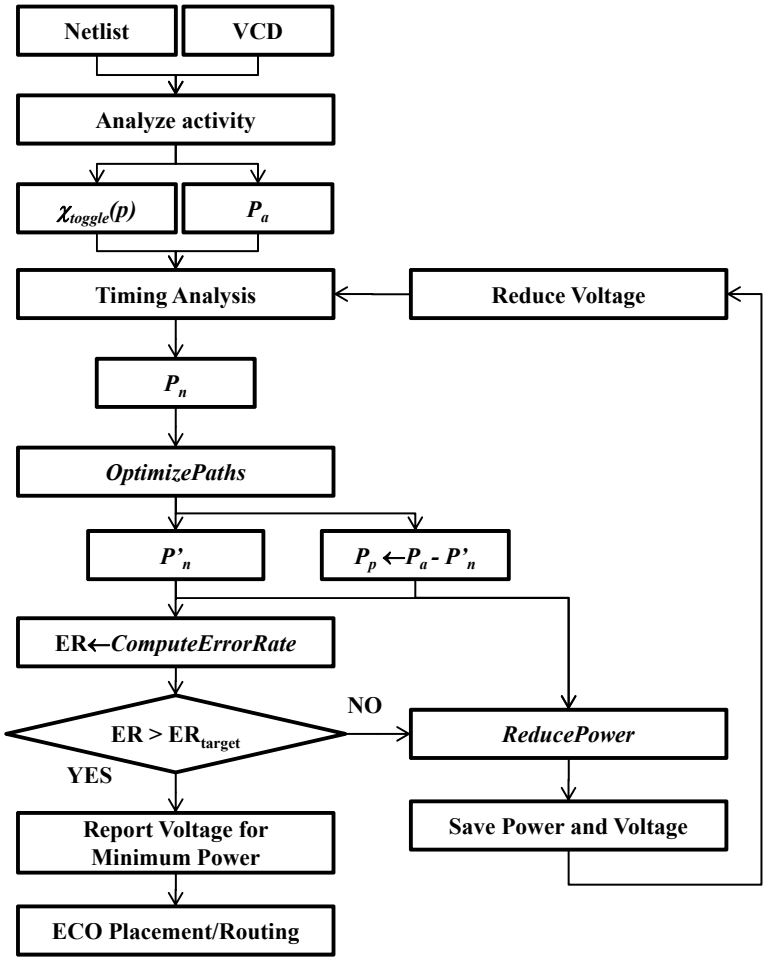

Figure 3.3: Algorithmic flow of a heuristic for minimizing power for a target error rate. P_a is the set of all paths toggled during simulation. P_p is the set of all non-negative-slack paths. P_n is the set of all negative-slack paths in P_a. $\chi_{toggle}(p)$ is the set of cycles in which path p is toggled.

Previously optimized (visited) fanin and fanout cells are protected from slack decrease because they belong to paths that have higher toggle rates, and thus, higher priority of optimization. If cell upsizing on a path fails to shift the path back into the set of positive slack paths, then the path is ignored during subsequent iterations of path optimization.

Any cell swap that increases the error rate (by causing a path to switch from the set of positive slack paths to the set of paths allowed to have negative slack) is rejected. Otherwise, we recompute the sensitivity of the swapped cell and all cells in its fanin / fanout network and select the next cell for downsizing.

Power Reduction. After path optimization, the error rate of the circuit is minimized at the present voltage. From this state, we proceed to minimize the power at the present voltage by utilizing the available error rate budget. Algorithm 1 (*ReducePower*) describes our power reduction procedure. The goal of the power reduction heuristic is to efficiently allocate the remaining error budget to infrequently-exercised paths in order to maximize power reduction achieved by cell downsizing. Typically, cells on P_{-} paths can exploit additional downsizing, because these paths are not bound by the normal timing constraint of the circuit.

The first step in power reduction is to choose additional paths to become negative-slack paths until the target error rate of the circuit is matched. Paths are selected in order to minimize the additional contribution to the error rate of the circuit. After defining the partition between negative and non-negative-slack paths, cell downsizing is performed for all cells in the circuit in order of minimum sensitivity. We define the sensitivity of a cell in Equation 3.1 as the change in cell slack (Δs_c) divided by the change in cell power (Δw_c) when the cell c is downsized by one size. The slack of cell c is defined as the minimum slack on any timing arc containing c. The power of cell c is the sum of static power ($w_{stat}(c)$) and dynamic power ($w_{dyn}(c)$) for the cell. This formulation of sensitivity is similar to those proposed by previous works targeting leakage power reduction [13, 14].

$$Sensitivity(c) = \frac{s_c - s_{c'}}{w_c - w_{c'}} \quad , where \;\; w_c = w_{stat}(c) + w_{dyn}(c) \tag{3.1}$$

Algorithm 1 Pseudocode (*OptimizePaths*, *ReducePower*).

Procedure $OptimizePaths(P, N_{V_i}, V_i)$

```
1. Clear 'visited' mark in all cells in the netlist N_{V_i};
2. while P ≠ ∅ do
3.    Select path p from P with maximum toggle rate;
4.    for each cell c in path p do
5.       if c.visited == true then continue;
6.       c.visited ← true;
7.       for each logically equivalent cell m for the cell instance c do
8.          Resize cell c with logically equivalent cell m;
9.          Q ← c ∪ visited fanin and fanout cells of c;
10.         for each path q in P that contains a cell in Q do
11.            if Δslack(q, c, m, V_i) < 0 then restore cell change;
12.         end for
13.      end for
14.   end for
15.   P ← P − p;
16. end while
```

Procedure $ReducePower(P_p, P_n, N_{V_i}, V_i, ER_{target})$

```
1. P_+ ← P_p and P_− ← P_n;
2. while P_+ ≠ ∅ do
3.    Select path p from P_+ with minimum ΔER(p);
4.    ER ← ComputeErrorRate(P_− + p);
5.    if ER ≤ ER_target then
6.       P_− ← P_− + p;   P_+ ← P_+ − p;
7.    else
8.       break;
9.    end if
10. end while
11. Insert all downsizable cells into set C;
12. ComputeSensitivity(C, N_{V_i}, V_i, −1);
13. while C ≠ ∅ do
14.    Downsize cell c from C with minimum Sensitivity(c);
15.    Q ← c ∪ fanin and fanout cells of c;
16.    for each path p in P_+ that contains a cell in Q do
17.       if slack(p, V_i) < 0 then
18.          Restore cell change;
19.          C ← C − c;
20.          continue while loop;
21.       end if
22.    end for
23.    ComputeSensitivity(Q, N_{V_i}, V_i, −1);
24.    if cell c is not downsizable then
25.       C ← C − c;
26.    end if
27. end while
```

3.4 Path Extraction and Error Rate Estimation

Path Extraction. Our heuristic has many path-based procedures – *OptimizePaths*, *ReducePower*, and *ComputeErrorRate* – and it is impossible to consider all of the topological paths in these procedures. Therefore, we reduce the number of paths that we consider by extracting only paths toggled during functional simulation. The value change dump (VCD) file can be used to extract toggled paths. To produce a VCD file, we perform gate-level simulation with *Cadence NC-Verilog v6.1* [4]. Figure 3.4 shows an example VCD file and the path extraction method. The VCD file contains a list of toggled nets in each cycle time, as well as their new values. We can use this information to extract truly toggled paths in each cycle. Changed nets in each cycle are marked, and these nets are traversed to find toggled paths. We detect a toggled path when toggled nets compose a connected path of toggled cells from a primary input or flip-flop input to a primary output or flip-flop output. In Figure 3.4, nets a, x, and y have toggled in the first and third cycles (#1, #3), and nets b and y have toggled in the second and fourth cycles (#2, #4). We extract two paths: $a - x - y$ and $b - y$.

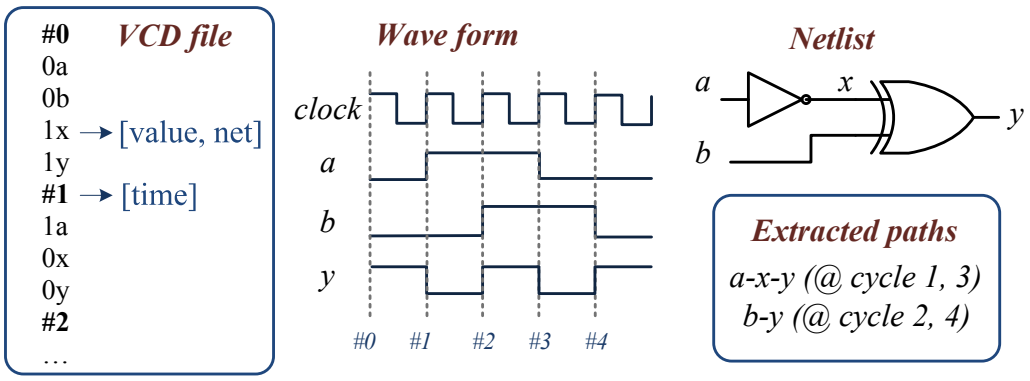

Figure 3.4: VCD file format and path extraction.

Toggle Rate and Error Rate Estimation. In order to accurately minimize power for a target error rate, we must be able to produce accurate estimates for error rate during our optimization flow. Thus, we propose a novel approach to error rate estimation that enables design for a target error rate.

We calculate the toggle rate of an extracted path using the number of cycles in which the path toggles. $\chi_{toggle}(p)$ represents the set of cycles in

which path p has toggled during the simulation. $TR(p)$ represents the toggle rate of path p and is defined as:

$$TR(p) = |\chi_{toggle}(p)|/X_{tot} \tag{3.2}$$

where $|\chi_{toggle}(p)|$ is the number of cycles in which path p has toggled, and X_{tot} is the total number of cycles in the simulation. Using the toggled cycle information of negative-slack paths, we can calculate the error rate precisely. The error rate (ER) of the design is calculated as:

$$ER = \frac{|\bigcup_{p \in P_n} \chi_{toggle}(p)|}{X_{tot}} \tag{3.3}$$

where P_n is the set of negative-slack paths in the set of all toggled paths. In Figure 3.4, if paths $a - x - y$ and $b - y$ both have a toggle rate of 0.4 (number of toggled cycles is 2 and number of total cycles is 5), and if path $a - x - y$ has negative slack, then timing errors will occur in cycles #1 and #3. Therefore, the error rate is 0.4 for this example.

3.5 Heuristic Design Choices

In this section, we discuss heuristic design choices.

