Exploiting Timing Error Resilience in Processor Architecture

JOHN SARTORI and RAKESH KUMAR, University of Illinois at Urbana-Champaign

Escalating variations in modern CMOS designs have become a threat to Moore's law. In light of the increasing costs of standard worst-case design practices, timing speculation has become a popular approach for dealing with static and dynamic non-determinism and increasing yield. Timing speculative architectures allow conservative guardbands to be relaxed, increasing efficiency at the expense of occasional errors, which are corrected or tolerated by an error resilience mechanism. Previous work has proposed circuit or design-level optimizations that manipulate the error rate behavior of a design to increase the efficiency of timing speculation. In this paper, we investigate whether architectural optimizations can also manipulate error rate behavior to significantly increase the effectiveness of timing speculation. To this end, we demonstrate how error rate behavior indeed depends on processor architecture, and that architectural optimizations can be used to manipulate the error rate behavior of a processor. Using timing speculation-aware architectural optimizations, we demonstrate enhanced overscaling and up to 29% additional energy savings for processors that employ Razor-based timing speculation.

Categories and Subject Descriptors: C.1.m [**Computer Architectures**]: Miscellaneous

General Terms: Design, Performance, Reliability

Additional Key Words and Phrases: error resilience, computer architecture, energy efficiency, timing speculation

ACM Reference Format:
Sartori, J., and Kumar, R. 2011. Exploiting Timing Error Resilience in Processor Architecture. Submitted to PEC Special Issue, ACM Trans. Embedd. Comput. Syst. 9, 4, Article 39 (March 2011), 26 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Traditionally, processors have been architected to operate correctly under worst case operating conditions. Ensuring timing correctness under all possible circumstances requires that conservative guardbands be imposed on operating frequency and voltage, limiting the performance and energy efficiency of modern processor designs, especially as device feature sizes continue to shrink and the impact of process and dynamic variations escalates. The growing costs of providing the illusion of perfect hardware on top of increasingly stochastic and unreliable devices have become prohibitive. To counter the rising costs of variability more efficiently, several timing speculative error resilient design techniques have been proposed [Ernst et al. 2003; Bowman et al. 2009; Greskamp and Torrellas 2007; Kehl 1993; Dhar et al. 2002]. These techniques relax correctness guards to gain efficiency in the average case at the expense of some errors. Errors are corrected or tolerated by hardware or software error resilience mechanisms to maintain the level of output quality expected by the user.

Author's addresses: J. Sartori and R. Kumar, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

The magnitude of efficiency benefits available from timing speculation depends on two factors – where and how often the processor produces errors when operating at an overscaled voltage or frequency. If the frequency of errors can be reduced for a timing speculative design, the range of overscaling can be extended, affording additional energy or performance gains. Previous works have demonstrated the potential to increase the energy efficiency [Kahng et al. 2010a; 2010c; 2010b] or performance [Greskamp et al. 2009; Sarangi et al. 2008] benefits of timing speculation by modifying the error distribution of a timing error resilient design. However, these work focused only on circuit-level techniques. It remains to be shown whether architecture-level optimizations can similarly affect the error distribution of a timing speculative design to generate energy or performance gains.

In this work, we demonstrate that the error distribution indeed depends on architecture. We show that the error distribution of a design that has been architected for error free operation may limit scalability and energy efficiency for better-than-worst-case operation. Thus, optimizing architecture for correctness can result in significant inefficiency when the actual intent is to perform timing speculation. In other words, one would make different, sometimes counterintuitive, architectural design choices to optimize the error distribution of a processor to exploit timing error resilience. Thus, we make a case for timing error resilience-aware architectures and propose architectural optimizations that improve the efficiency of timing speculation.

This work on timing error resilience-aware architecture makes the following contributions.

— We show that the error distribution of a timing speculative processor strongly depends on its architecture. As such, we demonstrate that architectural optimizations can be used to significantly improve the efficiency of timing speculation.
— We confirm, with experimental results for different implementations of a 4-tap FIR filter and Alpha, MIPS [Bertacco et al.], FabScalar [Choudhary et al. 2011], and OpenSPARC [Sun] processor cores, that timing error resilience-aware architectural design decisions can indeed significantly increase the efficiency of a timing speculative architecture.

Note that we have used voltage overscaling as the proxy for all variation-induced errors in this paper. Our analysis and conclusions should apply for other sources of timing variation as well.

The rest of the paper is organized as follows. Section 2 describes our fault model and explains how the slack and activity distributions of a processor determine the error rate, and consequently, the energy efficiency of a timing speculative architecture. Section 3 describes the architectures that we evaluate and provides examples of how architectural decisions can influence the slack and activity distributions of a design. Section 4 describes our experimental methodology. Section 5 presents results and analysis showing that optimizing an architecture for timing speculation can significantly improve energy efficiency. Section 6 discusses related work. Section 7 concludes the paper.

2. BACKGROUND

Before exploring if and how architectural optimizations affect the efficiency of timing speculation, we first provide details about our fault model and how slack and activity determine the error rate. The extent of energy benefits gained from exploiting timing error resilience depends on the error rate of a processor. In the context of voltage overscaling, for example, benefits depend on how the error rate changes as voltage decreases. If the error rate increases steeply, only meager benefits are possible [Kahng et al. 2010a]. If the error rate increases gradually, greater benefits are possible.

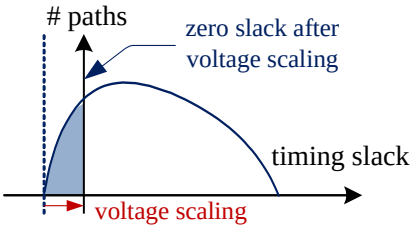
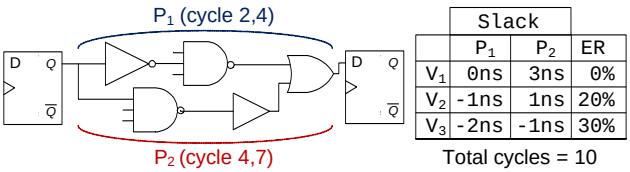


Fig. 1. Voltage scaling shifts the point of critical slack. Paths in the shaded region have negative slack and cause errors when toggled.



	Slack		
	P_1	P_2	ER
V_1	0ns	3ns	0%
V_2	-1ns	1ns	20%
V_3	-2ns	-1ns	30%

Total cycles = 10

Fig. 2. Slack and activity distributions determine the error rate.

The timing error rate of a processor in the context of voltage overscaling depends on the timing slack and activity of the paths of the processor. Figure 1 shows an example slack distribution. The slack distribution of a circuit shows the number of paths in a design at each value of timing slack. As voltage scales down, path delay increases, and path slack decreases. The slack distribution shows how many paths can potentially cause errors because they have negative slack (shaded region). Negative slack means that path delay is longer than the clock period.

From the slack distribution, it is clear which paths cause errors at a given voltage. In order to determine the error rate of a processor, the activity of the negative slack paths must be known. A negative slack path causes a timing error when it toggles. Therefore, knowing the cycles in which any negative slack path toggles reveals the number of cycles in which a timing error occurs.

For example, consider the circuit in Figure 2 consisting of two timing paths. P_1 toggles in cycles 2 and 4, and P_2 toggles in cycles 4 and 7. At voltage V_1, P_1 is at critical slack, and P_2 has 3ns of timing slack. Scaling down the voltage to V_2 causes P_1 to have negative slack. Since P_1 toggles in 2 of 10 cycles, the error rate of the circuit is 20%. At V_3, the negative slack paths (now P_1 and P_2) toggle in 3 of 10 cycles, and the error rate is 30%.

3. UNDERSTANDING AND MANIPULATING THE ERROR DISTRIBUTION OF TIMING SPECULATIVE ARCHITECTURES

In this section, we argue that both the slack and activity distributions of processors are strongly dependent on processor architecture. This implies that architectural features can be chosen to optimize the slack and activity distributions, and by extension, the error distribution and energy efficiency of a timing speculative processor. First, we demonstrate how slack and activity distributions depend on processor architecture. Then, we show how architectural optimizations can change the slack and activity distributions.

3.1. Architectural Dependence of Slack and Activity Distributions

In this section, we show that slack and activity distributions indeed depend on architecture. First, we present four functionally equivalent architectural variants of a 4-tap FIR filter We describe how the architectural characteristics of each filter determine the properties of its slack and activity distributions.

The baseline FIR filter, shown in Figure 3(a), is the simplest and most well-known arrangement of the FIR filter architecture, containing four MAC units. A pipelined version of the filter (Figure 3(b)) was created by creating a cutset across the outputs of the multipliers and adding a latch to each arc. We also created a folded version of

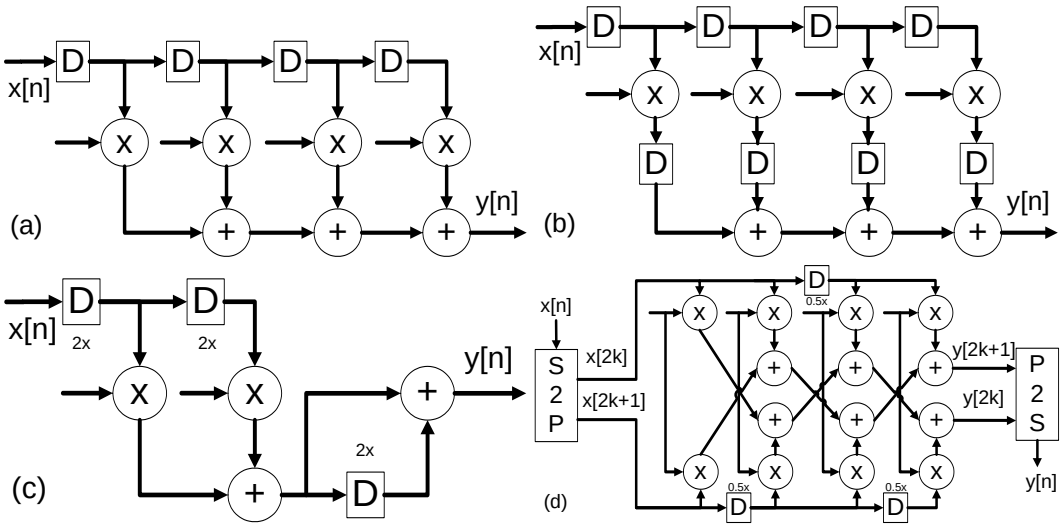


Fig. 3. 4-tap FIR filter designs: (a) Baseline, (b) Pipelined, (c) Folded, (d) Blocked.

the filter (Figure 3(c)), in which multiple operations are mapped to a single hardware unit. Folding by a factor of two multiplexes the filter hardware so that half of the filter coefficients are active in even cycles, the other half are active in odd cycles, and an output sample is computed every two cycles. The blocked architecture of Figure 3(d) was created by replicating the internal filter structure to compute two samples in parallel.

