Appears in the 40th International Conference on Dependable Systems and Networks (DSN ’10)

A Unified Model for Timing Speculation: Evaluating the Impact of Technology Scaling, CMOS Design Style, and Fault Recovery Mechanism

Marc de Kruijf, Shuou Nomura, Karthikeyan Sankaralingam
Vertical Research Group
University of Wisconsin – Madison
{dekruijf, nomura, karu}@cs.wisc.edu

Abstract

Due to fundamental device properties, energy efficiency from CMOS scaling is showing diminishing improvements. To overcome the energy efficiency challenges, timing speculation has been proposed to optimize for common-case timing conditions, with errors occurring under worst-case conditions detected and corrected in hardware. Although various timing speculation techniques have been proposed, no general framework exists for reasoning about the trade-offs and high-level design considerations of timing speculation.

This paper develops two models to study the end-to-end behavior of timing speculation: a hardware-level efficiency model that considers the effects of process variations on path delays, and a complementary system-level recovery model. When combined, the models are used to assess the impact of technology scaling, CMOS design style, and fault recovery mechanism on the efficiency of timing speculation. Our results show that (1) efficiency gains from timing speculation do not improve as technology scales, (2) ultra-low power (sub-threshold) CMOS designs benefit most from timing speculation – we report a 47% potential energy-delay reduction, and (3) fine-grained fault recovery is key to significant energy improvements. The combined model uses only high-level inputs to derive quantitative energy efficiency benefits without any need for detailed simulation, making it a potentially useful tool for hardware developers.

1. Introduction

Due to fundamental limitations in reducing threshold voltage and gate capacitance, technology scaling in the CMOS roadmap is increasingly driven by materials innovation at each technology node [1]. Furthermore, increasing process, voltage, and temperature (PVT) variations and aging effects require circuits and systems designers to embed considerable design guardband to prevent any failure, diminishing the benefit of technology scaling. As a result, energy efficiency from CMOS scaling is showing diminishing or practically no improvements. Energy efficiency gains from scaling have been a fundamental driver of VLSI systems. Hence, diminishing energy efficiency gains has disruptive implications for the design of computing systems.

Timing speculation has been proposed to help overcome this energy efficiency limitation [3, 5]. Under timing speculation, circuits are designed to operate correctly under common-case timing conditions but are allowed to fail under dynamically worst-case conditions. Classical scaling benefits can still be obtained for the common-case behavior, while the system provides some mechanism for error detection and correction to recover from infrequent failures. While individual microarchitecture, CAD, and device-level approaches have been proposed to analyze timing speculation, no framework exists to analyze timing speculation at the level of the overall system.

In this paper, we propose such a general, system-level framework for the analysis of timing speculation. We develop a unique end-to-end model that considers (1) CMOS technology scaling, (2) CMOS design style (i.e. high performance vs. low power), and (3) the fault recovery system. The following paragraph describes our approach.

First, we develop a hardware model for timing speculation that maps error rate to energy efficiency gains, accounting for process variation. By varying circuit operation parameters, we apply this model to a spectrum of CMOS design styles and project to different technology node sizes using the ITRS roadmap [1]. Second, we develop a system-level recovery model to derive the overheads of recovery in the event of an error and apply it to a spectrum of recovery systems. We finally combine this model with our hardware model to yield a system-level model that provides realistic upper-bounds on achievable energy efficiencies.

We report three key findings using this model: (1) timing speculation is largely unaffected by future technology scaling, (2) timing speculation is most beneficial when applied to ultra low-power hardware designs, and (3) coarse-grained recovery systems show only limited energy efficiency im-

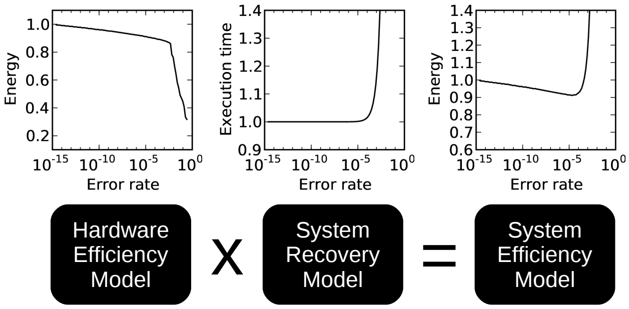


Figure 1. Our system-level model breakdown.

provements. In addition, our modeling framework is a contribution in its own right. It is the first unified model that covers technology scaling, device-level behavior, and architecture organization to evaluate the impact of timing speculation. Moreover, the model takes only four high-level inputs. It thus provides designers with a system evaluation tool that can be used for rapid analysis of timing speculation without detailed circuit or microarchitecture simulation.

2. Overview

Figure 1 gives an overview of the two models we develop in this paper and how they are combined. We first develop a hardware efficiency model that derives the relationship between error rate and hardware energy efficiency given two input parameters: (1) a hardware path delay distribution, and (2) the effect of process variation on path delay for a given hardware design and technology process.

The model is described in Section 3, and in Section 4 we use projections from the ITRS roadmap [1] to derive the parameter that models the effects of process variation for different hardware design styles and technology processes. We consider three design styles, which we refer to as *high performance CMOS*, *low power CMOS*, and *ultra-low power CMOS*, and explore both 45nm and 11nm technology processes, where 11nm represents the end of the CMOS roadmap according to ITRS projections.