Experiment 1: Path Ordering During Optimization. The order in which we select paths for optimization affects the optimization result, since we prevent cells from being visited multiple times during optimization. This order matters also because we protect previously optimized paths from slack degradation to other attempted upsizing moves, as previously optimized paths have a higher optimization priority. We evaluate two prioritization functions for path selection during optimization. The first ranks paths in order of decreasing toggle rate ($TR(p)$). Paths with the highest toggle rates have the greatest potential to decrease error rate when optimized. We compare against a function that ranks paths in order of decreasing $TR(p)/|slack(p)|$. In this alternative, we prefer paths with smaller negative slack, since these paths can be converted into non-negative-slack paths with

least upsizing.

Experiment 2: Optimization Radius. The goal of optimization is to maximize the slack of a targeted path through cell upsizing. We evaluate two alternatives for the radius of optimization. In one case, we only target cells on a given path for upsizing. In the second case, we target both the cells on the path as well as cells in their fanin / fanout networks, since swaps in the fanin / fanout network can also affect cell slack.

Experiment 3: Path Traversal During Optimization. When optimizing a path, the order in which cells are visited can have an effect on the optimization result, since cell swaps affect input slew and output load. We consider two options – traversal from front to back and from back to front. We iterate over the cells in a path and make swaps until there is no further increase in the path slack.

Experiment 4: Accuracy of Path Selection During Power Reduction. During power reduction, non-negative-slack paths are selected to be added to the set of paths allowed to have negative slack, thus utilizing the available error rate budget. Paths are prioritized in order of increasing incremental contribution to error rate, $\Delta ER(p)$. However, after moving a path from P_+ to P_-, $\Delta ER(p)$ can change for paths that shared error cycles with the moved path.

To obtain precise ordering in terms of error rate contribution, we can update $\Delta ER(p)$ after each path selection. However, this introduces a runtime overhead, since we must continuously update $\Delta ER(p)$ for all remaining P_+ paths. We compare such *precise prioritization* against the alternative case where $\Delta ER(p)$ is calculated only once for all P_+ paths before path partitioning.

Experiment 5: Error Rate Budget Utilization. During power reduction, the final error rate after cell downsizing could be less than the target error rate, ER_{target}, since some paths in P_- might still have non-negative slack, even after maximum downsizing on the path cells. In this case, we might continue to reduce the power of the design by selecting more paths to add to P_- and downsizing cells again. We evaluate two cases – one where a single pass is performed for path selection and cell downsizing, and one where the *ReducePower* procedure is repeated until there is no further reduction in

power (i.e., repeat *ReducePower* whenever some paths added to P_{-} still have non-negative slack after cell downsizing).

Experiment 6: Starting Netlist. Here, we evaluate heuristic performance for different starting netlists corresponding to loose (clock period increased by 10%) and tight (reduced by 40%) timing constraints. This can significantly affect the final voltage reached, the dependence on ECO, and the amount of power savings afforded by the power minimization algorithm.

Experiment 7: Voltage Step Size. In each iteration of the power minimization heuristic, we step down the voltage by a value V_{step} and run the *OptimizePaths* and *ReducePower* procedures to produce a netlist for the present level of voltage scaling. The size of V_{step} can influence the optimization result and runtime of the heuristic. Thus, we compare two values of V_{step} – 0.01 V and 0.05 V – and compare the characteristics of the final netlist as well as the heuristic runtime.

Experiment 8: Iterative Optimization. In each iteration of the heuristic, we perform optimization of negative-slack paths at that voltage level. During the next iteration, we have a choice between starting from the previously optimized netlist ($N_{V_{i-1}}$) or the original netlist (N_0). We compare the netlists produced in each case and see if they have similar power and runtime characteristics.

3.6 Gradual Slack Design

Our recovery-driven design methodology can be extended for *gradual slack design* [19], which re-shapes the slack distribution of a processor to create a gradual failure characteristic, rather than the typical critical wall. While error rate-optimized, recovery-driven designs achieve better energy efficiency at a single target error rate, gradual slack designs have the ability to trade reliability, throughput, or output quality for energy savings over a range of error rates. Figure 3.5 describes the optimization approach for gradual slack design.

To achieve a gradual slack distribution with our recovery-driven design flow, we do not optimize for a single target error rate by selecting P_{-} paths. Instead, we select the maximum target error rate corresponding to the desired

range of scalability, and optimize only the negative slack paths in the scaling range with the highest switching activity, in order to maximize the range of voltage scalability for target range of error rates. We downsize only cells that have negligible activity so that the slack distribution for the active paths and the error rate of the processor are not affected. In this way, we maintain the desired gradual sloping slack distribution rather than creating a critical wall distribution with a cluster of active paths in the permanent negative slack region.

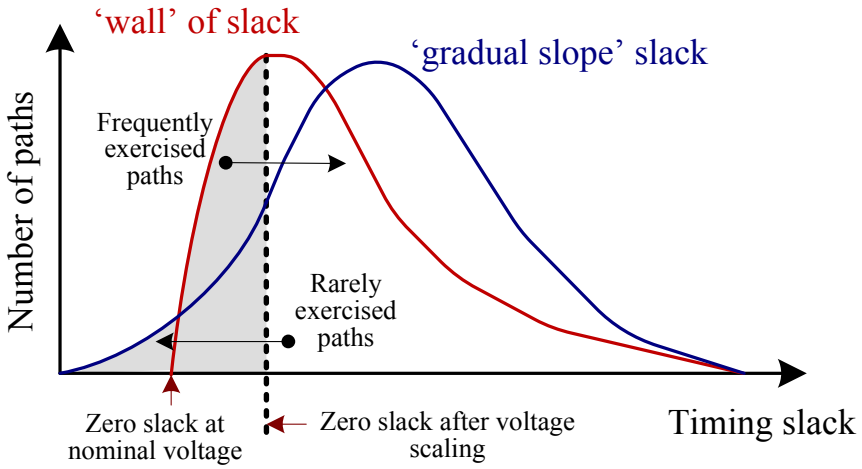

Figure 3.5: The goal of the 'gradual slope' slack optimization is to transform a slack distribution having a critical 'wall' into one with a more gradual failure characteristic. This allows performance / power tradeoffs over a range of error rates.

3.7 Processor Power Reduction

Algorithm 2 shows a heuristic for minimizing the power of a processor core for a target error rate. The first step of the above power-minimization heuristic involves characterizing the modules of the processor core in terms of their power consumption at different error rate and voltage targets. These data are provided by *PowerOptimizer* and are used to select the optimal operating voltage(s) for the processor core, as well as the error rate targets to assign to the processor modules.

The next step in the processor-level heuristic is to use the data from *PowerOptimizer* to solve an optimization problem. The optimization objec-

Algorithm 2 Processor-level Design Heuristic.

Procedure $OptimizeProcessor(ER_{target}, MODULES, DOMAINS)$

1. **for each** module m in the optimization list of MODULES **do**
2. **for each** error rate $ER < ER_{target}$ **do**
3. $PowerOptimizer(N(m), ER)$;
4. **end for**
5. Use the results from $PowerOptimizer$ to characterize $P_m(V, ER)$
6. **end for**
7. **for each** voltage $V \in V_{range}$ **do**
8. Minimize $P_{core}(V) = \Sigma(P_m(V, ER))$ s.t. $ER_{core}(ER_{module_1}, ..., ER_{module_M}) \leq ER_{target}$
9. Record minimum power $P_{core}^{min}(V)$ and module error rate assignment $S(V) = [ER_{module_1}, ..., ER_{module_M}]$
10. **end for**
11. Select the voltage V_{opt} at which power P_{core}^{min} is minimized
12. Let $V^*(S(V)[m])$ be the voltage that minimizes power for module m at $ER = S(V)[m]$
13. Locate the $DOMAINS$ neighbors $\{V_1, ..., V_{DOMAINS}\}$ nearest to the set of voltages $V^*(S(V_{opt}))$
14. Assign each module m to the voltage domain $V_D[m] \in \{V_1, ..., V_{DOMAINS}\}$ that minimizes power $P_m(V_D[m], S(V_{opt})[m])$
15. Layout the processor, selecting for each module $m \in$ MODULES the netlist $N(m, V_D[m], S(V_{opt})[m])$;

tive is to minimize the power of the processor core subject to the constraint that the processor error rate must be less than the chosen target rate. Using the data from *PowerOptimizer*, we can formulate expressions for the power and error rate of the processor core in terms of the module error rates and the operating voltage. Thus, the goal of the optimization problem for a particular voltage is to find the assignment of error rate targets to modules that satisfies the optimization objective. We use a disjunctively-constrained knapsack-based [32] approach to solve the optimization problem. The knapsack solver selects the voltage and error rate assignment for which the power of the processor core is minimized and uses the selected error rate-optimized netlist of each module to lay out the processor.

For multiple voltage domain designs ($DOMAINS > 1$), the heuristic selects the voltage level of each domain and the partitioning of modules to voltage domains to minimize core power. This involves first selecting the error rate targets for the modules based on a minimum-power global assignment, then selecting the levels for the voltage domains and module-to-level assignments such that the power of the modules is minimized. The latter step is performed using a nearest neighbor search to identify the neighbors nearest to the set of optimal module voltages corresponding to the module error rate assignments in the space of voltages.

CHAPTER 4

RECOVERY-DRIVEN PROCESSORS

The proposed design methodology enables *recovery-driven processors* – processors that are optimized to deliberately produce timing errors at a rate that can be gainfully tolerated by an error recovery mechanism. Below, we describe two recovery-driven processor designs – one targeting hardware-based error resilience and another targeting software-based error resilience.

4.1 Case Study: Circuit-Level Timing Speculation

One popular hardware-based scheme for error detection and correction is circuit-level timing speculation [8]. Circuit-level timing speculation-based techniques detect errors by sampling the same computation twice – once using the regular clock and again using a delayed clock. The two outputs are compared. When the outputs do not match, an error is signaled. Correction involves treating the delayed clock output as the correct output. Razor [8] is one good example of a circuit-level timing speculation-based scheme.

A recovery-driven processor design targeted for Razor takes into account the frequency of errors that can be gainfully tolerated by Razor (determined by the error recovery overhead) as well as the number of latches in which an error is allowed (which determines the cost of making the circuit robust to errors). For the design-level heuristic, this means that when we define the partition between faulty (P_-) paths and error-free (P_+) paths, we must consider the error rate contribution of a path, which adds to the recovery overhead of Razor, as well as the costs of using a Razor FF at the endpoint of a path and buffering any short paths terminating at that endpoint. If adding a path to the set of faulty paths means that we must replace a regular FF with a Razor FF, then we must ensure that the power benefit (in terms of power reduction for additional cell downsizing) outweighs the additional cost

of the Razor FF and short path buffering.