Figure 4 compares the path slack distributions of the different filter implementations, confirming our intuition that the slack distributions of the filter designs depend strongly on the architecture. Table I presents more detailed information on how slack and activity change for different architectures. The mean and standard deviation of the slack distribution (μ_{slack} and σ_{slack}, respectively) tell how much initial slack exists, on average, and how regular the slack distribution is, i.e., how spread out the values of path delay are. Designs with more regular (less spread) slack distributions allow less overscaling past the critical point because a large number of paths fail at the same time, potentially causing a steep increase in error rate. The average path activity (α_{path}) shows how frequently paths toggle. Higher path activity can mean that error rate increases more steeply, since negative slack paths generate more errors when they toggle more frequently.

Table I and Figure 4 reveal that pipelined and folded architectures have more regular slack distributions. These architectures have shorter paths that have less capacitance, less delay sensitivity to voltage scaling, and less variation in absolute path delay. This creates extra slack compared to other architectures, but limits scaling past the critical point. The folded architecture has high path activity, since the internal filter elements must operate at twice the frequency of the baseline design to achieve the same sample rate. Likewise, the blocked architecture has reduced path activity, since the same sample rate can be achieved at half the operating frequency. Although it has reduced activity, the blocked architecture has increased complexity and longer paths than the baseline. This results in more spread in the slack distribution, allowing more overscaling when errors can be tolerated, although errors may start at a higher voltage. Figure 5 shows how the power and error rate of each filter architecture vary with voltage, confirming the expected effects of the slack and activity distributions on the error rate of each architecture.

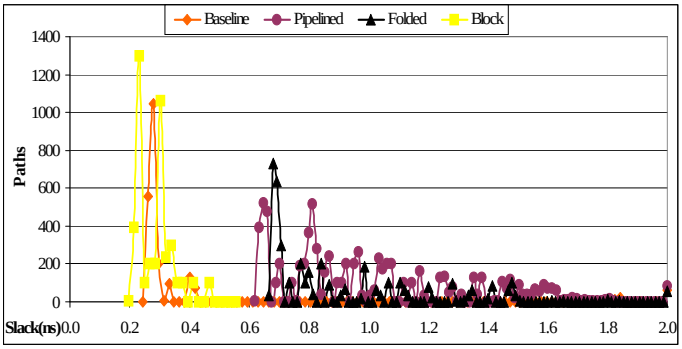


Fig. 4. Slack distributions for the FIR filter architectures.

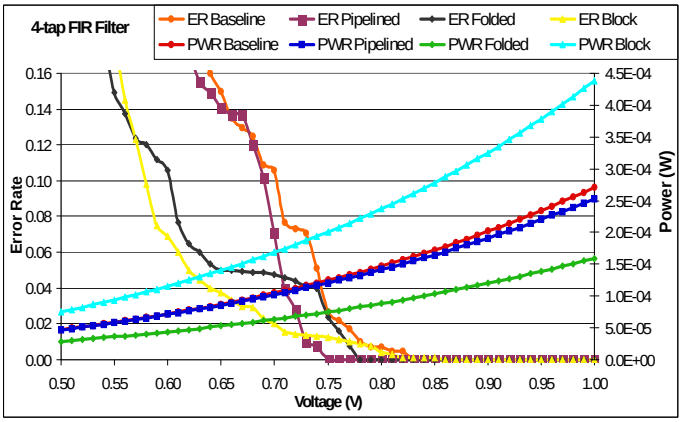


Fig. 5. Power and error rate vs. voltage for the FIR filter architectures.

Table I. Mean and standard deviation of path slack, relative to the sampling period, and average path activity normalized against the baseline.

	Baseline	Pipelined	Folded	Blocked
μ_{slack}	0.183	0.496	0.449	0.154
σ_{slack}	0.185	0.159	0.145	0.124
$Avg(\alpha_{path})$	1.0	1.9	3.4	0.5

Our simple DSP filter examples show that the architecture of a design shapes the properties of its slack and activity distributions. We now show that the same is true for general purpose processors. As demonstrated in Section 2, error rate is a function of the slack and activity distributions, and our primary goal is to use architectural optimizations to manipulate the error rate behavior of a design. Thus, we use error rate as a proxy for slack and activity. We begin by synthesizing four variants of the FabScalar [Choudhary et al. 2011] processor with different microarchitectural characteristics.

Figure 6 shows that the four different FabScalar microarchitectures have significantly different error rate behavior, demonstrating that slack, activity, and error rate indeed depend on microarchitecture. Differences in the error rate behavior of different cores are due to several factors. First, changing the sizes of microarchitectural units like queues and register files changes logic depth and delay regularity, which in turn

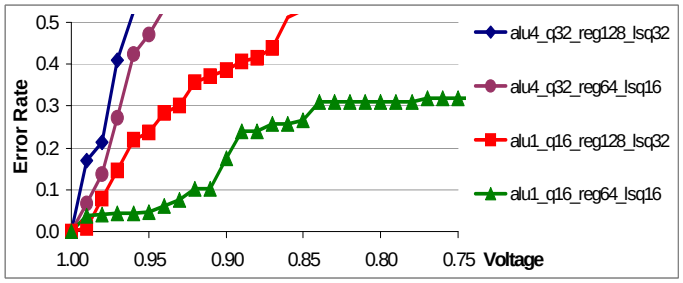


Fig. 6. Different microarchitectures exhibit different error rate behaviors, demonstrating the potential to enhance the energy efficiency of a timing speculative architecture through microarchitectural techniques. (Notations described in Section 5.3.)

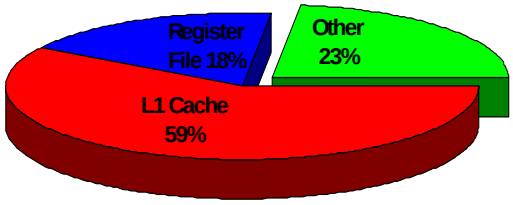


Fig. 7. Typically, slack distributions of processors are dominated by regular structures. Caches and register files account for a large fraction of the critical paths of a processor [Pan et al. 2009].

effects the slack of many timing paths. Secondly, varying some architectural parameters like superscalar width has a significant effect on complexity [Palacharla et al. 1997]. Changing complexity, fanout, and capacitance change path delay sensitivity to voltage scaling and cause the shape of the slack distribution to change. Finally, changing the architecture alters the activity distribution of the processor, since some units are stressed more heavily, depending on how the pipeline is balanced. High activity in units with many critical paths can cause error rate to increase more steeply. Likewise, an activity pattern that frequently exercises longer paths in the architecture limits overscaling. E.g., long dependence chains lengthen the dynamically exercised critical path of structures such as the issue queue and load store queues that perform dependence checking. As these queues become full, they begin to generate errors at higher voltages.

3.2. Architectural Optimizations that Manipulate Slack and Activity Distributions

Now that we understand the relationships between slack, activity, error rate, and architecture, we consider what must be done to optimize processor architecture for improved timing speculation efficiency. In this section, we propose specific architectural optimizations for general purpose processors that manipulate their slack and activity distributions. In Section 5, we show how these changes to the slack and activity distributions translate into significant energy savings for timing speculative architectures.

Regular Structures Typical energy-efficient processors devote a large fraction of die area to structures with very regular slack distributions, such as caches and register files. These structures typically have high returns in terms of energy efficiency (performance/watt) during correct operation. For example, 75-80% of the critical paths in the Alpha EV7 reside in the L1 caches and register files (Figure 7) [Pan et al. 2009].

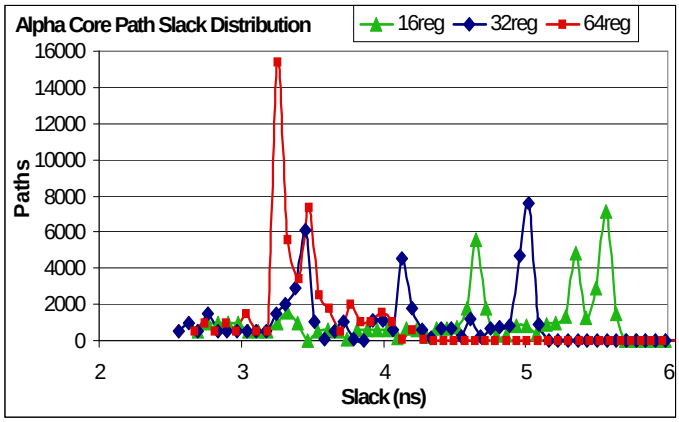


Fig. 8. Reducing the size of the register file (a regular structure) increases the spread of the slack distribution, resulting in fewer paths bunched around the point of critical slack.