Next, we develop a separate system recovery model that statistically derives the overheads of error recovery. The model is generally applicable to all *backward error recovery* (checkpoint-based) systems. These systems are in contrast to *forward error recovery* systems, such as systems using triple modular redundancy (TMR), which typically have much higher resource overheads and hence we do not study them here. Our model determines overheads using two input parameters: (1) the time between checkpoints and (2) the checkpoint restoration cost. We present the model in Section 5 and in Section 6 we describe a spectrum of recovery systems and derive the input parameters for each.

Finally, in Section 7 we present results applying our hardware efficiency model to each CMOS design style and technology node, and our recovery model to each recovery system. We also describe how the models are combined to produce our aggregate model for system efficiency, and present the results matching up the hardware and recovery designs we study. The final model has the following four inputs:

1. A hardware path delay distribution.
2. The effect of process variation on path delay, which can be derived from ITRS projections for:
   - a given CMOS design style.
   - a given CMOS process technology.
3. The time between recovery checkpoints.
4. The time to restore a checkpoint.

In Section 8, we present the limitations of the model and the limitations of our results. Section 9 discusses related work and finally Section 10 concludes.

3. Hardware efficiency model

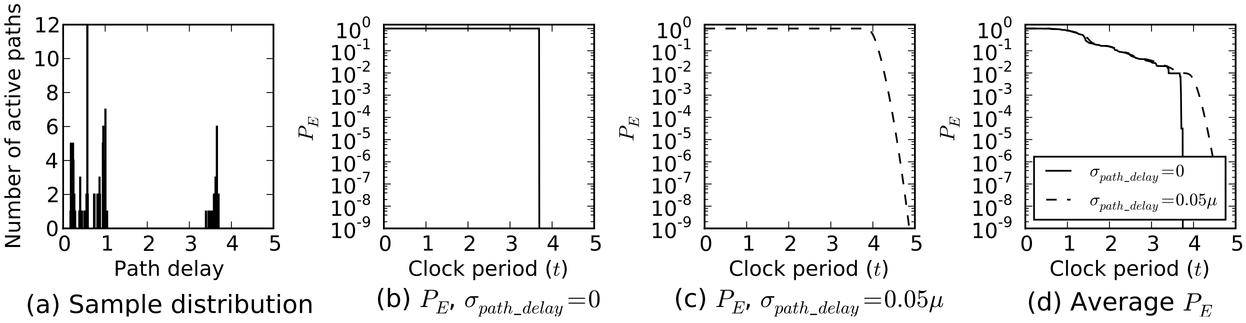
Timing speculation relaxes technology scaling constraints and addresses conservative guardbands required to combat PVT variations. It also allows hardware to execute at higher frequency or lower voltage than in the worst case. This reduction in voltage and/or increase in clock frequency improves energy efficiency at the expense of introducing errors into the system. We assume that the efficiency goal is to minimize *energy-delay*, measured by the energy-delay product, which we abbreviate as EDP. Since $energy = power \times delay$:

$$EDP = energy \times delay = power \times delay^2 \qquad (1)$$

In this section, we derive a function, EDP_{hw}, to encapsulate our hardware efficiency model for timing speculation. EDP_{hw} captures the efficiency of the hardware by mapping hardware error rate to hardware EDP, relative to a baseline without timing speculation. The model can be trivially extended to other metrics such as energy-delay squared.

Before we proceed, however, we first highlight the two factors that allow the basic concept of timing speculation. First, combinational logic delays (and hence arrival times for the clocked elements like flip-flops) vary due to the application and input data. Timing speculation allows the clock period to be lower than the worst-case arrival time assumed during design. Second, PVT variations introduce variability in gate delays, and hardware designed using timing speculation need not account for worst-case variability. In this paper, we refer to these two factors as the *Application Factor* and the *Variability Factor*, respectively.

To derive EDP_{hw}, we measure the Application Factor using an empirically measured path delay distribution that represents logic-level and application-level sources of variability. For the Variability Factor, we specifically model



(a) Sample distribution (b) P_E, $\sigma_{path\_delay}=0$ (c) P_E, $\sigma_{path\_delay}=0.05\mu$ (d) Average P_E

Figure 2. Effect of process variation on the relationship between clock period and error probability.

process variability, and measure how process variations affect timing speculation. Although we consider only process variations, the model is extendable to other forms of PVT variation as well. In summary, our model takes the following two input parameters:

1. A representative processor path delay distribution.
2. A value σ_{path_delay} that represents the effect of process variation on path delay for a given hardware design and technology process. Path delay variations due to process variation are modeled as normally distributed [14], where σ_{path_delay} is the standard deviation of the normal distribution. It can be derived from gate-delay equations, CAD tools, or other simulation techniques.

With these parameters, we derive EDP_{hw} in three steps. First, we use our representative path delay distribution and σ_{path_delay} to derive a function P_E that maps a clock period to a per-cycle error rate. Second, we derive a function $convert$ that converts a relative clock period to a relative EDP. Finally, given P_E^{-1}, the inverse function of P_E, we compute EDP_{hw} using the following equation:

$$EDP_{hw}(rate) = convert(\frac{P_E^{-1}(rate)}{P_E^{-1}(rate_{base})}) \qquad (2)$$

The variable $rate_{base}$ represents the baseline error rate for a system without timing speculation. In this paper, we assume a baseline of 3.8×10^{-16}, which corresponds with roughly one timing error every 30 days on a single 1GHz processor core[1].