Figure 4.1 shows a methodology for tuning the target error rate for a recovery-driven processor that employs a hardware error resilience mechanism, such as Razor. First, an initial estimate of the optimal target error rate is made, based on characterization of the costs and benefits of error resilience with respect to error rate. This involves estimating the energy savings afforded by voltage scaling and the energy cost of error recovery and determining the voltage (and corresponding error rate) at which the cost and benefit equalize. Next, the processor is optimized for the selected target error rate using the *OptimizeProcessor* heuristic described in Section 3.7. Finally, we characterize the optimized design and check that the error rate for which energy is minimized matches the target error rate for which the design was optimized. We do this iteratively until a good matching is achieved and energy is minimized. Note that this methodology can be used for other timing speculation-based mechanisms such as Intel's EDS [29], as well as an instruction retry-based mechanism, with minor modifications.

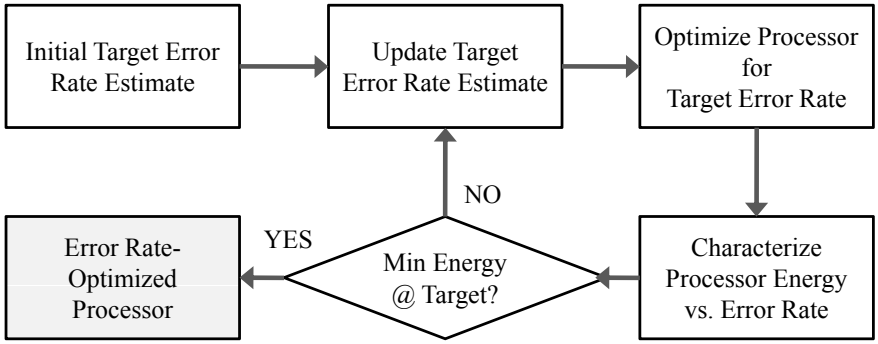

Figure 4.1: Optimizing a processor for hardware error resilience (like Razor or EDS) involves tuning the target error rate to locate the error rate at which energy is minimized.

4.2 Case Study: Application Noise Tolerance

Error-tolerant applications [31] represent an opportunity to save power and increase performance by allowing errors to propagate to the application level rather than expending power to detect and correct them at the hardware level. For several such applications, data errors simply result in reduced

output quality, instead of program failure.

Designing a recovery-driven processor for error tolerant applications requires several considerations. First, the set of processor modules is partitioned into two subsets – one containing modules that produce errors that the applications can tolerate and another containing modules that should not allow errors to propagate to the application level. For the class of error-tolerant applications that we consider in this paper, errors in the arithmetic units (i.e. ALU, FPU) can be tolerated. For this class of applications (which relies heavily on numerical computation), the arithmetic units account for approximately 35% of the dynamic power consumption of the processor.

In addition to the list of modules to optimize, the *OptimizeProcessor* procedure requires a target error rate. The error rate is chosen such that all applications in the class have acceptable quality for the target error rate.

For the modules that produce errors that the application cannot tolerate, one of two approaches can be followed. One option is to operate those modules on the same voltage rail as the modules in which faults are allowed (single rail design). In this case, we feed these modules to the optimization heuristic targeting some hardware recovery mechanism that guarantees correctness, such as Razor. The two groups must agree on a common voltage that minimizes power consumption for the entire processor, and the optimal voltage reported by the optimization heuristic can be used as a constraint for the second optimization. Alternatively, the two groups can operate in separate voltage domains (dual rail design), in which case each optimization can select a different optimal voltage.

Soft processor design can also be used to adapt the reliability of the processor for reliability-diverse workloads, with more power savings available as the error rate target decreases. To create a soft processor design, the gradual slack module-level heuristic is used, and the optimal voltage and error rate targets of the modules are chosen based on the range of error rate targets that the processor should support.

4.3 Heterogeneously-Reliable Multi-Core Processors

Different applications have different levels of intrinsic robustness. Some applications cannot tolerate errors, while others can seamlessly tolerate dat-

apath errors at the expense of performance or output quality [22]. Ideally, a processor core should be matched to the robustness of the application it is running.

Just as single-ISA heterogeneous multi-core processors [20] were proposed to efficiently meet the varying performance needs of different classes of applications, we propose *heterogeneously-reliable multi-core processors* (Figure 4.2) in which the cores on the processor are designed for different reliability targets using our processor-level design heuristic. For a reliability-diverse workload, a heterogeneously-reliable CMP can potentially achieve better energy efficiency than homogeneous CMPs by mapping an application with a specific reliability requirement to an appropriate core on the processor. A homogeneous CMP will waste power or performance by either over-provisioning for error correction when it is unnecessary or under-provisioning when protection is necessary and suffering a performance loss or power overhead to ensure reliability.

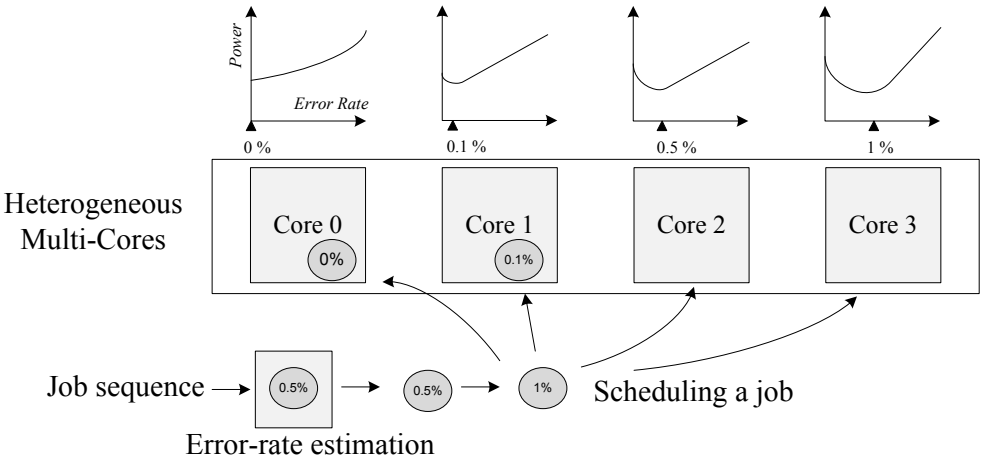

Figure 4.2: The heterogeneously-reliable CMP matches each application to the core that has minimum power for the application's reliability requirement.

An example heterogeneously-reliable dual-core CMP consists of one core designed for hardware error tolerance with Razor, and one core that exploits application-level error tolerance that is designed to allow errors in arithmetic units under nominal conditions. Error tolerant applications are mapped to the core designed to allow arithmetic errors, while applications that cannot tolerate errors are mapped to the core designed for Razor-based error tolerance. When the workload contains only applications that cannot tolerate

errors, the frequency of the correctness-relaxed core is reduced to prevent timing errors.

Another example heterogeneously-reliable multi-core processor may have different cores optimized for applications with different levels or types of software-based error resilience. For instance, one core may be optimized for an application that can tolerate floating point errors, while the other core is optimized for an application that can tolerate errors in the SAD (sum-of-absolute-difference) unit. Chapter 6 presents evaluations for an example heterogeneously-reliable multi-core processor.

The energy benefits of a heterogeneously-reliable multi-core processor will depend on the workload diversity as well as the efficacy of application-to-core mapping. A comprehensive exploration of mechanisms and policies for mapping applications to cores is a subject of future work.

CHAPTER 5

METHODOLOGY

Our methodology for demonstrating the benefits of recovery-driven design has two parts – a design-level methodology to characterize the power and reliability of circuit modules optimized for different voltage and error rate targets, and an architecture-level methodology to estimate processor power and performance when the proposed design-level techniques are applied at the processor-level.

5.1 Design-Level Methodology

We use the *OpenSPARC T1* processor [26] to test our optimization framework. Table 5.1 describes the selected modules and provides characterization in terms of cell count and area. Module designs are implemented in TSMC 65GP technology using a standard flow of synthesis with *Synopsys Design Compiler vY-2006.06-SP5* [27] and place-and-route with *Cadence SoC Encounter v8.1* [5]. Runtime is reduced by adopting a restricted library of 66 commonly-used cells[1] (62 combinational and 4 sequential). Conventionally constrained designs are synthesized for the target operating frequency (0.8 GHz), and tightly constrained designs are synthesized for a 40% smaller clock period to increase timing slack.

Figure 5.1 illustrates our recovery-driven design flow. We perform gate-level simulation to produce a VCD file[2] using *Cadence NC-Verilog v6.1* [4]. To find timing slack and power values at the specific voltages, we prepare *Synopsys Liberty* (.lib) files for each voltage value – from 1.00 V to 0.50 V in

[1]Heuristic efficiency depends on the number of available logically equivalent cells. Since we use all available cell sizes for different drive strengths, our heuristic will also be effective with a full set of library cells.

[2]Gate-level simulation is performed for one million cycles, and the size of the VCD file is about 500 MB for our test cases. To implement larger designs, a compressed VCD file could be used – e.g., *Synopsys VCD Plus* format.

Table 5.1: Target modules for experiments.

Module	Stage	Description	Cell #	Area (μm^2)
lsu_dctl	MEM	L1 Dcache Control	4537	13850
lsu_qctl1	MEM	LDST Queue Control	2485	7964
lsu_stb_ctl	MEM	ST Buffer Control	854	2453
sparc_exu_ecl	EX	Execution Unit Control	2302	7089
sparc_ifu_dec	FD	Instruction Decode	802	1737
sparc_ifu_errdp	FD	Error Datapath	4184	12972
sparc_ifu_fcl	FD	L1 Icache and PC Control	2431	6457
spu_ctl	SPU	Stream Processing Control	3341	9853
tlu_mmu_ctl	MEM	MMU Control	1701	5113

0.01 V increments – using *Cadence Library Characterizer v9.1* [3]. Complete characterization for 51 voltage points takes a couple of days, but this is a one-time cost.

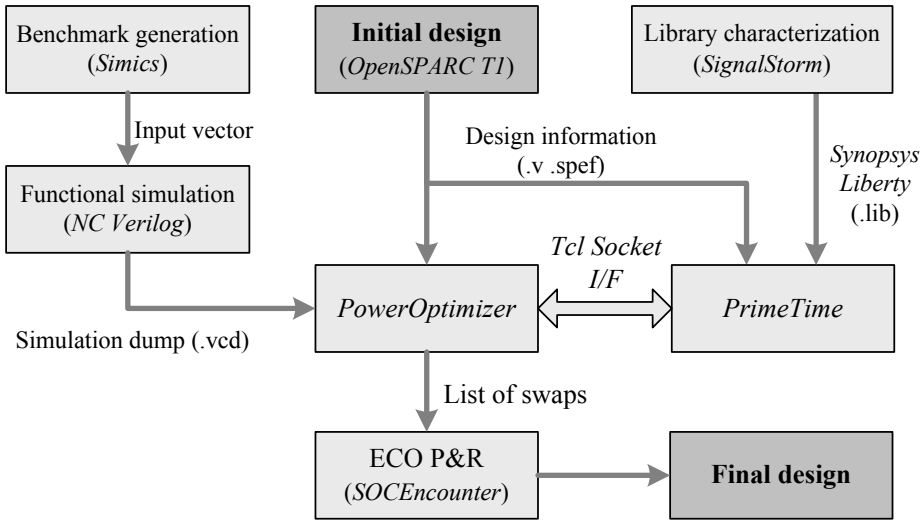

Figure 5.1: CAD flow incorporating the power optimization heuristic to minimize the power of a design for a given error tolerance technique.