While regular structures are architecturally attractive in terms of processor efficiency for correct operation, such structures have slack distributions that allow little room for overscaling. This is because all paths in a regular structure are similar in length, and when one path has negative slack, many other paths also have negative slack. For example, consider a cache. Any cache access includes the delay of accessing a cache line, all of which have nearly the same delay. So, no matter which cache line is accessed, the delay of the access path will be nearly the same. Compare this to performing an ALU operation, where the delay can depend on several factors including the input operands and the operation being performed. When exploiting timing speculation-based error resilience for energy reduction, the energy-optimal error rate is found by balancing the marginal benefit of reducing the voltage with the marginal cost of recovering from errors [Ernst et al. 2003]. When many paths fail together, error rate and recovery overhead increase steeply upon overscaling, limiting the benefits of timing speculation. Reducing the number or delay of paths in a regular structure can reshape the slack distribution, enabling more overscaling and better timing speculation efficiency.

For the Alpha core, the register file is the most regular critical structure. Figure 8 shows slack distributions for the Alpha core with different register file sizes. As the size of the register file increases, the regularity of the slack distribution also increases, as does the average path delay. Figure 8 confirms that the spread of the slack distribution decreases with a larger register file. Additionally, path slack values shift toward zero (critical) slack due to the many critical paths in the register file. Table II shows standard deviation and mean values for the slack distributions of the processors with different register file sizes. The table confirms that regularity (represented by the standard deviation of slack) increases, and average slack decreases with the size of the register file. (Note that smaller σ_{slack} means a more regular slack distribution.) We confirmed similar behavior when the cache size was changed. For example, σ_{slack} reduced by 25% for the Alpha core and 23% for the MIPS core when the cache size was increased from 2KB to 4KB.

Architectural design decisions that reshape the slack distribution by devoting less area to regular structures or moving regular structures off the critical path can enable more overscaling and increase energy efficiency for timing speculative processors. In other words, additional power scaling enabled by architectures with smaller regular structures can outweigh the energy benefits of regularity when designing a timing

Table II. Mean and standard deviation of path slack, relative to the clock period, for the Alpha processor with different register file sizes.

	16reg	32reg	64reg
μ_{slack}	46%	41%	34%
σ_{slack}	10%	9%	6%

speculative architecture. Since regularity-based decisions may also impact power density, yield, and performance, the final architectural decision should consider these constraints in addition to the optimization metric. Section 5 presents examples showing that reducing the regularity of the slack distribution can provide significant energy benefits when employing Razor-based timing speculation.

Note that [Liang and Brooks 2006] also advocates several choices that may affect the delay regularity of an architecture. However, unlike [Liang and Brooks 2006], our goal is not necessarily to increase slack but rather to reshape the slack and activity distributions of a processor. Decisions advocated in [Liang and Brooks 2006] increase slack but also make the slack distribution more regular. For example, when choosing the architecture for an arithmetic unit, we might advocate selection of a ripple-carry adder for its irregular slack distribution and lower average case delay [Sartori and Kumar 2010], despite its higher critical path delay. [Liang and Brooks 2006], on the other hand, would choose a Kogge-Stone adder to decrease critical path delay, also making the slack distribution more regular.

Logic Complexity Typically, processors are architected for energy efficiency during error free operation at a single power/performance point and are not expected to scale to other points. However, timing speculative architectures achieve benefits by scaling beyond the typical operating point to eliminate conservative design margins. The change in the shape of the slack distribution as voltage changes depends on the delay scalability of the paths. Therefore, unlike conventional architectures, architectures optimized for timing speculation should consider the delay scalability of different microarchitectural structures.

There are several architectural characteristics that affect delay scalability that conventional processors ignore to varying degrees. One factor that affects delay sensitivity to voltage scaling is logic complexity. In a conventional processor, microarchitectural components are optimized largely oblivious to complexity, as long as the optimization improves processor efficiency at the nominal design point. However, more complex structures with more internal connections, higher fanouts, deeper logic depth, and larger capacitance are more sensitive to voltage scaling, potentially limiting overscaling for a timing speculative processor.

Figure 9 demonstrates how the critical path delay of the ALU of the OpenSPARC T1 [Sun] processor changes with voltage scaling. Path P1 is the critical path at nominal voltage. However, the delay of P2 is more sensitive to voltage scaling due to increased fanout. The slack distribution of a processor with many complex logic structures becomes more critical more quickly as voltage is scaled, limiting overscaling.

To maximize the energy efficiency benefits of timing speculation, architectural decisions should be scalability-aware. For example, complex architectural structures with high degree of fanout should be optimized to reduce complexity, if possible. Similarly, less complex implementations of architectural units can be chosen when performance is not significantly impacted. Example optimizations include changing superscalar width and queue sizes – factors that strongly influence logic complexity. The capacitance of a logic structure also influences the rate at which delay increases with voltage reduction. If the impact on processor efficiency is acceptable, less area should be

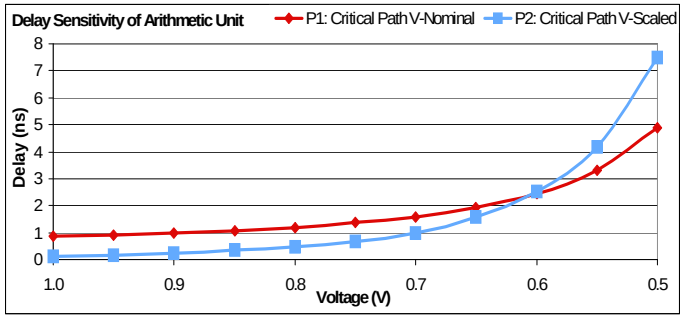


Fig. 9. Some paths are more sensitive to voltage scaling than others. Complex logic with many high fanout paths like P2 can limit overscaling in a timing speculative architecture.

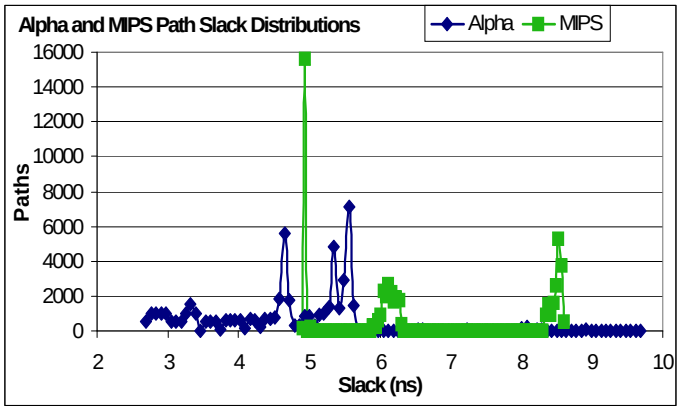


Fig. 10. The reduced regularity and complexity of the MIPS architecture, compared to the Alpha architecture, results in a slack distribution with greater average slack and reduced regularity.

devoted to complex and centralized structures with high internal capacitance (e.g., rename logic, wakeup/select logic, bypass logic, etc.).

Comparing the Alpha and MIPS architectures reveals again how architectural changes affect the slack distribution. Figure 10 compares the slack distributions of the MIPS and Alpha processors. The MIPS slack distribution has both higher mean and standard deviation than the distribution for the Alpha processor, indicating reduced regularity and complexity. These factors can be attributed to reduced word length, simpler ALU design, smaller area devoted to the register file, and a simpler, smaller instruction set, which results in less complex control logic throughout the processor.

Utilization Modern processors consistently employ architectural techniques such as pipelining, superscalar processing, caching, etc. to improve utilization by reducing the number of control and data hazards and mitigating long latency memory delays. In general, when designing for correctness, architectural design choices that increase utilization are desirable, as higher utilization of a processor core often leads to better performance. However, architectures with highly utilized critical paths are susceptible to high error rates, since increased activity on negative slack paths means more frequent errors. Architectural optimizations that reduce the activity of critical paths have the potential to reduce the error rate when timing speculation is performed.

The filter architectures described in Section 3.1 demonstrate how changes to the architecture affect the activity distribution. Table I shows that average path activity can be varied over a range of 6.8x by changing the amount of parallelism used in the filter architecture.

Superscalar Width As noted above, superscalar width has a stong impact on processor complexity [Palacharla et al. 1997]. In addition, changing the superscalar width can significantly impact the activity distribution of a processor. We evaluate the effect of changing the superscalar width of the MIPS architecture. We observed that average activity increases by up to 25% for the superscalar version of the processor, compared to the scalar version. Section 5 provides results that show how architectural changes that affect the activity distribution alter the energy efficiency of Razor-based timing speculation.

Note that activity reduction has associated costs in terms of performance during correct operation. We do not advocate reducing activity at all costs, but rather balancing the error rate reduction and energy efficiency benefits of activity reduction with the throughput benefits of high utilization. Note also that a work such as [Liang and Brooks 2006] is unconcerned with the activity distribution of a processor, since the goal is to *prevent* errors, not to reshape the error distribution.

Pipeline Depth The relationship between pipeline depth and energy efficiency is well understood in the context of error free architectural design [Hartstein and Puzak 2003]. The energy-optimal pipeline depth of an architecture is reached when the marginal benefit of adding a pipeline stage equals the marginal cost, according to the power/performance relationship defined by the energy metric. The benefit of increased pipeline depth is additional timing slack, which translates into increased frequency (performance) or reduced voltage (power). The cost of increased pipeline depth is increased latch area and power, as well as reduced throughput (IPC).

While increasing the pipeline depth can result in increased energy efficiency for the zero error rate case, increasing the pipeline depth may also increase the cost of error recovery. This is because the cost of error recovery is proportional to the depth of the pipeline for many error recovery mechanisms [Ernst et al. 2003]. Consequently, the optimal pipeline depth for an error resilient architecture is less than the optimal depth when designing for correctness. Ignoring the overhead of error recovery for an error resilient architecture can result in selection of a suboptimal pipeline depth.