Step 1: The derivation of P_E. To measure path delay distributions for a representative processor design, we used delay-aware simulation of the OpenRISC processor with Synopsys VCS and the Synopsys 90nm technology library. Although we gathered these data for several micro-benchmarks, we found little variability since paths were heavily re-used across all of the benchmarks. Our ultimate conclusion was that the choice of micro-benchmark was of limited relevance for our simple OpenRISC processor core. Therefore, in this paper we use the path delay distribution for only a single representative micro-benchmark, H.264 decoding, which was derived originally from the PARSEC benchmark suite. For this application, Figure 2(a) shows a path delay distribution for a single representative cycle.

The P_E mapping of clock period to error rate for the single-cycle distribution from Figure 2(a) is shown in Figure 2(b). It assumes no process variation. The optimal clock period for this cycle is exactly the latency of the worst path delay, in this case 3.71 nanoseconds. This clock period has error probability $P_E(3.71) = 0$. Any shorter clock period produces an error with probability 1, *e.g.* $P_E(3.70) = 1$. When path delay distributions are considered for multiple cycles – each cycle with a different optimal clock period – then P_E for all cycles becomes a monotonically decreasing function as the probability is averaged across all cycles, as shown by the solid curve in Figure 2(d). This monotonically decreasing function is illustrative of the Application Factor described earlier in this section, and represents the opportunity for timing speculation in the absence of variations.

Process variations introduce device-level variability and further opportunity for timing speculation. We develop a model that incorporates path delay variations, modeled using the input parameter σ_{path_delay}, into the derivation of P_E. The model is derived as follows. First, we observe that path delay is composed of wire delay and gate delay. Let w (with σ_w) denote wire delay and g (with σ_g) gate delay. Also, let $path\_delay_p$ denote the path delay of path p, with $path\_delay_p = w_p + g_p$. Since $path\_delay$ is normally distributed, $\sigma_{path\_delay} = \sqrt{\sigma_w^2 + \sigma_g^2}$, applying the standard result for the sum of normal distributions. We let wires account for 35% of path delay and make them immune to process variations [14], so that $\sigma_w = 0$ and hence $\sigma_g = \sigma_{path\_delay}$.

We now derive P_{E_c} for a given cycle c. Intuitively, the probability of *no* error occurring while executing path p at clock period t is the sum of the probabilities where $path\_delay_p$ is less than t. Mathematically, this probability is given by $F(t - w_p, g_p, \sigma_g)$, where $F(x, \mu, \sigma)$ is the *CDF*

[1] Although a baseline with no faults is ideal, our mathematical framework uses continuous distributions with zero fault probability appearing in the limit only.

of a normal distribution with parameters μ and σ. It follows that the probability of error across all paths p in P is:

$$P_{E_c}(t) = 1 - \prod_{p}^{P} F(t - w_p, g_p, \sigma_g)$$

This function produces the step we expect for $\sigma_{path\_delay} = 0$ (no variation) at $t = 3.9$ shown in Figure 2(b), and Figure 2(c) shows the resulting graph for $\sigma_{path\_delay} = 0.05\mu$. To get our final P_E, we average across all cycles to derive the following equation, which is plotted in Figure 2(d) for $\sigma_{path\_delay} = 0$ and $\sigma_{path\_delay} = 0.05\mu$:

$$P_E(t) = \frac{\sum_{c=1}^{C} P_{E_c}(t)}{C}$$

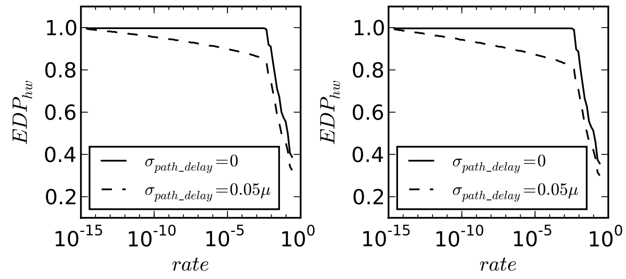
Step 2: The derivation of $convert$. Clock period reduction corresponds directly to frequency improvement, or it can be traded off to relax timing constraints on the hardware and allow V_{dd} reduction as well. For simplicity, we choose to convert all clock reduction to either frequency improvement or V_{dd} reduction. Optimizing for both frequency and voltage adds a level of complexity that we wish to avoid, while our experiments show that it generally has minimal impact for modern process technologies.

For conversion to frequency increase, recall from Equation 1 that $EDP = power \times delay^2$. Frequency increase translates to the inverse of clock period reduction. Considering $dynamic\ power = CV^2f$, frequency is linearly proportional to $power$. However, frequency increase also linearly decreases $delay$. Since $delay$ is squared, it follows that EDP decrease is linearly proportional to the frequency increase. Hence, reduction in EDP simply corresponds directly to reduction in clock period.

For conversion to V_{dd} reduction, we compute a mapping between clock period and V_{dd} by integrating over quadratically interpolated k_{volt}-factors – approximately the derivative of gate delay with respect to V_{dd} – from the Synopsis 90nm cell library. Empirically, our generated numbers are similar to voltage scaling numbers reported for the Pentium M processor [7]. Considering $dynamic\ power = CV^2f$ and $EDP = power \times delay^2$, with $delay$ held constant, EDP decreases with the square of the decrease in voltage.