Timing information is continually available from *Synopsys PrimeTime c2009.06* [28] static timing tool through the Tcl socket interface, during the optimization process. After our optimization, all netlist changes are realized using *Cadence SoC Encounter* in ECO (engineering change order) mode.

Gate-level simulation is performed using test vectors obtained from full-

system RTL simulation of a benchmark suite consisting of integer and floating point SPEC benchmarks. These benchmarks are each fast-forwarded to their early SimPoints using the OpenSPARC T1 system simulator, Simics [21] Niagara. After fast-forwarding in Simics, the architectural state is transferred to the *OpenSPARC* RTL using *CMU Transplant* [7].

Our recovery-driven design techniques optimize for average activity. To ensure that the activity profiles used during optimization (training) are representative and adequate, we use mutually exclusive training and test workloads. We optimize based on the average activity of half of our benchmarks and test using the other half. Training and test sets are chosen randomly and contain half integer and half floating point benchmarks. Table 5.2 shows the benchmarks in the training and test sets.

Table 5.2: Benchmarks.

Benchmarks for design optimization (training set)	
ART	Image Recognition / Neural Nets
BZIP2	Compression
MCF	Combinatorial Optimization
MESA	3D Graphics Library
Benchmarks for design evaluation (test set)	
EQUAKE	Seismic Wave Propagation
GZIP	Compression
TWOLF	Place and Route Simulator
SORT	Sorting
Additional benchmarks for processor-level evaluation	
AMMP, APPLU, MGRID, PARSER, SWIM, CRAFTY, EON, WUPWISE, VPR, VORTEX-2, FACEDETECT†, CG†, LSQ†, FIR† († *error-tolerant application*)	

5.2 Architecture-Level Methodology

We use SMTSIM [30] integrated with Wattch [2] to simulate processors whose single core parameters are in Table 5.3. The simulator reports performance and power numbers at different voltages. Our evaluations are done using benchmarks in Table 5.2. These benchmarks were chosen to maximize di-

versity in terms of performance and reliability requirements. We base our out-of-order processor microarchitecture model on the MIPS R10000 [33].

Table 5.3: Processor specifications.

Property	Value	Property	Value
L1 cache	16 kB, 4-way, 1 cyc	RegFile	72 (int), 72 (FP)
L2 cache	2 MB, 8 way, 8 cyc	Branch Predict	gshare (8k entries)
Execution	2-way OO	Mem Access	315 cyc

To get a processor-wide error rate at a given frequency and voltage, we first sum the error rates from all the sampled OpenSPARC modules and then scale up the sum based on area such that it includes all modules that we target for optimization. The error rate of a module that has not been characterized is assumed to be proportional to area. For hardware recovery-driven design, we target all logic modules except SRAM structures like register files and caches. Such modules already operate on a separate clock domain and are assumed to run, for a given voltage, at the highest frequency that produces no timing errors (0.8 GHz). For designs that rely on error-tolerant applications, we scale the error rates of each module group separately, according to an error rate characterization of sampled modules in the group. Once the processor core-wide error rate is calculated, we can use performance and power numbers reported by our simulators 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 application-level reliability-driven design, we scale the power of the module groups independently, as we did for error rate. 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 for Razor-based designs, we calculate the number of Razor flip-flops as the number of flip-flops that are endpoints to paths that can have negative slack at any operating voltage. For these flip-flops, we scale the power reported by *PrimeTime* to account for the increased power consumption of Razor flip-flops, which 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 [8], as well as the recovery time overhead of Razor to calculate the energy overhead of Razor error recovery. The overhead of Razor also includes buffering on short paths that have endpoints at the Razor flip-flops to meet the short path constraint [8] imposed by Razor.

Various design approaches also suffer throughput degradation under certain circumstances. For Razor, an error triggers a recovery period during which the processor recovers the pipeline to a correct state. During this time, we assume that no progress is made, but we do account for the power and time consumed during recovery when reporting processor throughput and energy numbers. When a processor designed for application-level reliability runs an application that requires correctness, we scale down the frequency of the processor so that no timing violations occur. The safe clock frequency of the design is determined by the worst case negative slack timing path in the processor plus a safety margin. All of our application simulations are executed for 1 billion cycles after fast-forwarding to the early SimPoints [23].

CHAPTER 6

RESULTS

6.1 Evaluation of Error Rate Estimation

Accurate error rate estimation is critical to power minimization when designing for a target error rate. Inaccurate estimation can lead to over- or under-optimization. Figure 6.1 compares the error rate estimation approach proposed in Section 3.4 of this work against the result computed during functional simulation, and an estimator used by the slack optimization heuristic in [19], [17].

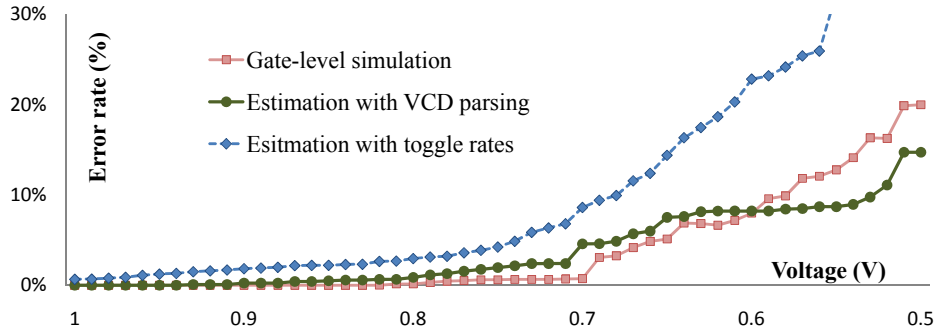

Figure 6.1: Actual error rate from functional simulation vs. estimated error rate for *lsu_stb_ctl*. Estimated error rate is also compared against an estimator used in [19] demonstrating much better correlation with actual error rate.

Our estimation technique compares favorably against the previously proposed estimator, and matches well with actual error rate. Root mean squared error for our technique is 0.1575 as opposed to 0.6002 for the technique used in the slack optimizer. Figure 6.2 compares the runtime of our estimation technique with that of actual simulation demonstrating over an order of magnitude decrease.

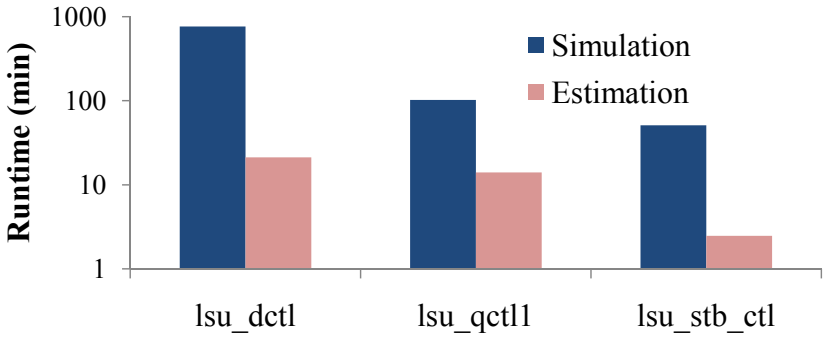

Figure 6.2: Runtime comparison of actual and estimated error rates.

6.2 Evaluation of Heuristic Design Choices

Figure 6.3 shows power and runtime of the various heuristic design alternatives that we evaluated, as described in Section 3.5. For **path ordering during optimization**, considering the slack in the prioritization function results in higher power than the case where only toggle rate is used. Runtime is somewhat smaller, but since our optimization iterates over a path multiple times until no slack increase is observed, both results perform similarly. For the same reason, **path traversal order** has little effect on the optimization result. We choose the toggle rate priority function for its simplicity and lower power.

The results for **optimization radius** show that swapping cells in the fanin/fanout network not only increases power at some error rates, but also greatly increases runtime due to the large amount of swaps that are performed. Thus, we choose to swap cells only on the optimized path. In the experiments on **accuracy of path selection** and **error rate budget utilization**, we observe no difference in power. Both updating the error rate contribution continuously during path selection and ensuring full utilization of the error rate budget increase runtime significantly without providing power benefits, and these techniques are not used in the final heuristic implementation.

The choices of **starting netlist** and **voltage step size** have a significant effect on power. Our recovery-driven design heuristic employs two main procedures – *OptimizePaths* (cell upsizing to reduce the error rate) and *ReducePower* (cell downsizing to reduce area and power). When starting the optimization flow from a loosely constrained design, path optimization pro-