We formulate an expression for the optimal pipeline depth for an error resilient architecture by modifying Hartstein and Puzak's model for optimal pipeline depth [Hartstein and Puzak 2003]. The model combines expressions for performance and power to produce an energy efficiency metric ($performance/power$). Optimal pipeline depth is found by maximizing the metric. We modify the power and performance expressions of the original model to account for the effects of error resilient design and operation on the pipeline. As such, the power and delay expressions are modified to incorporate the effects of voltage scaling, and the performance equation is modified to include the penalty of stalling to correct errors, according to the operating voltage and resulting error rate. Equation 1 gives the performance expression for the updated model, Equation 2 gives the power expression, and Equation 3 combines the expressions to form the energy efficiency metric. Table III explains the meaning of each model parameter.

$$\frac{T}{N_I} = \frac{1}{f_s a} + \frac{\gamma_h N_h p}{f_s} + \frac{\gamma_e e p T_o}{N_I} \tag{1}$$

$$P_T = (f_{cg} f_s P_d f_v^2 + P_l f_v) N_L p^\eta \tag{2}$$

Table III. Parameters for the pipeline energy efficiency model.

T	Time
N_I	Number of instructions
f_s	Frequency ($f_s = 1/(t_0 + t_p/p)$)
t_0	Latch delay
t_p	Logic delay of the pipeline
p	Pipeline depth
a	Average degree of superscalar processing
γ_h	Hazard recovery time as a fraction of pipeline delay
N_h	Number of hazards
γ_e	Error recovery time as a fraction of pipeline delay
e	Error rate (errors / cycle)
P_T	Total power
f_{cg}	Clock gating factor
P_d	Dynamic power
P_l	Leakage power
N_L	Number of latches
η	Latch growth factor
f_v	Voltage scaling factor
v_o	Normalized critical voltage
k	Regularity factor (relates path slack to pipeline depth)
w	Criticality factor (relates error acceleration to voltage)

$$BIPS^m/W \;=\; ((T/N_I)^m P_T)^{-1} \tag{3}$$

The equation describing the performance of an error-resilient architecture (Equation 1) includes an additional term ($\gamma_e e p T_o/N_I$) to model the relationship between pipeline depth and error recovery overhead. To model the impact of voltage overscaling on processor power and reliability, we introduce a voltage overscaling factor (f_v). Dynamic power scales quadratically with voltage, and leakage power scales linearly with voltage. The voltage scaling factor also influences the error rate, since path delays increase as voltage decreases. Equation 4 describes how the error rate increases as voltage is scaled down. The error acceleration parameter (w) describes the rate at which error rate increases after scaling past the critical voltage (v_o).

$$e = min(1, ((1 - f_v)/(1 - v_o))^w) \tag{4}$$

The critical voltage (v_o) depends on pipeline depth as well. This is because adding pipeline stages reduces not only the delay of each stage but also the timing slack of each stage. Figure 11 illustrates this effect. Equation 5 models the dependence of v_o on the length of the pipeline. In the equation, v_{ob} denotes the normalized critical voltage for the baseline pipeline, with depth p_b. (We assume a traditional 5-stage pipeline as the baseline.) The regularity factor (k) controls how quickly the number of negative slack paths (and thus error rate) grows with the number of pipeline stages. As the pipeline depth grows larger than the baseline pipeline depth, the amount of available timing slack decreases proportionally. Note that the equation assumes that pipelining divides all timing paths equally. All previous works on optimal pipeline depth make the same assumption.

$$v_o = 1 - (1 - v_{ob}) * (p_b/p)^k \tag{5}$$

The model described above was used to evaluate the energy efficiency of a processor architecture at different error rates and pipeline depths, in order to find the dependence of optimal pipeline depth on an error resilience mechanism. Each error resilience mechanism has a different optimal error rate. This implies that each er-

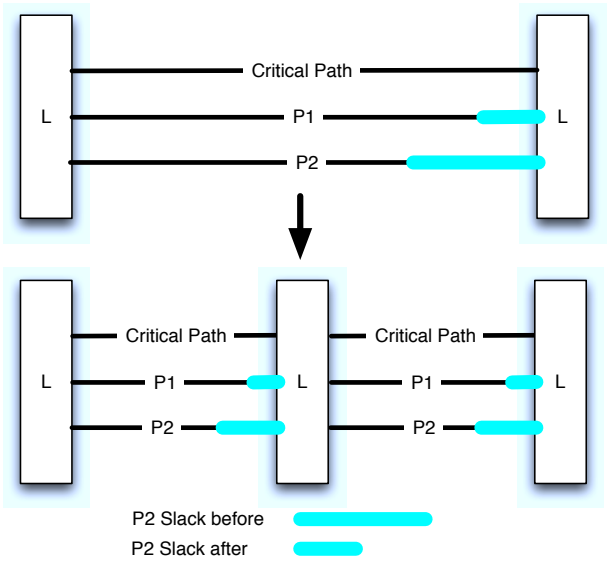


Fig. 11. Pipelining alters the slack distribution. The highlighted segments denote path slack. When pipeline depth increases, the lengths of the timing paths, and also, the amount of timing slack per stage, is reduced.

ror resilience mechanism may also have a different optimal pipeline depth. Figure 12 shows how energy efficiency varies with pipeline depth for architectures operating at different error rates. Each error rate represents a different magnitude of power savings and a different error recovery overhead. Notice that the optimal pipeline depth and energy efficiency vary significantly depending on the error rate, demonstrating that it is essential to take the error resilience mechanism into account when selecting the pipeline depth of an error-resilient architecture. In practice, the actual number of pipeline stages should be chosen not only based on the above considerations, but also based on the desired performance/power target for nominal, error free operation.

Different error resilience mechanisms have different error recovery overheads. Figure 13 shows the energy efficiency (normalized to the error free baseline), as well as the optimal pipeline depth and error rate for different values of error recovery overhead (γ_e). As the recovery overhead increases, the optimal error rate decreases. Thus, the optimal pipeline depth increases. The data demonstrate that the optimal pipeline depth depends on the error recovery overhead for a given error recovery mechanism, stressing the importance of taking the error recovery mechanism into account when selecting the pipeline depth of an error resilient architecture.

4. METHODOLOGY

We have developed a design flow that takes an RTL design through synthesis, placement, and routing, power estimation, timing analysis, area estimation, gate-level simulation, and error rate measurement. Designs are implemented with the *TSMC* 65GP library (65nm), using *Synopsys Design Compiler* [Synopsys a] for synthesis and *Cadence SoC Encounter* [Cadence c] for layout. In order to evaluate the power and performance of designs at different voltages and to provide V_{th} sizing options for synthesis, *Cadence Library Characterizer* [Cadence a] was used to generate low, nominal, and high V_{th} libraries at each voltage (V_{dd}) between $1.0V$ and $0.5V$ at $0.01V$ intervals. Power, area, and timing analyses are performed in *Synopsys PrimeTime* [Synopsys b].

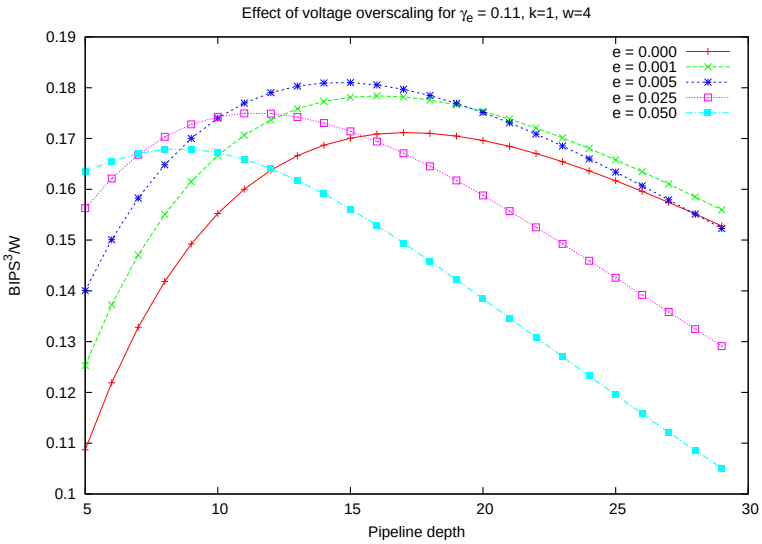


Fig. 12. Each curve shows how energy efficiency varies with pipeline depth for a given error rate. The optimal pipeline depth varies significantly, depending on the error rate.

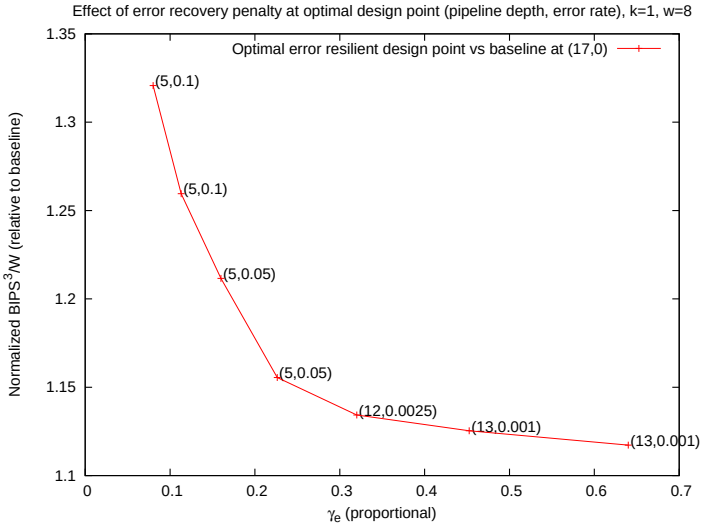


Fig. 13. The energy-optimal pipeline depth and error rate for an architecture depend on the error recovery overhead (γ_e).