Step 3: The derivation of EDP_{hw}. Figure 3 shows the results of applying Equation 2 to compute EDP_{hw}. We use $convert$ for frequency scaling (Figure 3(a)) and voltage scaling (Figure 3(b)) and use the same P_E data as in Figure 2. The resulting function maps error rate to energy efficiency improvement.

Recall that the only input parameters are a representative path delay distribution and the variation in path delay caused by process variation, represented by the input parameter σ_{path_delay}. In the following section, we derive σ_{path_delay} for different CMOS design styles and technology process nodes based on gate-delay equations.



(a) Frequency scaling (b) Voltage scaling

Figure 3. EDP_{hw} curves using $convert$ for frequency scaling (a) and voltage scaling (b).

4. CMOS design styles

In this section, we describe three different CMOS design styles that cover a spectrum of energy efficiency targets. We consider high performance CMOS, low power CMOS, and ultra-low power CMOS design. Using technology projections from the ITRS roadmap, we derive σ_{path_delay}, the hardware-specific input to our hardware efficiency model. For high performance and low power CMOS, we determine this parameter for both 45nm and 11nm technology. For ultra-low power, we consider only 45nm technology.

Normal distributions approximate the impact of process variation due to *systematic* effects (from lithographic aberrations) and *random* effects (from dopant fluctuations) [14]. We use the VARIUS model [14] and apply this observation to derive path delay variations due to systematic effects, σ_{path_sys}, separately from path delay variations due to random effects, σ_{path_rand}. Their combined effect, σ_{path_delay}, is also normally distributed, and hence $\sigma_{path\_delay} = \sqrt{\sigma^2_{path\_sys} + \sigma^2_{path\_rand}}$.

The derivation of σ_{path_rand} and σ_{path_sys}. For each design style, our approach to deriving σ_{path_rand} and σ_{path_sys} is as follows.

First, we obtain technology-specific values for V_{dd} from the 2008 ITRS roadmap [1]. We also obtain the normal distribution parameters for V_{th} variations due to process variations, $\mu_{V_{th}}$ and $\sigma_{V_{th}}$ from the ITRS and Rabaey [13]. The ITRS roadmap provides values for $\mu_{V_{th}}$, and for $\sigma_{V_{th}}$ we use data from Rabaey that show $\sigma_{V_{th}} = 32$mV at 45nm technology. Although trends are upward sloping, in the absence of 11nm data for exact variation in V_{th} we conservatively apply this same value for 11nm technology.

Second, we present a function to map variations in V_{th} to variations in the gate delay of individual transistors. As suggested in the VARIUS model, we assume random effects and system effects contribute equally to overall V_{th} variation, such that their individual effects are normally distributed with $\sigma = \sigma_{V_{th}}/\sqrt{2}$. Hence, we compute gate delay variations due to random effects, σ_{gate_rand}, and systematic effects, σ_{gate_sys}, such that they are equal to each other.

Third, we derive σ_{path_rand} from σ_{gate_rand}. We model a path as n FO4 gates and the delay variation over the path as the sum of the individual gate variations. We assume n = 12, which models an aggressively pipelined design [18]. Since the gate delay distributions are normally distributed, the sum is normally distributed as well with $\sigma_{path\_rand} = \sqrt{n} \times \sigma_{gate\_rand}$.

Fourth, we derive σ_{path_sys} from σ_{gate_sys}. We assume the same four-core chip layout as VARIUS and the recommended range value of $\phi = 0.5$ to model the correlation effects of systematic variations. In the VARIUS paper, the authors find that variations across pipeline stages are much larger than within stages. Hence, we assume a hardware design with fine-grain body biasing [19] applied to each pipeline stage to mitigate the effects. To derive σ_{path_sys} from σ_{gate_sys}, we observe that variations in path delay vary linearly with variations in gate delay, since changes in variation do not affect the correlation effects. We determine the scaling constant by using the results for intrastage variations from VARIUS, and measure it as approximately 0.114. Hence, in this paper we use $\sigma_{path\_sys} = 0.114 \times \sigma_{gate\_sys}$.

Finally, $\sigma_{path\_delay} = \sqrt{\sigma^2_{path\_sys} + \sigma^2_{path\_rand}}$ as described earlier. For each CMOS design style and technology node, the values we derive for V_{dd}, $\mu_{V_{th}}$, $\sigma_{V_{th}}$, σ_{gate_rand}, σ_{path_rand}, σ_{path_sys}, and σ_{path_delay} are shown in Table 1. The final value σ_{path_delay} is fed as input to our hardware efficiency model from Section 3.

4.1. High performance CMOS

For our high performance CMOS design style, we assume a high performance hardware design using low V_{th} transistors for maximum performance under a high power budget. For the purposes of this study, we ignore the effect of increased leakage power with low V_{th} transistors.