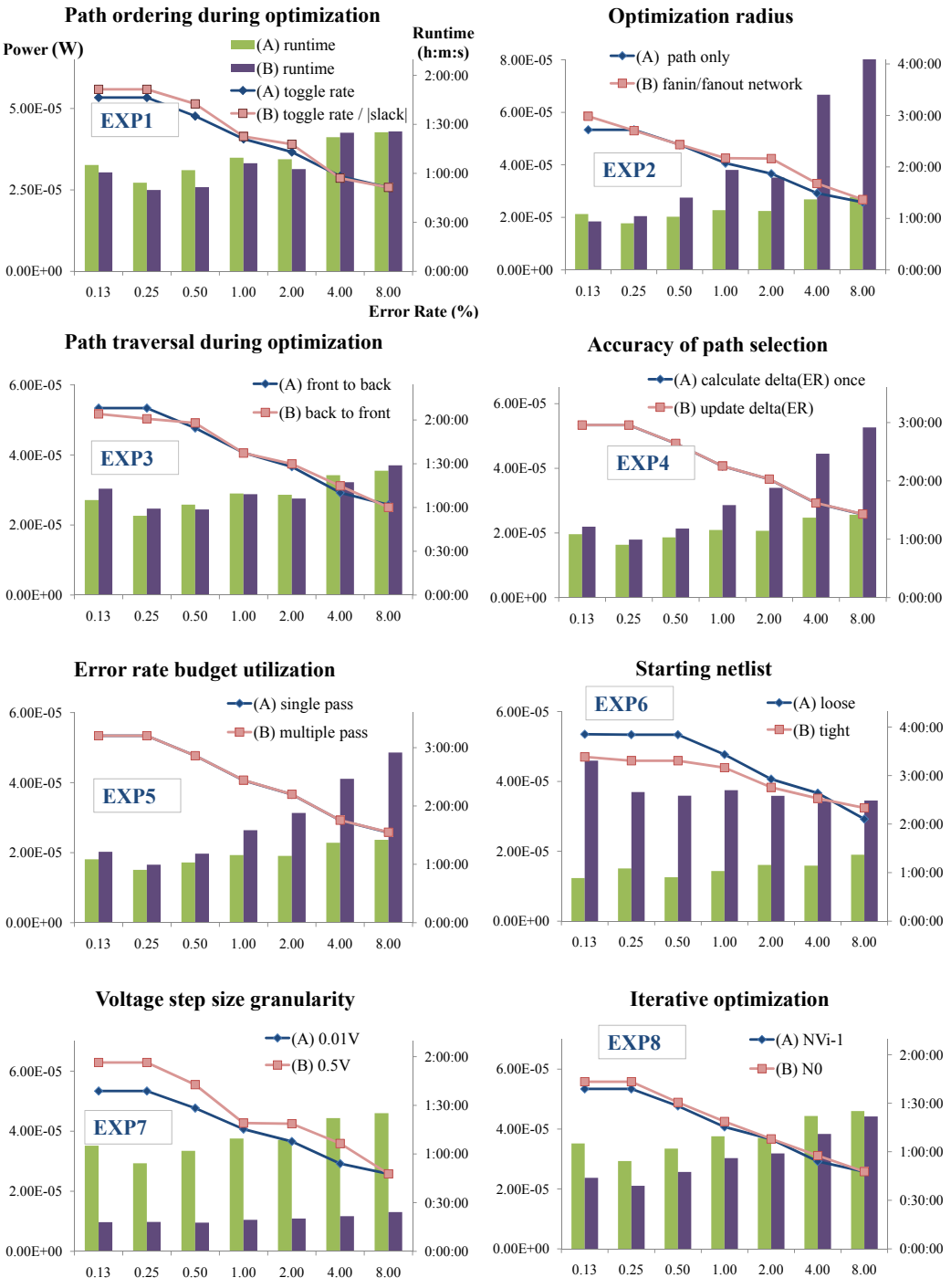

Figure 6.3: Evaluation of different heuristic design choices. The choices are evaluated in terms of power of the resulting design as well as runtime.

vides the most substantial contribution to power reduction by reducing the error rate and extending voltage scaling. However, when starting from a tightly constrained design, much optimization has already been performed, and the power reduction stage of our heuristic is essential for power minimization. Overall, a tightly constrained netlist provides a better starting point since it permits more voltage scaling, which has a stronger effect on power reduction and scales the power of all cells, while area reduction only affects the downsized cells. Also, starting from a tightly constrained design reduces the dependence on ECO, which improves the optimization efficiency. Using a coarser-granularity voltage step reduces runtime significantly, but comes at the cost of power, since the heuristic cannot hone in on the optimal voltage as easily. For higher error rates, a large step size can provide a near-optimal power result and a large reduction in runtime. Thus, error rate-aware adaptive step sizing can be beneficial.

In terms of **iterative optimization**, we observe that our heuristic is able to achieve the same result independent of the starting netlist. Thus, we choose the option that minimizes runtime.

6.3 Comparison Against Alternative Flows

To demonstrate the benefits of our recovery-driven design flow, we compare five alternative design flows – traditional P&R implementations with conventional and tight timing constraints, a *BlueShift*-like path constraint tuning (PCT) approach, gradual slack design [19], [17], and our heuristic for error rate-optimized recovery-driven design. Figure 6.4 compares the power consumptions of the various design techniques at several target error rates.

Recovery-driven designs reduce power by enabling extended voltage scaling and keeping area overhead low with respect to other optimization techniques. Compared to a conventionally optimized design, a recovery-driven design operates at a much lower voltage for a given target error rate, due to the functionally-aware optimization approach that optimizes the paths that cause the most errors. Compared against a highly-optimized design that uses tightly-constrained P&R, a recovery-driven design reduces power by minimizing the amount of area spent on path optimization. Traditional tightly-constrained designs are functionally agnostic and optimize all paths

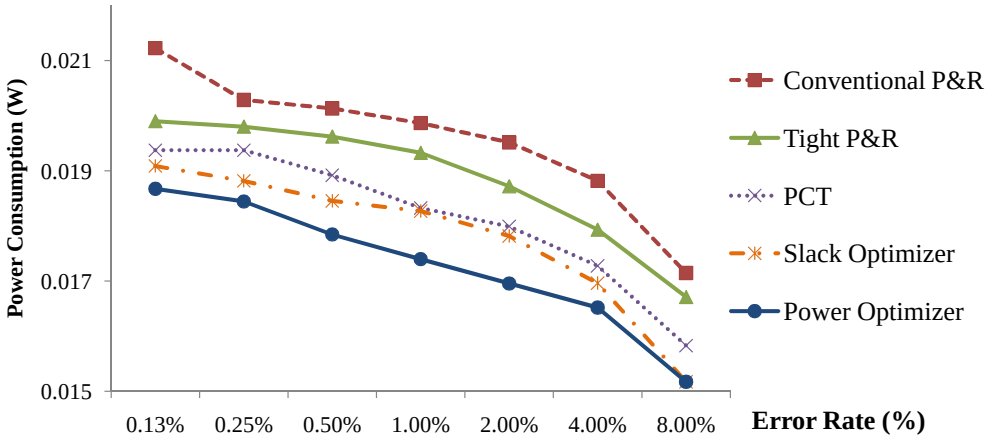

Figure 6.4: Power consumption of each design technique at various target error rates for target modules in Table 5.1. (Additional results are available in [9].)

heavily, incurring a large area overhead. Recovery-driven designs, on the other hand, use functional information to target only the paths that cause the most errors, thereby minimizing the area cost of additional voltage scaling. In scenarios where the cost of area is high, such as for technologies with higher leakage like those forecasted in future technology generations, the cost of functionally-agnostic optimizations will increase, and the benefits of recovery-driven design will increase. Table 6.1 shows power savings for recovery-driven design for each module with respect to each baseline design at different target error rates.

In our power minimization heuristic, after deciding how to allocate the error rate budget, the *ReducePower* stage performs aggressive cell downsizing to reduce circuit area and power. Table 6.2 compares recovery-driven design against other design flows in terms of area overhead with respect to the baseline design. Design for a target error rate has similar area overhead to PCT but still produces a design with lower power. The reason is that designing for a target error rate allows more aggressive voltage scaling before the target error rate is exceeded. At lower voltages, there are more negative slack paths to be optimized during *OptimizePaths*, which increases area overhead. However, aggressive downsizing keeps area overhead low, and since the paths targeted by *PowerOptimizer* are the paths that cause the most errors in the design, the area is well spent, and the additional voltage scaling contributes to a net benefit in terms of power savings. PCT, on the other hand, adds

Table 6.1: Power savings for error rate-optimized recovery-driven designs compared to traditional P&R.

MODULE	Target Error Rate (ER_{target})						
	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%	8.0%
Power savings w.r.t. conventional P&R							
lsu_dctl	29.1	16.8	16.8	16.8	16.8	16.8	21.6
lsu_qctl1	8.8	6.7	5.8	8.1	11.0	9.0	8.6
lsu_stb_ctl	17.9	17.9	18.1	15.4	9.6	19.2	2.9
sparc_exu_ecl	6.0	6.0	18.3	18.3	22.7	23.3	17.4
sparc_ifu_dec	13.7	10.1	8.6	14.3	15.9	18.5	15.1
sparc_ifu_errdp	2.2	2.8	5.7	5.7	5.7	9.3	9.3
sparc_ifu_fcl	14.5	15.4	16.5	19.2	19.2	19.2	19.2
spu_ctl	13.1	13.1	13.1	13.2	8.8	1.6	8.9
tlu_mmu_ctl	0.8	0.8	0.8	0.8	0.8	0.8	0.8
Power savings w.r.t. tight P&R							
lsu_dctl	17.0	17.0	17.0	17.0	17.0	17.0	23.6
lsu_qctl1	7.6	7.6	4.8	8.1	8.2	6.4	4.1
lsu_stb_ctl	9.3	7.4	11.2	10.0	3.2	-9.8	-2.1
sparc_exu_ecl	-16.6	-16.6	-1.3	-3.4	2.2	1.6	4.6
sparc_ifu_dec	-6.4	-5.3	-5.3	5.6	8.9	9.3	9.7
sparc_ifu_errdp	10.5	14.6	17.2	17.2	9.9	13.3	9.7
sparc_ifu_fcl	7.9	5.8	7.0	10.1	10.1	10.1	10.1
spu_ctl	15.1	15.1	15.1	12.8	10.9	-0.6	11.0
tlu_mmu_ctl	8.0	8.0	8.0	8.0	8.0	8.0	8.0

tighter timing constraints to the registers where the most errors are captured and optimizes all paths with endpoints at those registers. Since our heuristic targets paths individually, we can target the error-causing paths more efficiently, reduce overhead, and increase voltage scaling and power savings.

Compared to tightly constrained P&R and gradual slack design, design for a target error rate incurs significantly less area overhead and reduces power. On one hand, tightly constrained P&R is functionally agnostic and fails to identify the set of paths that maximizes voltage overscaling per unit area overhead. Gradual slack design, on the other hand, optimizes the design to make tradeoffs between power, throughput, and reliability over a *range* of error rates. Thus, a gradual slack design is over-optimized for any single target error rate.

Figure 6.5 compares recovery-driven design for a target error rate against

Table 6.2: Average area overhead with respect to the baseline.

Tight P&R	PCT	SlackOpt	PwrOpt 0.125%	PwrOpt 0.25%
19.1%	5.0%	11.9%	3.9%	4.3%
PwrOpt 0.5%	PwrOpt 1%	PwrOpt 2%	PwrOpt 4%	PwrOpt 8%
4.8%	5.4%	5.8%	6.0%	5.3%

gradual slack design. The results show that designing for a target error rate minimizes power at the target error rate. However, since a recovery-driven design can have a non-zero error rate even under nominal conditions, power efficiency at error rates lower than the target may drop off steeply. Likewise, since design for a target error rate creates a slack wall at the error-optimal voltage, additional benefits for error rates higher than the target are limited. A gradual slack design, on the other hand, is optimized for a *range* of error rates. Although this means that it is less efficient than an error rate-optimal design for any single error rate, it also means that performance or output quality can be efficiently traded for power savings over the entire range of error rates. Thus, whenever more errors can be tolerated, a gradual slack design can reduce power consumption. This may not be possible for an error rate-optimal design, since it forgoes scalability to achieve additional power savings at the target error rate.