Gate-level simulation is performed with *Cadence NC-Verilog* [Cadence b] to gather activity information for the design, which is subsequently used for dynamic power estimation and error rate measurement. Switching information generated during the gate-level simulation is dumped to a value change dump (VCD) file. To calculate the error rate of a design at a particular voltage, toggled nets from the VCD file are traced to find toggled paths in each cycle. The delays of toggled paths are measured, and any cycle in which a negative slack path toggles is counted as an error cycle. The error rate

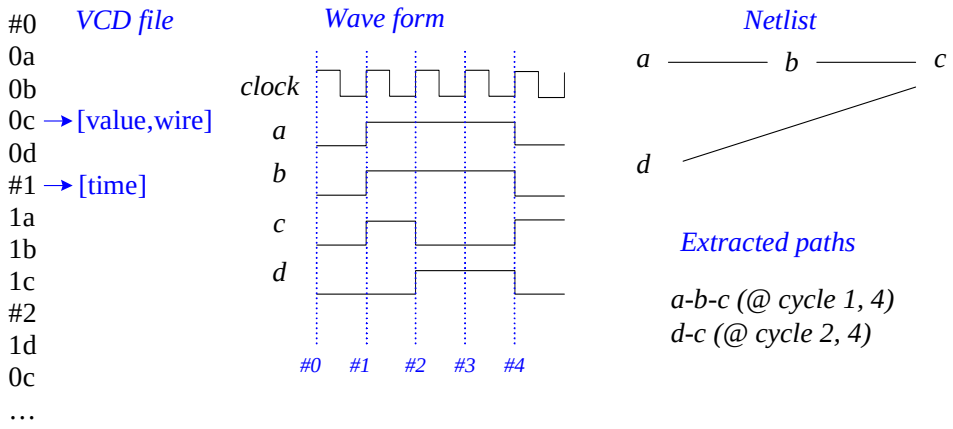


Fig. 14. VCD file format and path extraction from the VCD file.

of the design is equivalent to the cardinality of the set of error cycles divided by the total number of simulation cycles (X_{tot}), as shown in Equation 6,

$$ER = \frac{|\bigcup_{p \in P_n} \chi_{toggle}(p)|}{X_{tot}} \tag{6}$$

where P_n is the set of negative slack paths and $\chi_{toggle}(p)$ is the set of cycles in which path p toggles.

Figure 14 shows an example VCD file and illustrates the path extraction method. The VCD file contains a list of toggled nets in each cycle, as well as their new values. Toggled nets in each cycle are marked, and these nets are traversed to find toggled paths. A toggled path is identified when toggled nets compose a connected path of toggled cells from a primary input or flip-flop to a primary output or flip-flop. In Figure 14, nets a, b, and c toggle in the first and fourth cycles (#1, #4), and nets d and c toggle in the second and fourth cycles (#2, #4). Two toggled paths are extracted: $a-b-c$ and $d-c$. Paths $a-b-c$ and $d-c$ both have toggle rates of 0.4 ($|\chi_{toggle}(p)| = 2$ and $X_{tot} = 5$). If both paths have negative slack, then timing errors will occur in cycles #1, #2, and #4. Therefore, the error rate is 0.6 for this example.

In addition to inducing timing errors by increasing logic delays, voltage scaling may prompt reliability concerns for SRAM structures, such as insufficient Static Noise Margin (SNM). Fortunately, the minimum energy voltage for our processors is around 750mV, while production-grade SRAMs have been reported to operate reliably at voltages as low as 700mV [Fujimura et al. 2010]. Research prototypes have been reported to work for even lower voltages. In any case, modern processors typically employ a "split rail" design approach, with SRAMs operating at the lowest safe voltage for a given frequency [Intel Corporation 2008].

In our evaluation of general purpose processor architectures, we run instruction traces from a set of 8 SPEC benchmarks (ammp, art, equake, mcf, parser, swim, twolf, wupwise) on the processors. The traces are captured after fast-forwarding the benchmarks to their early Simpoints [Hamerly et al. 2005].

We model Razor-based error resilience in our evaluations (though our design principles are generally applicable to any timing speculative architecture). Table IV summarizes the average processor-wide static and dynamic overheads incurred by our designs that use Razor for error detection and correction. In our design flow, we measure

Table IV. Average processor-wide Razor overheads for error-tolerant architectures.

Hold buffering	Razor FF	Counterflow	Error Recovery
2% energy	23% energy	<1% energy	P cycles

Table V. Design parameters and their possible values.

I/D$ kB	ALU/FPU	INT Q-FP Q	INT/FP Regs	Ld/St Q
4,8,16,32	1,2,4	32-16,64-32	64,128	32,64

the percentage of die area devoted to sequential elements as well as the timing slack (with respect to the shadow latch clock skew of $1/2$ cycle) of any short paths that need hold buffering. When evaluating energy at the architecture level, we account for the increased area and power of Razor flip-flops, hold buffering on short paths, and implementation of the recovery mechanism. Most of the static overhead is due to Razor FFs. Buffering overhead is small, and the availability of cells with high and low V_{th} provides more control over path delay, eliminating the need for buffering on most paths. We also add energy and throughput overheads proportional to the error rate to account for the dynamic cost of correcting errors over multiple cycles. We model a counterflow pipeline Razor implementation [Ernst et al. 2003] with correction overhead proportional to the number of processor pipeline stages (P). We conservatively replace all sequential cells with Razor FFs. This conservative accounting measure means we can also claim greater immunity to aging-induced errors, e.g., due to NBTI, which can cause paths to become critical over time.

To evaluate the effects of architectural optimizations on the energy efficiency of timing speculation, we perform an exploration of the processor design space defined by the parameters found in Table V. All other parameters were chosen to be identical to the OpenSPARC core. Because it would be unreasonable to write, synthesize, layout, and test custom RTL for each of the hundreds of OpenSPARC processor configurations that we study, we instead evaluate the power, performance, and error rate of the architectures using a combination of gate and microarchitecture-level simulation.

To estimate the performance and power of each architecture, we use SMTSIM [Tullsen 1996] with Wattch [Brooks et al. 2000]. We also use Wattch to report the activity factor for each microarchitectural structure in each configuration, for each benchmark. We approximate the error rate of an architecture as the weighted sum of error rates from each of the microarchitectural components that we vary in our exploration. To obtain the component error rates, we used RTL from the OpenSPARC T1 processor [Sun]. We modified the existing OpenSPARC module descriptions to create an RTL description for each component configuration in Table V and used our detailed design flow, as described above, to measure error rate and power at different voltages. Error rate at the architecture level is given by the sum of the component error rates, where each component error rate is weighted by the activity factor captured during architecture-level simulation. While this error rate estimation technique is not as accurate as our design-level technique, it provides suitable accuracy to study the error behavior of many architectures without requiring full gate-level evaluations of many complex architectures.

5. RESULTS

In Section 2, we showed how the slack and activity distributions determine the error rate. In Section 3, we showed how architecture influences the slack and activity distributions. In this section, we demonstrate that architectural optimizations can significantly improve the energy efficiency of timing speculation, first for simple DSP

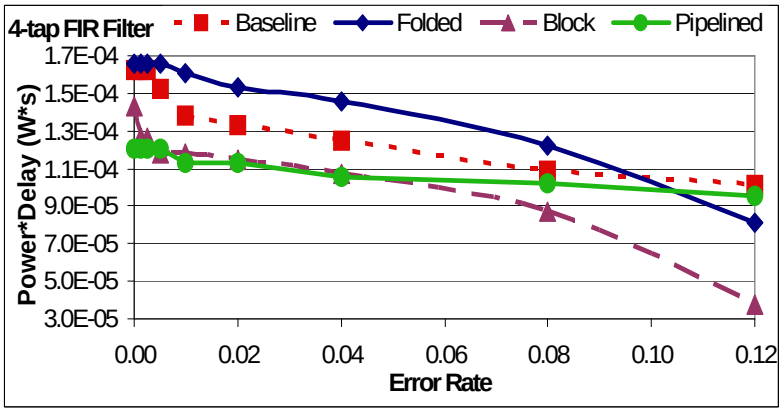


Fig. 15. Energy efficiency comparison showing crossovers between filter architectures for different voltage overscaling-induced error rates.

filter architectures, and then for general purpose processor cores (Alpha, MIPS, and OpenSPARC).

5.1. DSP Filter Architectures

First, we compare the filters with respect to different energy efficiency metrics over a range of error rates to observe how the optimal architecture changes for error free and error resilient operation. Figure 15 compares the filter architectures in terms of power-delay product. The low capacitance, shorter paths, and highly regular slack distribution of the pipelined architecture allow it to achieve better energy efficiency for error free operation. However, the clustering of path delays in the pipelined design causes the error rate to increase rapidly once errors begin to occur. This causes power savings to quickly level off for the pipelined architecture. Consequently, the blocked architecture becomes more energy efficient at moderate error rates. While higher complexity and deeper logic depth limit the amount of voltage scaling for correct operation with the blocked architecture, low activity allows the error rate of the filter to stay lower longer as voltage is reduced, enabling an extended range of power savings for the blocked design. The baseline and folded architectures do not minimize energy over any range of error rates, due to the high activity and regularity of the folded architecture and the increased sensitivity to voltage scaling of the baseline (without the benefit of reduced activity that the blocked architecture has).

The choice of the efficiency metric (which expresses the relative importance of power and performance to the architect) influences which architecture is most efficient at different error rates. Figure 16 compares the filters in terms of power efficiency. Both the pipelined and folded architectures are approximately the same in terms of sensitivity to voltage scaling and regularity. The pipelined filter has the best power efficiency for low error rates, due to extra slack afforded by the increased regularity of the slack distribution. This enables more scalability before the onset of errors. However, regularity results in a steep increase in the error rate, allowing the folded architecture to gain the power efficiency edge for mid-range error rates. The folded architecture has reduced complexity, fewer paths, and less fanout, resulting in the best scalability of any architecture. It also has low power consumption due to simple logic and low area (Figure 5). Nevertheless, though it has better scalability and low power, once it starts making errors, its error rate increases dramatically, due to increased activity. This allows the block filter, with reduced activity, to take the lead at high error rates.