45nm. The ITRS roadmap presents $\mu_{V_{th}} = 103$mV and $V_{dd} = 1.0$V for high performance transistors at the 45nm technology node. From Rabaey we have $\sigma_{V_{th}} = 32$mV, which is $\sigma_{V_{th}} = 0.311\mu$ relative to the mean. To convert variation in threshold voltage to variation in gate delay, we consider the equation for delay of an inverter at normal V_{dd}:

$$gate\_delay \propto \frac{V_{dd}(1 + \frac{V_{th}}{\mu_{V_{th}}})}{(V_{dd} - V_{th})^{\alpha}} \quad (3)$$

We find that the relationship is close to linear with respect to V_{th} for typical $\alpha = 1.3$. Assuming a linear relationship and with V_{th} normally distributed, gate delay is also normally distributed. Using Equation 3, we find $\sigma_{gate\_rand} = \sigma_{gate\_sys} = 0.149\mu$. We compute path delay variations due to random effects $\sigma_{path\_rand} = \sqrt{12} \times \sigma_{gate\_rand} \approx 0.043\mu$. For systematic effects, we compute $\sigma_{path\_sys} \approx 0.017\mu$. Finally, the systematic and random effects combined yield $\sigma_{path\_delay} \approx 0.046\mu$.

11nm. Scaling from 45nm to 11nm impacts high performance transistors by reducing V_{dd} to 0.65V, while V_{th} remains relatively constant, According to ITRS scaling projections, $\mu_{V_{th}} = 118$mV. With $\sigma_{V_{th}} = 32$mV, $\sigma_{V_{th}} = 0.271\mu$, which is slightly *lower* than at 45nm. However, since the difference between V_{dd} and V_{th} reduces at 11nm, it follows from Equation 3 that the impact of $\sigma_{V_{th}}$ on gate delay is greater than with 45nm. Applying Equation 3, $\sigma_{gate\_rand} = \sigma_{gate\_sys} = 0.163\mu$. It follows that $\sigma_{path\_rand} \approx 0.047\mu$ and $\sigma_{path\_sys} \approx 0.019\mu$. The combined effect is $\sigma_{path\_delay} \approx 0.051\mu$.

4.2. Low power CMOS

Our low power CMOS design style assumes typical high V_{th} transistors that consume low power.

45nm. For low power transistors at 45nm technology, the ITRS roadmap shows mean threshold voltage $\mu_{V_{th}} = 535$mV and $V_{dd} = 1.0$V. From Rabaey, $\sigma_{V_{th}} = 32\text{mV} = 0.060\mu$, We use the same equation as with high performance CMOS to convert V_{th} variation to gate delay variation, and find $\sigma_{gate\_rand} = \sigma_{gate\_sys} = 0.092\mu$. With these values we compute $\sigma_{path\_rand} \approx 0.027\mu$ and $\sigma_{path\_sys} \approx 0.011\mu$. The combined effect is $\sigma_{path\_delay} \approx 0.029\mu$.

11nm. Technology scaling impacts low power CMOS designs by reducing both supply voltage and threshold voltage. From the ITRS roadmap, $\mu_{V_{th}} = 376$mV and $V_{dd} = 0.7$V at 11nm. From Rabaey, $\sigma_{V_{th}} = 32\text{mV} = 0.085\mu$. Applying Equation 3, $\sigma_{gate\_rand} = \sigma_{gate\_sys} = 0.136\mu$. From there, we derive $\sigma_{path\_rand} \approx 0.039\mu$ and $\sigma_{path\_sys} \approx 0.016\mu$. The combined effect is $\sigma_{path\_delay} \approx 0.042\mu$.

4.3. Ultra-low power CMOS

As an extreme design point of energy efficiency, we consider sub-threshold operation. In sub-threshold operation, the operating voltage V_{dd} is lower than V_{th}, which minimizes power and energy. However, the devices themselves are very slow. In theory successful operation is possible as long as V_{dd} exceeds the thermal voltage value, ϕ_t. In our models, we consider low power transistors from the ITRS roadmap for ultra low power CMOS.

45nm. The delay equations for sub-threshold gates are different than for high performance and low power CMOS. The delay of a characteristic inverter at sub-threshold V_{dd} is:

$$gate\_delay = \frac{CV_{dd}}{I_s e^{\frac{V_{dd} - V_{th}}{n\phi_t}}}$$

In this equation, n is the sub-threshold slope factor for ϕ_t, the thermal voltage. I_s is a device-dependent current parameter and C is capacitance. Details on sub-threshold operation are covered in the literature [21]. Holding V_{dd} constant and eliminating constants, the modeling equation of interest is:

Hardware Design Style	**Node**	V_{dd}	$\mu_{V_{th}}$	$\sigma_{V_{th}}$	σ_{gate_rand} & σ_{gate_sys}	σ_{path_rand}	σ_{path_sys}	σ_{path_delay}
High performance CMOS	45nm	1.0V	103mV	0.311μ	0.149μ	0.043μ	0.017μ	0.046μ
	11nm	0.65V	118mV	0.271μ	0.163μ	0.047μ	0.019μ	0.051μ
Low power CMOS	45nm	1.0V	535mV	0.060μ	0.092μ	0.027μ	0.011μ	0.029μ
	11nm	0.70V	376mV	0.085μ	0.136μ	0.039μ	0.016μ	0.042μ
Ultra-low power CMOS	45nm	0.25V	535mV	0.060μ	0.633μ	0.183μ	0.072μ	0.196μ

Table 1. Values for σ_{path_delay} (final column) and the values used to derive it.

$$gate\_delay \propto e^{\frac{-V_{th}}{n\phi_t}}$$

Since gate delay is exponentially related to the normally distributed V_{th}, gate delay is *log-normally* distributed, with:

$$\sigma_{gate\_delay} = \sqrt{(e^{\sigma^2} - 1)e^{2\mu+\sigma^2}},\ where\ \sigma = \frac{\sigma_{V_{th}}}{n\phi_t} \quad (4)$$

The thermal voltage ϕ_t is 26mV, and $n = 1.5$ is a typical operating point. Again assuming $\sigma_{V_{th}} = 32$mV, with equal contribution by random and systematic effects, we derive $\sigma_{gate\_rand} = \sigma_{gate\_sys} = 0.633\mu$.