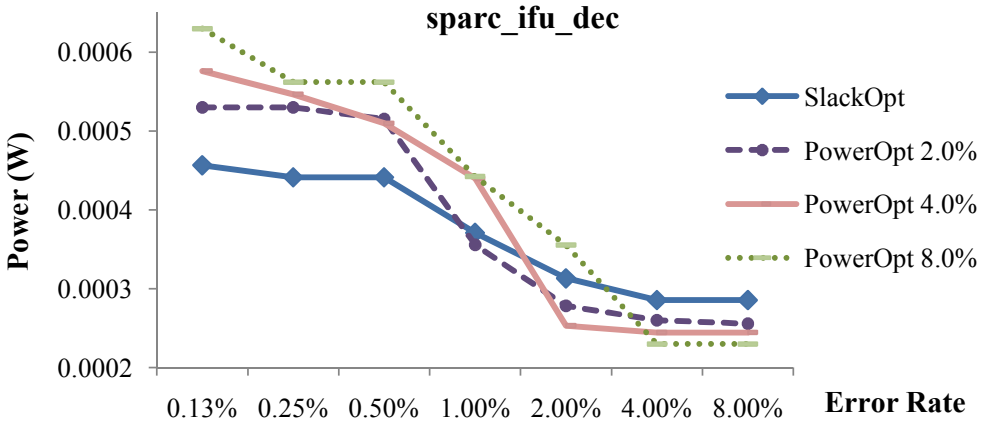

Figure 6.5: Recovery-driven design for a target error rate (*PowerOpt*) minimizes power at the target error rate. Gradual slack design (*SlackOpt*) optimizes a design for a range of error rates to provide adaptability and smooth performance / power tradeoffs.

Recovery-driven design optimizes for errors in the average operating behavior of a design. If the frequently exercised paths during operation are significantly different than those targeted during optimization, then too many errors may be produced, and voltage scaling may be limited for a target error rate. To evaluate the robustness of recovery-driven design when the workload changes, we compared the power reduction achieved when running the training (optimization) benchmarks against power reduction for the test benchmarks. Figure 6.6 shows that power reduction is slightly higher for the benchmark set that the processor was optimized for, but the difference is only about 1% on average.

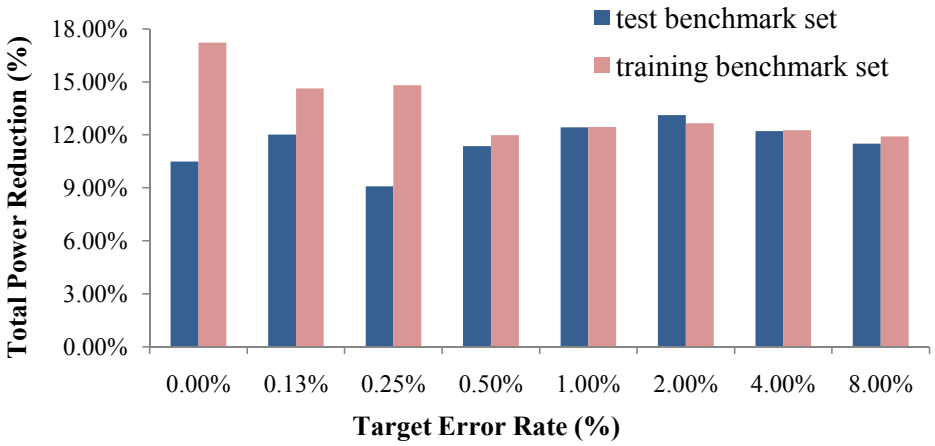

Figure 6.6: Total power reduction over tightly constrained design for the training (optimization) and test benchmark sets. Power reductions for the training set are slightly higher, since the design has been optimized specifically for the activity profile of this set.

6.4 Recovery-Driven Processors

In this section, we demonstrate the benefit of designing processors for specific hardware and software error resilience mechanisms, as described in Chapter 4.

6.4.1 Circuit-Level Timing Speculation

Figure 6.7 compares the energy consumption of a recovery-driven processor that has been designed and optimized for Razor against the power consump-

tion of processors designed for other objectives, such as gradual slack or PCT, and against processors that have been designed for correctness but use the traditional Razor methodology to save energy.

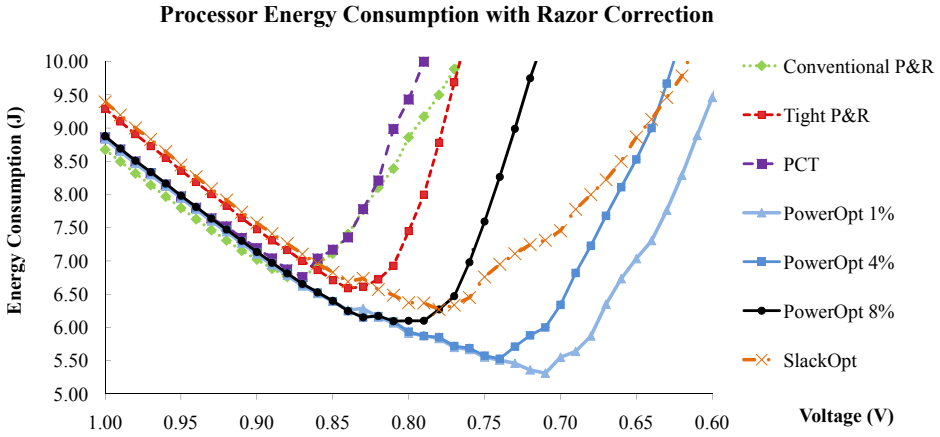

Figure 6.7: The benefit of designing a processor to produce errors and then correcting them with an error tolerance mechanism over designing for correctness and then relaxing the correctness guarantee can be significant. Results are shown for processors that employ Razor.

Figure 6.7 demonstrates that the minimum energy is indeed achieved by a processor that is designed to produce errors that can be gainfully tolerated by Razor. Designing the processor for the error rate target at which Razor operates most efficiently allowed us to extend the range of voltage scaling from 0.84 V for the best "designed for correct operation" processor to 0.71 V for the processor designed for an error rate of 1%, affording an additional 19% energy reduction.

Error recovery with a circuit-level approach like Razor imposes a throughput penalty, since error recovery requires feeding correct values back into the pipeline. Figures 6.8 and 6.9 show the throughput reduction caused by error recovery for correction overheads of 5 and 50 cycles, respectively. As can be seen, a recovery-driven processor even minimizes the recovery overhead for the target operating voltage.

6.4.2 Application Noise Tolerance

To demonstrate the benefits of recovery-driven design targeted at application-level noise tolerance, we use a face detection algorithm [31] as the example

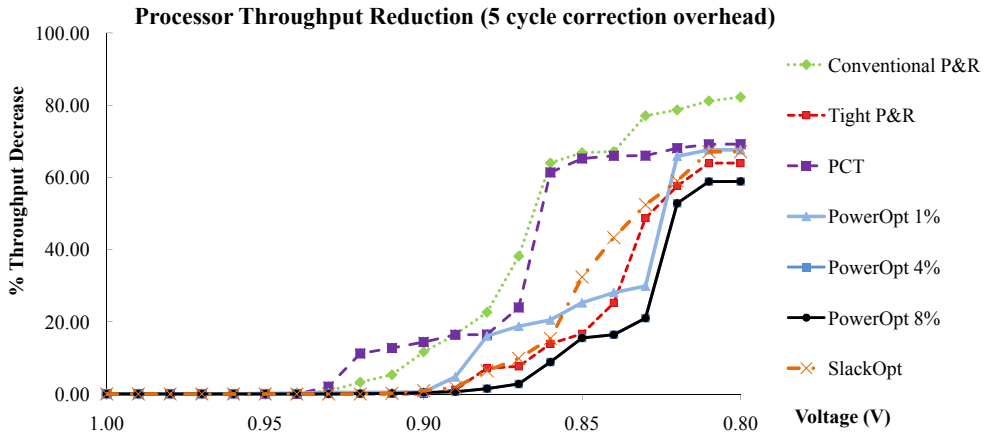

Figure 6.8: Throughput reduction at different voltages for an error recovery overhead of 5 cycles. This recovery overhead is appropriate for a simple pipeline or lightweight recovery technique.

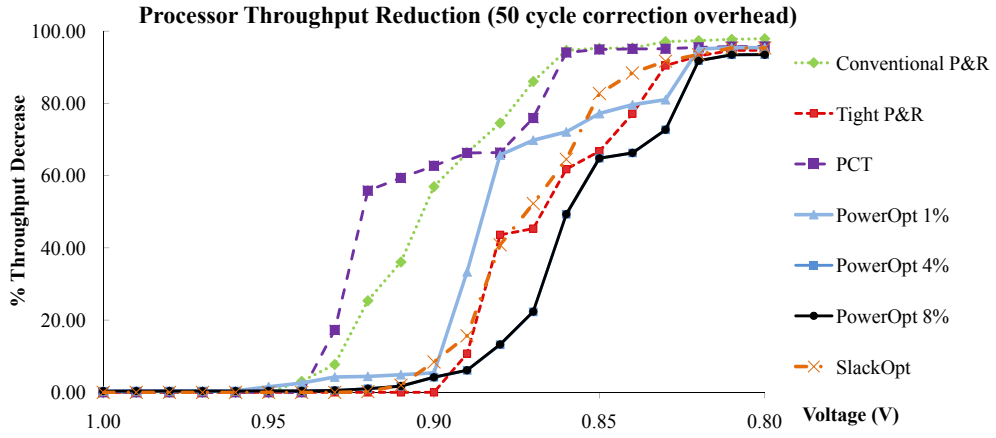

Figure 6.9: Throughput reduction at different voltages for an error recovery overhead of 50 cycles. This recovery overhead is appropriate for a more heavyweight recovery technique.

application. Face detection is naturally robust to errors in several processor modules and does not require strict computational correctness. Rather than causing program failure, errors may result in reduced output quality (false positive or negative detections) [22].

Face detection, as well as the other error-tolerant applications we consider, tolerates errors in the arithmetic units of the processor. For this class of applications (which relies heavily on numerical computation), the arithmetic units account for approximately 35% of the dynamic power consumption of the processor.

Figures 6.10 and 6.11 compare the power consumption of processors designed for application-level error tolerance of arithmetic errors using single and dual voltage rail designs, as described in Chapter 4. In these figures, all processors achieve the same output quality at a given error rate, but processors designed to allow errors consume less power, and power is minimized for these designs at their respective error rate targets. For example, at an error rate of 1%, where output quality is still maximized for the face detection application, the processor designed for an error rate target of 1% consumes 19% less power for dual-rail design and 15% less power for single-rail design than the baseline correctness-optimized processor. Benefits are even higher for larger error rates if some application output degradation is permissible.