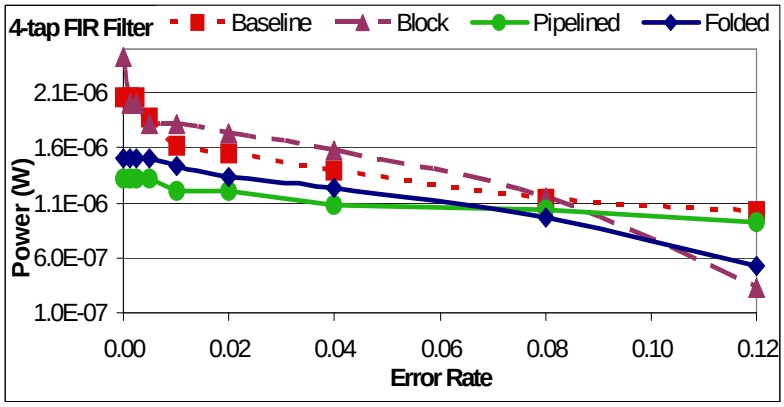


Fig. 16. Power efficiency comparison showing crossovers between filter architectures for different voltage overscaling-induced error rates.

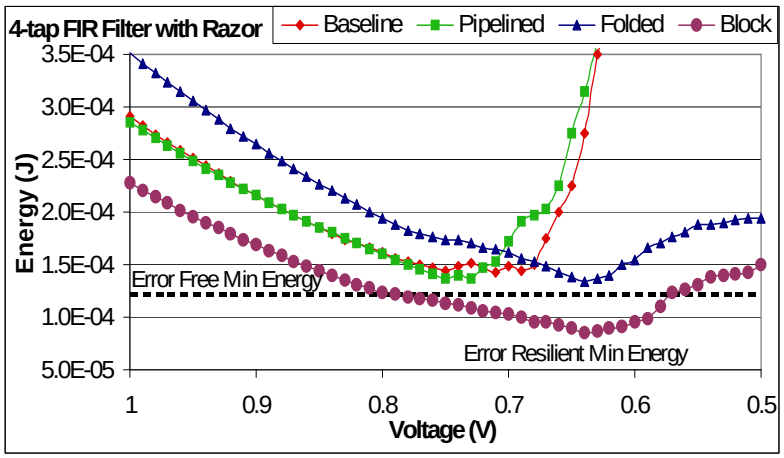


Fig. 17. Minimum energy for correct operation (denoted by the dotted line) is achieved with the pipelined architecture. When Razor is used to enable timing speculation, the blocked architecture minimizes energy, demonstrating that the architecture that minimizes energy by exploiting error resilience is different than the optimal architecture for error free operation.

Figure 17 compares the energy consumption of Razor implementations of the filter architectures. While the pipelined architecture has the best energy efficiency for error free operation, the blocked architecture consumes the least energy for Razor-based timing speculation (29% less energy than the error free pipelined filter). The reduced activity of the blocked filter allows more voltage scaling before the energy-optimal error rate for Razor is reached. Furthermore, the blocked filter, having fewer flip-flops and pipeline stages, has reduced implementation and recovery overheads for Razor, making it a more efficient choice for exploiting error resilience. Note that other filter architectures, including the optimal architecture for correct operation, do not achieve energy reduction with Razor, either due to static overheads (Razor flip-flops and buffering of short paths) or dynamic overheads (power and energy costs of error recovery). This result demonstrates the importance of timing speculation-aware architectural optimization techniques.

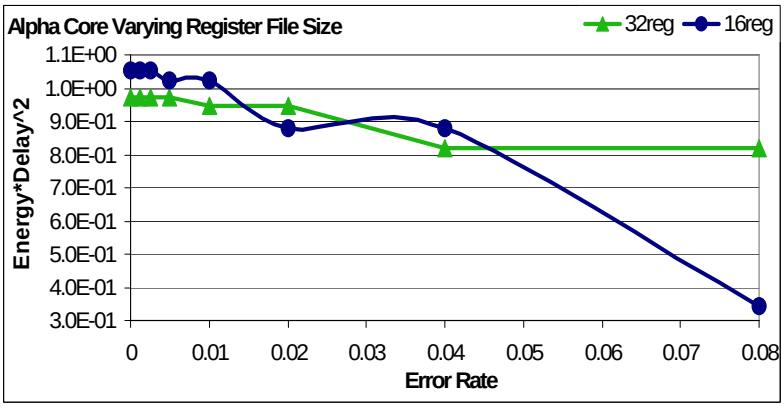


Fig. 18. A larger register file increases performance, but also results in increased regularity and activity, hindering voltage scaling and energy efficiency at larger error rates.

To summarize, our experiments with different DSP filter architectures validate our claim that the optimal architecture for correctness may not be efficient for exploiting timing error resilience. The results also confirm that architectural optimizations that alter the slack and activity distributions have the potential to increase the energy efficiency of timing speculation.

5.2. General Purpose Processor Architectures

In this section, we evaluate how changes to Alpha, MIPS, and FabScalar architectures that affect their slack and activity distributions (as described in Section 3.2) influence their energy efficiency for timing speculation. Figure 18 compares the energy efficiency of the Alpha processor for varying register file sizes. The design with a larger register file has higher throughput and better energy efficiency when both processors operate error free. However, the higher average path delay and path delay regularity associated with the larger register file hinder voltage scaling and energy efficiency at non-zero error rates. Furthermore, high performance corresponds to higher activity, which causes error rate to increase more quickly for the processor with the larger register file.

Because of the higher throughput of the 32-register design, there is a small range of error rates over which the 32-register design regains the efficiency advantage when the many regular paths in the 16-entry register file begin to have negative slack, and error rate begins to increase more rapidly. However, the design with fewer registers is able to scale to a much lower voltage for higher error rates because of its lower activity, increased average slack, and more gradually increasing error rate resulting from reduced regularity of the slack distribution.

Figure 19 shows energy consumption for the Alpha core with Razor-based timing speculation, confirming that the architecture with a smaller register file exploits timing error resilience more efficiently. The 16-register architecture reduces energy by 21% with respect to the optimal architecture for correctness, while the optimal error free architecture barely procures any energy savings (2%) when using Razor. Again, we observe significantly improved benefits from optimizing the architecture to exploit timing error resilience while the optimal error free architecture sees only a small energy reduction with timing speculation.

We evaluated the energy efficiency of the MIPS processor at different error rates when the superscalar width (and number of ALUs) was increased. The main effect

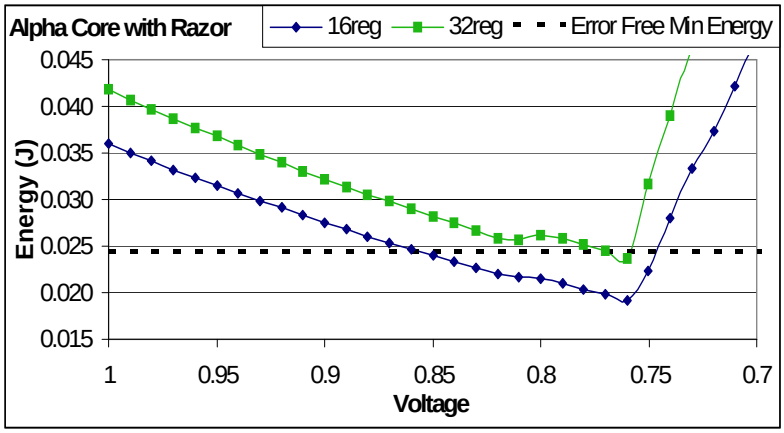


Fig. 19. The 16-register design, having reduced regularity and activity, achieves significant energy savings with Razor, while the 32-register design, which was optimal for correct operation, achieves almost no benefit.

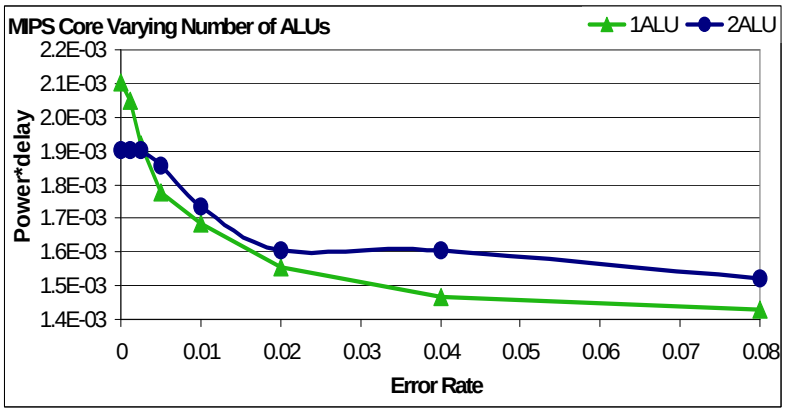


Fig. 20. Increased throughput for the multiple-ALU architecture results in better energy efficiency for error free operation, but increased activity results in worse efficiency at most non-zero error rates.

on the error rate from increasing the superscalar width of the processor is due to increased activity. Not only does this architectural change increase the throughput (and thus the activity factor) of the processor, increasing the superscalar width also increases the number of paths that are active when the processor is able to exploit ILP on multiple ALUs.

Figure 20 compares a single-ALU version of the MIPS architecture against one with two ALUs. The multiple-ALU architecture has better energy efficiency for correct operation due to increased throughput (up to 21% throughput reduction for the scalar case, 13% on average). However, when operating at non-zero error rates, the increased activity and complexity of the multiple-ALU architecture causes the error rate to increase more rapidly, limiting voltage scaling for higher error rates. More instructions per cycle means more errors per cycle, and more active ALUs means more paths causing errors when voltage is scaled down. The higher activity of the multiple-ALU architecture makes the single-ALU architecture more energy-efficient for most non-zero error rates.

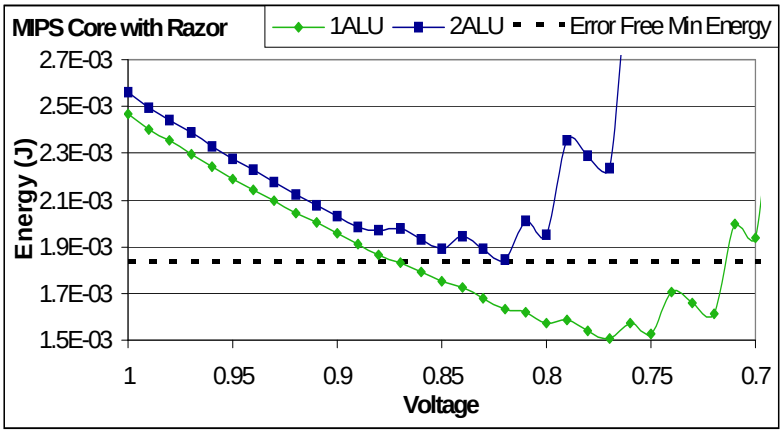


Fig. 21. The more complex superscalar architecture has throughput and energy benefits for error free operation but fails to achieve any energy savings with Razor-based timing speculation. The simpler, scalar design achieves substantial energy savings with Razor.