To find σ_{path_rand}, recall that the delay distribution for the whole path is the sum of the individual gate delay distributions. With high performance and low power CMOS these distributions were normal distributions. However, for ultra-low power (sub-threshold) CMOS, gate delay is log-normally distributed. We apply the central limit theorem of probability theory, which states that the sum of a sufficiently large number of independent random variables can be approximated by a normal distribution. With $n = 12$ the approximation is reasonably strong. Applying the central limit theorem and using $\sigma_{path\_rand} = \sqrt{n} \times \sigma_{gate\_rand}$ as before, we derive $\sigma_{path\_rand} \approx 0.183\mu$.

We derive σ_{path_sys} from σ_{gate_sys} in the same manner as with high performance and low power CMOS and compute $\sigma_{path\_sys} \approx 0.072\mu$. For simplicity, we combine σ_{path_rand} (from a normal distribution) and σ_{path_sys} (from a log-normal distribution), by approximating their sum as normally distributed. This approximation is largely justified by the fact that σ_{path_rand} is much larger than σ_{path_sys}. The combined effect is $\sigma_{path\_delay} \approx 0.196\mu$.

11nm. According to the ITRS roadmap, V_{th} changes by approximately 200mV from 45nm through 11nm. However, with Equation 4 the value of σ_{gate_delay} is relative to the absolute value of $\sigma_{V_{th}}$ and not $\mu_{V_{th}}$. Furthermore, σ_{gate_delay} is independent of V_{dd}. With the absolute $\sigma_{V_{th}}$ predicted to remain constant from 45nm to 11nm, technology scaling has essentially no impact on the effects of process variations for sub-threshold operation, and hence, our hardware energy efficiency calculations do not change. For this reason, we do not present analysis of the 11nm node for the ultra-low power CMOS design.

5. System recovery efficiency model

In this section, we describe a mathematical model for determining the execution time overheads of recovery for systems that use speculative execution and allow errors to occur. The model is specific to backward error recovery, which is the most commonly proposed approach to recovery [17]. Independent of the underlying hardware implementation, two high-level parameters can be used to characterize any backward error recovery system: the number of cycles between checkpoints ($cycles$) and the number of cycles to restore the most recent checkpoint ($restore$). With these two parameters, it is possible to probabilistically derive the execution time overhead for a given recovery system at a specific error rate. While we present this model for use with timing speculation, the model can be generalized for other uses and is not exclusive to timing errors. The model does not require any simulation to derive its inputs.

The two inputs to the model are $cycles$ and $restore$. Let $cycles$ denote the execution time in cycles between checkpoints and let $restore$ denote the cost in cycles of restoring the checkpoint and initiating re-execution. To derive an equation for the overhead of recovery, we define two functions: let $failures$ denote the number of failed attempts to execute over a checkpoint (*i.e.* the next checkpoint was not reached), and let $waste$ denote the number of wasted execution cycles that must be discarded when an error occurs. Both functions take as input an error rate, $rate$. With these two functions, the overhead in cycles of recovery is:

$$\begin{aligned} &overhead(rate) \\ &\quad = failures(rate) \times (waste(rate) + restore) \end{aligned}$$

First, we expand $failures$ as follows. Let the random variable X denote the number of cycles executed before an error occurs. X has a geometric distribution with $P(X = k) = (1 - rate)^{k-1}rate$. Finally, let p_{succ} denote the probability of a successful (error-free) execution between checkpoints. $p_{succ} = P(X > cycles) = (1 - rate)^{cycles}$. It follows that the number of attempts to execute over the checkpoint before success, denoted by the random variable Y, has a geometric distribution with $P(Y = k) = (1 - p_{succ})^{k-1}p_{succ}$ and expected number of executions $E(Y) = \frac{1}{p_{succ}}$. However, Y includes the last, successful execution, which is not included by $failures$. Hence, $failures = E(Y) - 1$:

$$failures(rate) = \frac{1}{(1 - rate)^{cycles}} - 1$$

Second, we expand $waste$ as follows. Let the random variable Z denote and the number of wasted execution cycles that must be discarded when a fault occurs. Z is distributed with $P(Z = k) = \frac{P(X=k)}{P(X<=cycles)}, k = 1, 2, \ldots, cycles$, and with expected number of cycles $E(Z) = \frac{\sum_{k=1}^{cycles} kP(X=k)}{P(X<=cycles)}$. Hence, $waste = E(Z)$:

$$waste(rate) = \frac{\sum_{k=1}^{cycles} k(1 - rate)^{k-1} rate}{1 - (1 - rate)^{cycles}}$$

This completes the derivation of $overhead$ in terms of $restore$, $rate$, and $cycles$. With $restore$ and $cycles$ both constant, $rate$ is the only remaining free variable and, hence, $rate$ can be used to directly compute $overhead$. The relative execution time in cycles with recovery versus execution without errors is expressed as:

$$exec\_time(rate) = \frac{cycles + overhead(rate)}{cycles} \quad (5)$$

6. Recovery systems

In this section, we describe three error-recovery systems. We cover fine-grained checkpointing at the instruction granularity to coarse-grained checkpointing at the granularity of a thousand instructions. For each system, we determine the values for $restore$ and $cycles$.