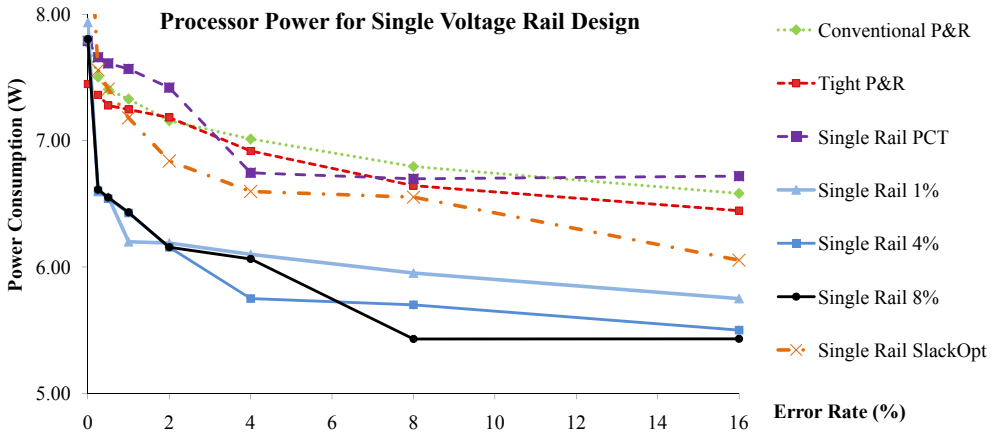

Figure 6.10: This figure demonstrates the power benefit of a processor that is designed to allow errors in the arithmetic units over a processor that is designed for correctness. All modules in the processor operate at the same voltage. Razor is used to correct errors in non-arithmetic units.

Note that we can always perform error-free computation on a core designed

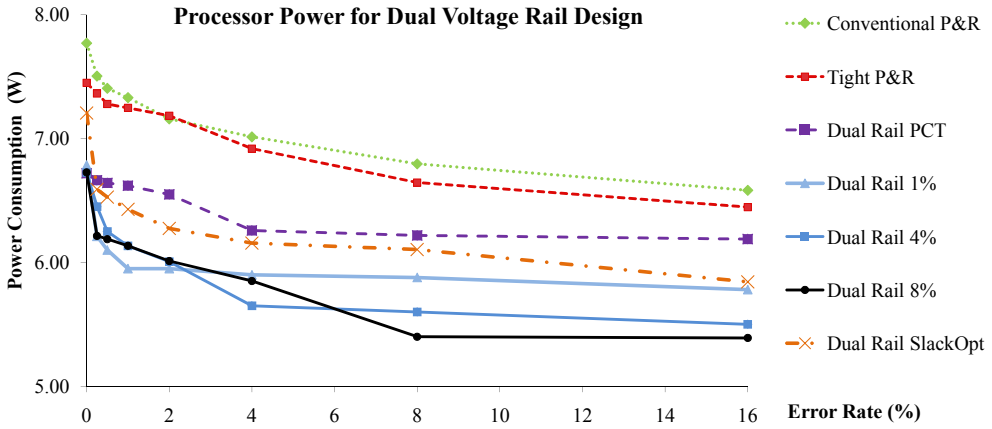

Figure 6.11: This figure demonstrates the power benefit of a processor that is designed to allow errors in the arithmetic units over a processor that is designed for correctness. The processor uses a dual voltage rail design with the arithmetic units on a separate rail.

for application-level noise tolerance by scaling down the frequency to the point where all paths have non-negative slack. However, this may represent a performance penalty when compared to relaxed-correctness operation.

Also note that trends in processor-level results may differ somewhat from trends in averaged module-level results. Whereas the power reduction of a recovery-driven design is limited by a module's critical paths, the power reduction of a recovery-driven processor is biased by the critical modules that begin causing errors first when voltage is scaled down. As we will show in the next section, results can be improved by utilizing multiple voltage domains.

6.5 Heterogeneously-Reliable Multi-Core Processors

To demonstrate the power benefits of heterogeneously-reliable multi-core processors over homogeneously-reliable multi-core processors, we evaluate a heterogeneously-reliable dual-core CMP where one core is designed for hardware-based error tolerance with Razor, and the second core relies on application-level error tolerance and is designed to allow errors in arithmetic units under nominal conditions. We consider three homogeneous configurations – one with baseline conventional cores designed for correctness, one with cores that are optimized for Razor-based correction, and one with cores that allow errors in arithmetic units and rely on software error tolerance.

We also consider three types of workloads – one includes only applications that do not tolerate errors (SPEC), one includes only applications that tolerate errors (face detection, conjugate gradient, FIR, least squares), and one includes a mixture of error-tolerant and error-intolerant applications. The Razor core and the application noise-tolerant core have been designed for an error rate of 1%.

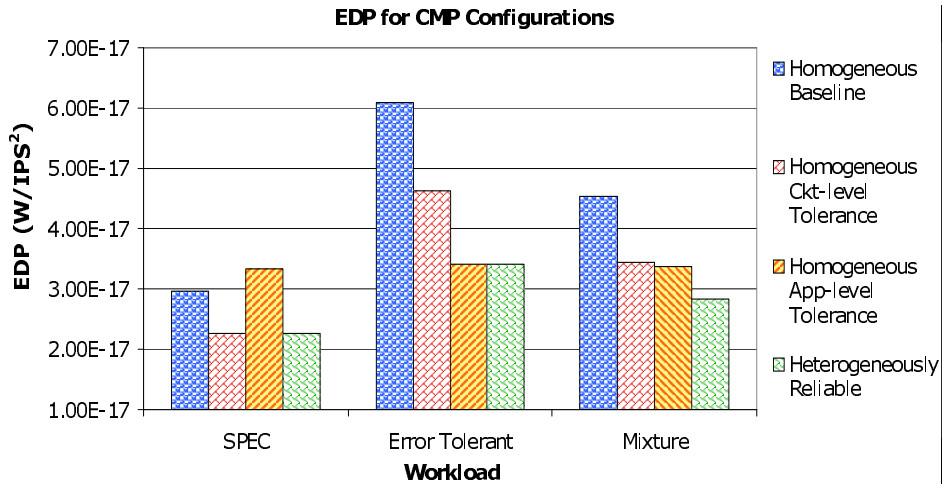

Figure 6.12: This figure compares a heterogeneously-reliable CMP against several homogeneous CMP configurations for a number of workload classes with different reliability requirements. The heterogeneously reliable CMP is able to adapt to the needs of the applications to provide power efficiency for a diverse set of applications.

Figure 6.12 compares the energy delay product (EDP) for various homogeneous CMP configurations against that of a heterogeneously-reliable CMP for different workload types. The results show that the homogeneous CMP that relies on applications to tolerate errors in the arithmetic units performs poorly for workloads with applications that have no error tolerance, since the frequency on the cores must be scaled down considerably to guarantee that no paths in the processor suffer from timing errors. In this case, the heterogeneously-reliable CMP has 32% lower EDP.

For workloads with exclusively error tolerant applications, the homogeneous CMPs that guarantee correctness (baseline and Razor-based error tolerance) suffer, since these configurations have over-designed arithmetic units. I.e., these configurations use additional power and area (due to guardbanding and Razor overhead, respectively) to ensure that no errors occur in these units, even though the error-tolerant applications can gainfully tolerate errors

in these units. These overheads result in 44% lower EDP for the heterogeneously reliable CMP with respect to the baseline CMP and 26% lower EDP with respect to the CMP with Razor-based error tolerance. For the workload with a mixture of applications with different reliability requirements, each homogeneous CMP suffers from the sub-optimality described above, and the heterogeneously-reliable CMP stands out uniquely as the most efficient design point, achieving EDP benefits of 37%, 17%, and 16% over the baseline CMP, and CMPs with Razor-based and software-based error tolerance, respectively.

The results above demonstrate how the heterogeneously-reliable CMP can adapt to different application requirements at a coarse granularity (error-tolerant or error-intolerant) to increase performance and reduce power. Note that energy benefits may be greater when performing task to core mapping at a finer granularity.

Another example heterogeneously-reliable multi-core processor that exploits diversity in the error resilience of applications consists of cores customized for different reliability targets. The applications are mapped to cores based on the amount of errors they can gainfully tolerate. Figures 6.10 and 6.11 demonstrate the power benefits of matching the reliability requirement of a task to the reliability design target of a core. Additional power savings become available as the tolerable error rate increases. Note that portions of a core may still be protected using a hardware-based error resilience mechanism.

6.6 Supporting Multiple Voltage Domains

Given a target error rate, the module-level power minimization heuristic in [18] selects an optimal operating voltage for a processor module. However, the proposed processor core-level methodology (Algorithm 1, $DOMAINS = 1$) selects a common voltage for all modules of a processor core. Table 6.3 shows that different modules vary (sometimes substantially) in their optimal voltage operating points due to a number of factors, including module area (number of paths/cells), slack distribution (fraction of paths that are critical), and activity factor (how often paths toggle). In addition, the table shows that the range of optimal module voltages increases when designing for a

non-zero error rate target.

Table 6.3: Optimal Module Voltages at Different Target Error Rates.

MODULE	Target Error Rate (ER_{target})						
	0.0%	0.125%	0.25%	0.5%	1.0%	2.0%	4.0%
lsu_dctl	0.75	0.72	0.71	0.75	0.74	0.73	0.72
lsu_qctl1	0.88	0.87	0.86	0.85	0.84	0.83	0.80
lsu_stb_ctl	0.77	0.76	0.75	0.75	0.70	0.68	0.66
sparc_exu_ecl	0.75	0.74	0.73	0.70	0.70	0.69	0.70
sparc_ifu_dec	0.68	0.67	0.66	0.63	0.70	0.58	0.57
sparc_ifu_errdp	0.77	0.58	0.57	0.56	0.55	0.54	0.53
sparc_ifu_fcl	0.79	0.77	0.76	0.75	0.74	0.73	0.72
spu_ctl	0.78	0.65	0.64	0.63	0.62	0.63	0.58
tlu_mmu_ctl	0.85	0.52	0.51	0.51	0.51	0.51	0.51
RANGE	0.20	0.35	0.35	0.34	0.33	0.32	0.29

Because of the above module-level variations, there can be a substantial difference in terms of power consumption between the locally and globally optimized module implementations. Figure 6.13 quantifies the difference between single and multiple voltage domain design for processor cores tolerating different error rates. We compare designs with different numbers of voltage domains, targeting different processor error rates in terms of their power consumption relative to a processor optimized for a common operating voltage. The results show that the power efficiency of recovery-driven processors will improve significantly with the number of voltage domains that are supported. In practice, the number of voltage domains should be chosen by carefully balancing the voltage overscaling benefits with the area and complexity overheads of supporting multiple power rails. The results of Figure 6.13 do not consider the overhead of level shifter circuitry.