Figure 21 confirms that the scalar design exploits timing error resilience more efficiently. Whereas the superscalar pipeline achieves better energy efficiency for correct operation, increased complexity and activity, along with increased implementation and recovery overheads for Razor, prevent the multiple-ALU architecture from achieving energy benefits with Razor. The single-ALU architecture has a more gradually increasing error rate, allowing extended voltage scalability and an 18% energy reduction with respect to the energy-optimal architecture for correctness.

To evaluate the potential benefits of manipulating the pipeline depth of an error resilient processor, we would like to explore optimal pipelining for an entire processor core. However, writing RTL for an entire processor for different pipeline depths is a challenging and time-consuming task. As far as we know, no open source processor RTL exists in which the pipeline depth can be scaled arbitrarily. The closest approximation we found is FabScalar, in which certain pipeline stages can be subdivided into multiple stages. We evaluate the effects of manipulating the pipeline depth in an error resilient FabScalar processor by comparing versions of the pipeline with Issue depths 1 and 2. The Issue stage has by far the most critical paths in the FabScalar processor.

Figure 22 compares the energy efficiency of the pipelines with Issue depth (ID) 1 and 2 at different error rates. For correct operation, the ID 2 pipeline has 9% better energy efficiency, because increasing the pipeline depth of the Issue stage allows the pipeline to be optimized for a higher frequency (or lower voltage), and achieve higher throughput (or lower power). As voltage is scaled down, however, the error rate of the ID 2 pipeline increases more quickly. This is because dividing a path with a pipeline latch not only partitions its logic between two stages, it also partitions the path's timing slack between two stages. Thus, pipelining reduces the average amount of timing slack in the pipelined stages, so that more paths fail sooner when voltage is scaled down. Due to the steeper increase of error rate in the ID 2 pipeline, the ID 1 pipeline has better energy efficiency at higher error rates.

Figure 23 compares the energy of the ID 1 and ID 2 pipelines with Razor. The ID 1 pipeline consumes 13% less energy with Razor than the ID 2 pipeline. This is due to two factors. First, as discussed above, the error rate of the ID 2 pipeline increases faster as voltage is scaled down, resulting in less voltage overscaling when Razor is used. Second, the ID 2 pipeline has a higher average error recovery cost, due to the increased

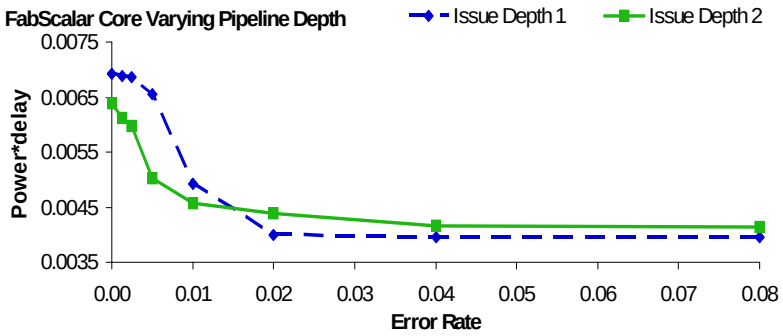


Fig. 22. Increased pipeline depth enables the ID 2 pipeline to achieve higher energy efficiency for correct operation. However, increased pipeline depth causes the error rate to increase more quickly, and the ID 1 pipeline has better energy efficiency for higher error rates.

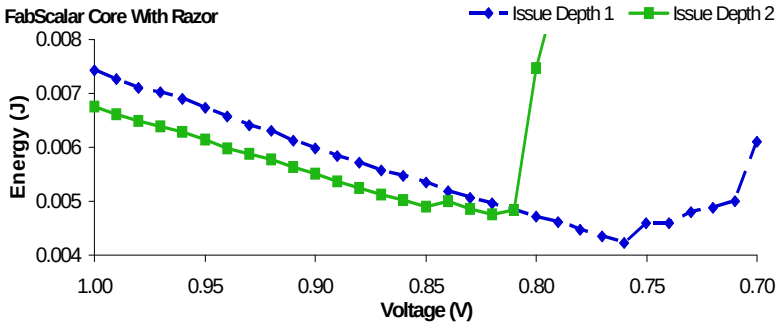


Fig. 23. The shallower pipeline (ID 1) achieves better energy efficiency with Razor, due to lower error recovery overhead and more gradual error rate increase, afforded by increased slack.

average cost of pipeline flushing during error correction. These results confirm that ignoring the error resilience mechanism when selecting the pipeline depth of an error resilient processor can lead to energy inefficiency. An error resilient processor should use a shallower pipeline depth than a processor that does not exploit error resilience. We expect the potential benefits of optimizing the pipeline depth to increase with more flexibility in the available pipeline depth.

To summarize, our experimental results with Alpha, MIPS, and FabScalar cores further confirm the benefits of architecting to exploit timing error resilience and demonstrate that architectures that have been optimized for energy-efficient error free operation see little or no energy benefits when exploiting timing speculation. These results re-confirm that changing the slack and activity distributions with architectural optimizations can improve the energy efficiency of timing speculation.

5.3. Design Space Exploration for OpenSPARC

In the previous section, we performed analyses of various architectural optimizations to validate our insights on resilience-optimized architectures. In this section, we present an exploration of the design space for resilience-optimized general purpose processor architectures to further confirm that the benefits of exploiting error resilience can be significantly enhanced by optimizing the architecture for timing speculation.

In our exploration, we evaluated nearly 400 architectural configurations by varying instruction and data cache sizes (ic,dc), the number of integer and floating point functional units (alu), instruction queue size (q), the number of physical registers (reg), and the size of the load/store queue (lsq). A tuple (ic,dc,alu,q,reg,lsq) denotes the parameters of a particular architecture of interest. For each architecture, we estimated power, performance, and error rate as described in Section 4 and used these data to characterize energy consumption of the architectures at different error rates.

Figure 24 compares the energy efficiency of three architectures that emerged as the optimal design points for different ranges of error rates. The optimal architecture for error free operation (ic8,dc16,alu1,q32,reg128,lsq64) has a moderate instruction cache size, larger data cache, and maximum sizes for queues and register files. For error free operation, this configuration achieves good performance and has low power, making it the energy-optimal architecture. However, the large cache and register file sizes result in a highly regular slack distribution, so that many paths fail in groups as voltage is scaled. The increased complexity and deeper logic of large instruction and load/store queues, while increasing performance, also makes the architecture fail sooner with overscaling.

For low to mid-range error rates, a different energy-optimal architecture (ic8,dc8,alu1,q32,reg128,lsq32) emerges. This architecture has a smaller data cache and load/store queue, resulting in reduced regularity and complexity. The immediate effect of increased spread and average slack in the slack distribution is that voltage can be scaled further before the error rate begins to increase dramatically, resulting in more power savings for timing speculation before reaching an energy-optimal error rate. When operating at low to mid-range error rates, the resilience-optimized architecture has 6% energy (W/IPC) benefits over the optimal error free architecture. Energy reduction is mainly due to enhanced power scaling (15% power reduction, on average), since throughput is reduced by 7% with respect to the optimal error free architecture. Thus, energy benefits will increase for a metric that weights power more heavily.

Note that compared to the optimal error free architecture, the optimal for low to mid-range error rates decreases the size of the load/store queue (LSQ), but not the instruction queue. This is primarily because the LSQ becomes full more often than the IQ, resulting in a longer dynamic critical path that limits voltage scaling. To a second degree, the size of the instruction queue also has a more pronounced effect on performance.

For higher error rates (around 6% and up), an architecture with minimum-sized data cache and register file (ic8,dc4,alu1,q16,reg64,lsq32) consumes the least energy. In addition to the significantly reduced regularity of the slack distribution (reduced area devoted to regular structures and reduced criticality of regular structures), this architecture also has small queue sizes with decreased complexity and better scalability. The throughput of this architecture is an additional 27% lower than the correctness-optimized baseline; however, the corresponding reduced activity actually has some benefit in terms of energy, since it results in a more gradually increasing error rate as voltage is reduced. The optimal architecture for higher error rates has the most gradually increasing error rate, enabling significant voltage scaling and an average of 38% energy reduction at higher error rates, with respect to the optimal error free architecture.

Graceful failure in the presence of overscaling translates into a lower dynamic energy overhead when exploiting Razor-based timing speculation. Figure 25 echoes the results of our previous experiments, showing that the optimal architecture for correctness achieves only minor (5%) energy benefits with Razor, while the resilience-optimized architecture reduces energy by 25% with respect to the error free minimum energy.

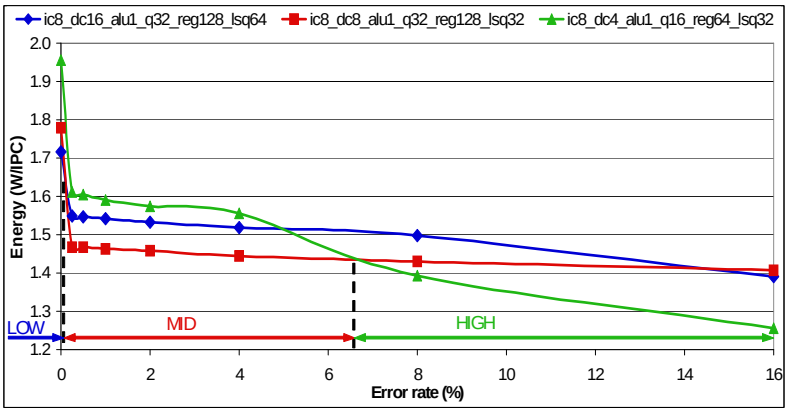


Fig. 24. The energy-optimal architecture is different for different ranges of voltage overscaling-induced error rates.