Razor. Razor [3] augments critical path pipeline latches with a *shadow latch* that is placed with a slight delay behind the main latch and always receives the correct value. These latches are compared every cycle to detect faults, and thus, $cycles$ is just 1. Using their proposed *counterflow pipelining* technique, the $restore$ cost is roughly 5 cycles.

Reunion. Reunion [16] uses loosely-coupled redundant execution on two cores to detect errors. We estimate $cycles$ as the sum of the *fingerprint interval* and the *comparison interval* at roughly 100 cycles. The cost of restoring checkpointed state, $restore$, we estimate at 100 cycles as well.

Paceline. Similar to Reunion, Paceline [5] uses redundant execution on paired cores for error detection. However, Paceline is designed specifically for timing speculation and uses speculative core overclocking to achieve performance gains. The cost is an additional recovery penalty in flushing the L1 caches. Using the numbers from the Paceline paper, we approximate $cycles$ at 100 and $restore$ at 1000 cycles.

Reunion and Paceline benefit from *architectural masking* of hardware faults, where the error rate of the system is less than the hardware fault rate by a factor called the *architectural vulnerability factor* (AVF) [9]. This is because, unlike Razor, Reunion and Paceline detect at instruction retirement rather than at individual pipeline stages. Hence, error in logic that does not impact a retiring instruction can be safely ignored. In our evaluation, we estimate an AVF of 0.25 for a simple in-order core, and account for it with Reunion and Paceline by effectively multiplying $rate$ by 0.25 before applying it to Equation 5.

7. Results

This section presents quantitative results derived from our models. First, we show energy efficiency results applying our hardware efficiency model from Section 3 to each of the CMOS design styles from Section 4. We discuss the quantitative implications of technology scaling by comparing the 45nm and the 11nm technology node. Second, we show execution time overheads applying our recovery model from Section 5 to each of the recovery systems described in Section 6. Finally, we combine our system recovery execution time overheads with the energy efficiency results for the three hardware design styles to determine the system-level trade-off between error rate and energy efficiency for each combination.

7.1. CMOS design styles

Figure 4 shows the curves produced applying our hardware energy efficiency equation (Equation 2) to each of the three CMOS design styles. For high performance and low power CMOS, we use $convert$ to translate clock period reduction into both frequency increase (frequency scaling; top row) and voltage reduction (voltage scaling; bottom row). For sub-threshold CMOS, we consider only frequency scaling, since V_{dd} is already very low. Error rate is shown on the x-axis and the y-axis shows EDP normalized to EDP at a nominal error rate of 3.8×10^{-16}. For high performance and low power CMOS, the figure shows curves for both 45nm and 11nm process technologies.

Effectiveness. Timing speculation provides energy efficiency improvements across all design styles. At modest error rates of 10^{-5} to 10^{-4} it provides reasonable EDP reductions of of 10% to 30%. Improvements of 40% or larger are feasible at very high error rates, although our results combining system recovery costs in Section 7.3 question the practicality of such high error rates.

In all cases, there is a sudden dip at an error rate around 4×10^{-3}. This error rate is marked by the dash-dotted gray curve. Below this error rate, all the EDP reduction is due to timing speculation for process variability alone, which corresponds directly with the Variability Factor described in Section 3. Above this rate is where the Application Factor kicks in – where some combinational logic delays are *by design* lower than the clock period.

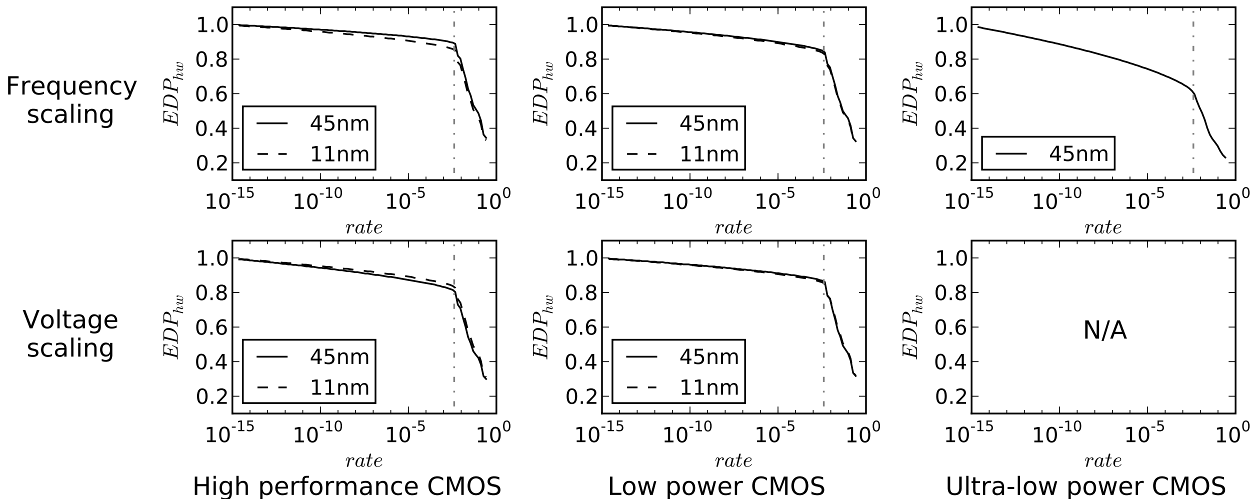


Figure 4. EDP_{hw} curves considering each design style, process technology, and scaling technique.