6.7 Robustness to Application Diversity

Different workloads exercise the timing paths of a processor core differently. Thus, the sets of frequently-exercised and infrequently-exercised paths may change, depending on the workload. Since recovery-driven designs are optimized according to an average case activity profile, it is important to ensure

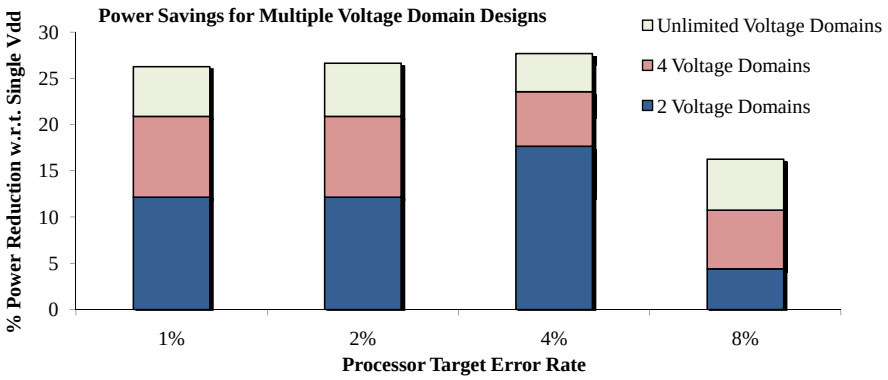

Figure 6.13: The benefit of a multiple voltage domain design over a single voltage domain design can be significant when designing for an error rate target. Substantial power savings can be achieved when each module is optimized for a locally optimal voltage rather than the globally optimal voltage of the module group. The stacked bars show the additional power savings afforded as the number of voltage domains increases.

that power efficiency is not degraded significantly when the activity profile of a workload is not the same as the activity profile for which the processor was optimized.

To gauge the robustness of recovery-driven design to workload diversity, we create several recovery-driven designs, optimized for the activity profiles of each benchmark in the test set – *equake*, *gzip*, *sort*, and *twolf*. Then, we compare the power consumption of each benchmark in the test set, running on the design that was optimized for the average case, against the design that was optimized specifically for that benchmark. Figure 6.14 compares the power consumption of average case design against workload-specific designs for different target error rates.

On average, the difference is small – only 1.5% difference in power at an error rate of 0.125% and 0.9% difference at 0.25% – demonstrating the robustness of recovery-driven design to application diversity. The difference will decrease as the target error rate increases. The reason for this robustness is that since some paths are allowed to cause errors, there is some "forgiveness" when the priority of path optimization deviates somewhat from the optimal. Our recovery-driven design heuristic bins paths into P_- paths that are allowed to cause errors and P_+ paths that should remain error free. As long as the difference in activity for a path is not so much as to make the path switch bins, the path dichotomy is preserved and power efficiency is not degraded.

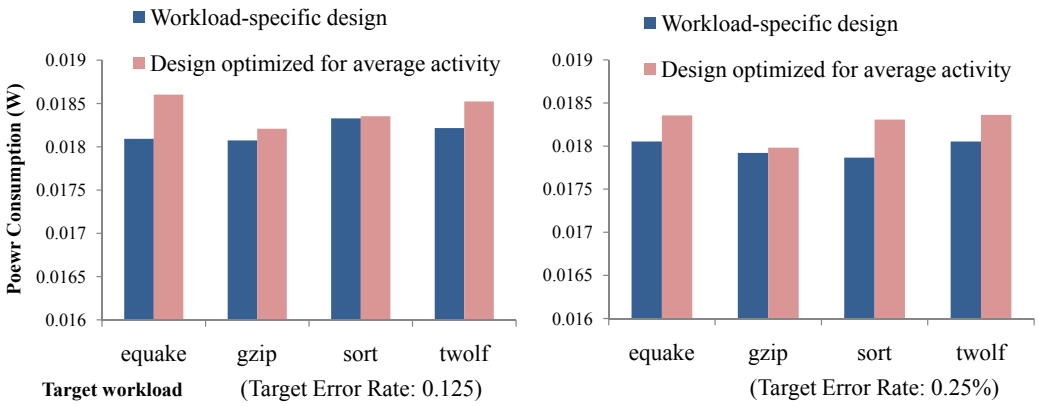

Figure 6.14: Recovery-driven design is robust to application diversity. On average, processor modules that have been optimized for the average case only consume 1% more power than modules that have been customized specifically for the activity profile of the test workload.

In the worst case, we only observe 3% degradation in power efficiency.

CHAPTER 7

CONCLUSION

In this thesis, we propose *recovery-driven design*, a design-level approach that optimizes a processor module for a target timing error rate instead of correct operation. We present a detailed evaluation and analysis of a recovery-driven design methodology to minimize processor power for a target error rate. We demonstrate that such a design flow can result in power savings – 6.1%, 6.8%, 9.1%, 10.0%, 9.4%, 7.9%, and 9.2%, on average, with a 15% reduction in area compared to tightly constrained traditional P&R at error rates of 0.125%, 0.25%, 0.5%, 1%, 2%, 4%, and 8%, respectively. Average power savings are 12.0%, 9.1%, 11.4%, 12.4%, 13.1%, 12.2%, and 11.5% with a 4.8% area overhead compared to conventional P&R at the same error rates. We have observed benefits of up to 29.1% for individual modules.

We extend our recovery-driven design flow to design recovery-driven processors – processors that are designed and optimized for a target error rate. We also present an extension of our recovery-driven design flow that creates a gradual slack design that is optimized for a range of error rates rather than a single target. The gradual slack technique is used to design soft processors that can trade throughput or output quality for energy savings over a range of reliability targets. Additional power savings are possible for recovery-driven designs that employ error resilience. We demonstrate up to 19% additional energy savings for a recovery-driven design that uses Razor to correct timing violations and up to 20% additional energy savings (with no loss in output quality) for a recovery-driven design that employs application-level error tolerance. We also show that *heterogeneously-reliable multi-core processors* – chip multiprocessors in which different cores are power-optimized for different reliability targets – have substantial power and EDP benefits over their conventional counterparts. Energy benefits are up to 29% and increase with the added flexibility of multiple voltage domains.

As the need for energy-efficient processing increases and as applications

exhibit increasingly diverse forms of error tolerance, the benefits of the proposed design philosophy will continue to increase.

REFERENCES

[1] T. Austin, V. Bertacco, D. Blaauw and T. Mudge, "Oppotunities and challenges for better than worst-case design," in *Proc. Asia and South Pacific Design Automation Conference*, 2005, pp. 2–7.

[2] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *International Symposium on Computer Architecture*, 2000, pp. 83–94.

[3] *Cadence LC User's Manual.* http://www.cadence.com/, 2010.

[4] *Cadence NC-Verilog User's Manual.* http://www.cadence.com/, 2010.

[5] *Cadence SOCEncounter User's Manual.* http://www.cadence.com/, 2010.

[6] 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," in *Proc. Design, Automation and Testing in Europe*, 2006, pp. 1110-1115.

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

[8] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 7–18.

[9] *Experimental Results for All Testcases (full version).* http://vlsicad.ucsd.edu/RecoveryDriven/DATA.pdf, 2010.

[10] J. P. Fishburn and A. E. Dunlop, "Tilos: A polynomial programming approach to transistor sizing," in *Proc. ACM/IEEE International Conference on Computer-Aided Design*, 1985, pp. 326–328.

[11] S. Ghosh and K. Roy, "CRISTA: A new paradigm for low-power and robust circuit synthesis under parameter variations using critical path isolation," in *IEEE Trans. on Computer-Aided Design*, vol. 26, no. 11, pp. 1947–1956, 2007.

[12] 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," in *Proc. International Symposium on High-Performance Computer Architecture*, 2009, pp. 213–224.

[13] P. Gupta, A. B. Kahng and P. Sharma, "A practical transistor-level dual threshold voltage assignment methodology," in *Proc. International Symposium on Quality Electronic Design*, 2005, pp. 421–426.

[14] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Gate-length biasing for runtime-leakage control," in *IEEE Trans. on Computer-Aided Design*, vol. 25, no. 8, pp. 1475–1485, 2006.

[15] P. Gupta, A. B. Kahng, P. Sharma and D. Sylvester, "Selective gate-length biasing for cost-effective runtime leakage control," in *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 327–330.

[16] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proc. International Symposium on Low Power Electronics and Design*, 1999, pp. 30–35.

[17] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Designing a processor from the ground up to allow voltage/reliability tradeoffs," in *Proc. International Symposium on High-Performance Computer Architecture*, 2010, pp 119–129.

[18] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Recovery-driven design: A methodology for power minimization for error tolerant processor modules," in *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 825–830.

[19] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *Proc. Asia and South Pacific Design Automation Conference*, 2010, pp. 825–831.

[20] R. Kumar, K.I. Farkas, N.P. Jouppi, P. Ranganathan and D.M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2003, p. 81.

[21] 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," in *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[22] J. Sartori, J. Sloan and R. Kumar, "Fluid NMR - Performing power/reliability tradeoffs for applications with error tolerance," in *Workshop on Power Aware Computing and Systems*, 2009.

[23] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, "Automatically characterizing large scale program behavior," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002, pp. 45–57.

[24] S. Sirichotiyakul, T. Edwards, C. Oh, R. Panda, and D. Blaauw, "Duet: An accurate leakage estimation and optimization tool for dual-Vt circuits," in *IEEE Trans. on VLSI Systems*, vol. 10, no. 2, pp. 79–90, 2002.

[25] A. Sultania, D. Sylvester, and S. S. Sapatnekar, "Tradeoffs between gate oxide leakage and delay for dual T_{ox} circuits," in *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 761–766.

[26] *Sun OpenSPARC Project.* http://www.opensparc.net/, 2010.

[27] *Synopsys Design Compiler User's Manual.* http://www.synopsys.com/, 2010.

[28] *Synopsys PrimeTime User's Manual.* http://www.synopsys.com/, 2010.

[29] J. W. Tschanz, K. Bowman, S-L. Lu, P. Aseron, M. Khellah, A. Raychowdhury, B. Geuskens, C. Tokunaga, C. Wilkerson and T. Karnik, V. De, "A 45nm resilient and adaptive microprocessor core for dynamic variation tolerance," in *International Solid-State Circuits Conference*, 2010, pp. 282–283.

[30] D.M. Tullsen, "Simulation and modeling of a simultaneous multithreading processor," in *Annual Computer Measurement Group Conference*, 2006, pp. 819–828.

[31] P. Viola and M. J. Jones, "Robust real-time face detection," in *International Journal of Computer Vision*, vol. 52, no. 2, pp. 137–154, 2004.

[32] T. Yamada, S. Kataoka and K. Watanabe, "Heuristic and exact algorithms for the disjunctively constrained knapsack problem," in *Information Processing Society of Japan Journal*, vol. 43, no. 9, pp. 2864–2870, 2002.

[33] K. Yeager, "The MIPS R10000 superscalar microprocessor," in *Proc. IEEE/ACM International Symposium on Microarchitecture*, 1996, pp. 28–40.