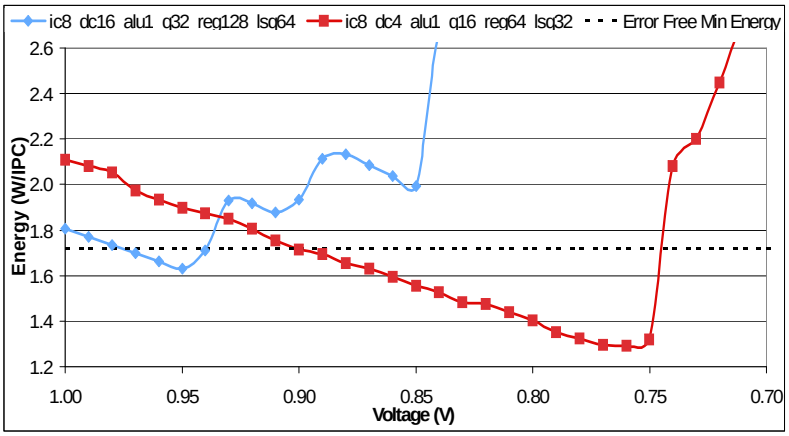


Fig. 25. The resilience-optimized architecture achieves significant energy savings with Razor, while the optimal error free architecture sees only minor benefits.

Table VI. Throughput Reduction for Resilience-aware Optimizations.

REG ($32 \rightarrow 16$)	ALU ($2 \rightarrow 1$)	OpenSPARC
6%	21%	27%

Since resilience-optimized architectures typically reduce the sizes of regular structures like caches and use simpler architectural features that, e.g., may throttle ILP, they may sacrifice some throughput in order to reduce energy. Table VI shows throughput reduction for the resilience-aware optimizations we evaluated in this section. Since we employed voltage overscaling, we demonstrate power and energy savings at the expense of some throughput. Note, however, that we could also demonstrate performance gains by overscaling frequency rather than voltage.

6. RELATED WORK

This work demonstrates that the slack and activity distributions of a processor influence the error rate and the efficiency of timing speculation. Furthermore, architectural

optimizations have the potential to alter the slack and activity distributions to increase the energy efficiency of timing speculation.

The closest related work [Sartori and Kumar 2010; Narayanan et al. 2010] compares the efficiency of different adder architectures at different error rates. Our work is at the architecture level, and we provide guidance on how to optimize the architecture of a processor to alter both the slack and activity distributions and increase the efficiency of timing speculation. We also perform a full design space exploration for resilience-aware general purpose processor architectures.

A related body of work exists at the level of design techniques that optimize circuit modules for a target error rate [Kahng et al. 2010b] or to fail gracefully in response to voltage overscaling [Kahng et al. 2010c; 2010a] through cell-based optimizations. Whereas these design-level techniques reshape the slack distribution or reliability of a circuit module, the architecture-level techniques presented in this paper target both the slack and activity distributions of a processor. Also, architecture-level optimizations can have a greater impact on the slack distribution of a processor, since for a design-level technique, the microarchitecture and synthesized netlist are fixed, and the ability of cell sizing to reshape path slack may be limited. This work demonstrates that architecture-level changes can improve the energy efficiency of a timing speculative architecture. A promising direction of work is to investigate co-optimization at the architecture and design levels to reshape the slack and activity distributions and maximize the energy efficiency benefits provided at each level.

Another relevant related work [Liang and Brooks 2006] explores microarchitectural parameter selection to optimize processor performance in the presence of process variations. The authors aim to reduce performance loss due to process variations by adding slack to the critical paths of a processor where possible. However, unlike our work, [Liang and Brooks 2006] attempts to *prevent* the onset of errors; they are not concerned with the activity distribution of the processor or scalability *after* the point where errors begin to occur. Our work, on the other hand, focuses on the error rate distribution. Since they are only concerned with correct operation, they have no reason to consider the activity distribution of a processor or the shape of the slack distribution. We consider all these factors in our approach to architecture, since they determine the energy benefits achievable through the exploitation of timing speculation

7. CONCLUSIONS

The energy inefficiencies of traditional, conservative design approaches have led to the introduction of error resilient design techniques that relax correctness in order to save power and energy. Until now, these design techniques have been applied to architectures that have been optimized for correctness.

In this work, we have demonstrated that the energy-optimal error free architecture may not be the optimal architecture for exploiting timing error resilience. In other words, one would make different, sometimes counterintuitive, architectural design choices when optimizing a processor to exploit timing speculation than when optimizing for correct operation. Consequently, the desired error rate and the error resilience mechanism should be taken into account when choosing the architecture for a timing speculative design. In addition to characterizing the effects of architectural optimizations on the slack and activity distributions, we have demonstrated that they can change the error rate behavior. Furthermore, we have demonstrated with experimental results for several DSP filter and general purpose architectures that optimizing architecture to exploit timing error resilience can significantly increase the energy efficiency of timing speculation. Energy efficiency benefits of up to 29% are achieved for Razor-based timing speculation.

ACKNOWLEDGMENTS

The authors would like to acknowledge the many collaborators, colleagues, and reviewers that helped in the development, refinement, and exploration of the ideas presented in this paper. Our work on Stochastic Computing has been generously supported by GRC, GSRC, NSF, Intel, and LLNL.

REFERENCES

BERTACCO, V., AUSTIN, T., AND WAGNER, I. *Bug Underground*. University of Michigan.

BOWMAN, K., TSCHANZ, J., WILKERSON, C., LU, S., KARNIK, T., DE, V., AND BORKAR, S. 2009. Circuit techniques for dynamic variation tolerance. In *DAC*. 4–7.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA*.

Cadence. *Cadence LC User's Manual*. Cadence.

Cadence. *Cadence NC-Verilog User's Manual.* Cadence.

Cadence. *Cadence SOCEncounter User's Manual*. Cadence.

CHOUDHARY, N., WADHAVKAR, S., SHAH, T., MAYUKH, H., GANDHI, J., DWIEL, B., NAVADA, S., NAJAFABADI, H., AND ROTENBERG, E. 2011. Fabscalar: composing synthesizable rtl designs of arbitrary cores within a canonical superscalar template. In *ISCA*. 11–22.

DHAR, S., MAKSIMOVIC, D., AND KRANZEN, B. 2002. Closed-loop adaptive voltage scaling controller for standard-cell asics. *ISLPED*.

ERNST, D., KIM, N. S., DAS, S., PANT, S., RAO, R., PHAM, T., ZIESLER, C., BLAAUW, D., AUSTIN, T., FLAUTNER, K., AND MUDGE, T. 2003. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO*. 7.

FUJIMURA, Y., HIRABAYASHI, O., SASAKI, T., SUZUKI, A., KAWASUMI, A., TAKEYAMA, Y., KUSHIDA, K., FUKANO, G., KATAYAMA, A., NIKI, Y., AND YABE, T. 2010. A configurable sram with constant-negative-level write buffer for low voltage operation with $0.149\mu m^2$ cell in 32nm high-k/metal gate cmos. In *ISSCC*.

GRESKAMP, B. AND TORRELLAS, J. 2007. Paceline: Improving single-thread performance in nanoscale cmps through core overclocking. *PACT*, 213–224.

GRESKAMP, B., WAN, L., KARPUZCU, W., COOK, J., TORRELLAS, J., CHEN, D., AND ZILLES, C. 2009. Blueshift: Designing processors for timing speculation from the ground up. *HPCA*.

HAMERLY, G., PERELMAN, E., LAU, J., AND CALDER, B. 2005. Simpoint 3.0: Faster and more flexible program analysis. In *JILP*.

HARTSTEIN, A. AND PUZAK, T. 2003. Optimum power/performance pipeline depth. In *MICRO*. 117.

Intel Corporation 2008. *Intel Atom Processor Z5xx Series*. Intel Corporation.

KAHNG, A., KANG, S., KUMAR, R., AND SARTORI, J. 2010a. Designing processors from the ground up to allow voltage/reliability tradeoffs. In *HPCA*.

KAHNG, A., KANG, S., KUMAR, R., AND SARTORI, J. 2010b. Recovery-driven design: A methodology for power minimization for error tolerant processor modules. In *DAC*.

KAHNG, A., KANG, S., KUMAR, R., AND SARTORI, J. 2010c. Slack redistribution for graceful degradation under voltage overscaling. In *ASPDAC*.

KEHL, T. 1993. Hardware self-tuning and circuit performance monitoring. *ICCD*, 188–192.

LIANG, X. AND BROOKS, D. 2006. Microarchitecture parameter selection to optimize system performance under process variation. In *ICCAD*. 429–436.

NARAYANAN, S., SARTORI, J., KUMAR, R., AND JONES, D. 2010. Scalable stochastic processors. In *DATE*.

PALACHARLA, S., JOUPPI, N., AND SMITH, J. 1997. Complexity-effective superscalar processors. *ISCA*.

PAN, Y., KONG, J., OZDEMIR, S., MEMIK, G., AND CHUNG, S. W. 2009. Selective wordline voltage boosting for caches to manage yield under process variations. In *DAC*. 57–62.

SARANGI, S., GRESKAMP, B., TIWARI, A., AND TORRELLAS, J. 2008. Eval: Utilizing processors with variation-induced timing errors. *MICRO*, 423–434.

SARTORI, J. AND KUMAR, R. 2010. Overscaling-friendly timing speculation architectures. In *GLSVLSI*.

Sun. *Sun OpenSPARC Project*. Sun.

Synopsys. *Synopsys Design Compiler User's Manual*. Synopsys.

Synopsys. *Synopsys PrimeTime User's Manual*. Synopsys.

TULLSEN, D. M. 1996. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*.