Sensitivity to CMOS design style. Figure 4 shows that timing speculation provides the greatest potential gains for ultra-low power CMOS, with EDP reduced by up to 40% at manageably high error rates. The figure also shows that there is essentially no difference between the EDP curves for high performance and low power CMOS.

For sub-threshold CMOS, even though gate delay variations are exponentially proportional to V_{th} variations and σ_{path_delay} is more than 4 times greater than for the other two design styles at 45nm, EDP improvement is only better by a factor of approximately 3 at low error rates. At high error rates above 4×10^{-3} the difference is even smaller. Note that we are comparing improvements from timing speculation and not the absolute energy efficiency of sub-threshold operation compared to the other two design styles.

Frequency versus voltage scaling. Figure 4 shows that the choice between frequency and voltage scaling has little impact on the relationship between error rate and EDP. Although voltage change affects EDP quadratically, the cost of trading off frequency for V_{dd} reduction increases as V_{dd} is reduced with each technology generation. At 11nm there is essentially no difference between the choice of frequency scaling or voltage scaling.

Effect of technology scaling. Figure 4 shows that, similarly to the difference between frequency and voltage scaling, the process technology has only minor impact on the relationship between error rate and EDP. While 11nm has slightly higher variability, the effect is insubstantial, and the potential to better harness variability at 11nm through timing speculation is largely undone by the diminished effectiveness of voltage scaling at the lower V_{dd}.

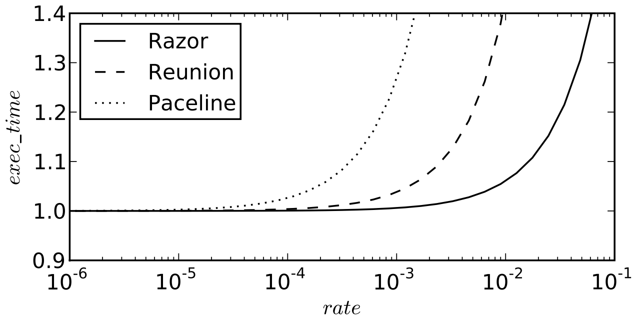


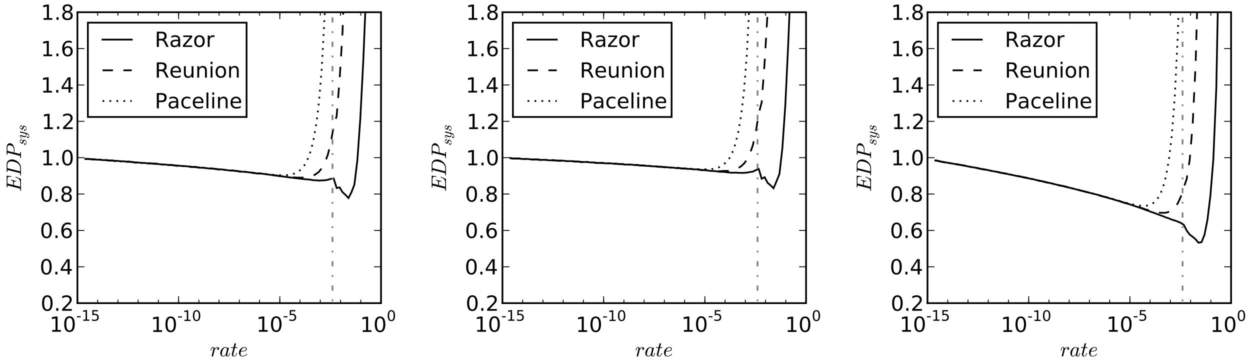
Figure 5. Equation 5 modeled using parameters for Razor, Reunion, and Paceline.

7.2. Recovery systems

Figure 5 shows the normalized execution time for each of the three recovery systems. The execution time (y-axis) is determined from the error rate (x-axis) using Equation 5. The figure shows that error rates of 10^{-5} have negligible overheads for all three techniques. For analysis, let us assume 10% overhead is the ceiling for acceptability for a system. With this assumption, fine-grained techniques that have a low $restore$ cost, like Razor, can tolerate one to two orders of magnitude higher error rates. However, if the complexities of such a technique are prohibitive, simpler systems like Reunion or Paceline provide acceptable performance at error rates of up to 10^{-4}.

7.3. Hardware and recovery combined

We now discuss overall system energy efficiency by applying the energy efficiency gains of different design styles to the recovery systems we consider. We combine the energy efficiency (EDP_{hw}) from Equation 2 with the execution time overhead ($exec\_time$) from Equation 5 using the equation $EDP = power \cdot delay^2$. EDP_{hw} is proportional



High performance CMOS   Low power CMOS   Ultra-low power CMOS

Figure 6. The curves from Figure 4 and Figure 5 combined to compute overall EDP.

to overall EDP and $exec\_time$ is proportional to the delay of the system. Hence, the EDP of the whole system is $EDP_{sys}(rate) = EDP_{hw}(rate) \cdot exec\_time(rate)^2$.

Figure 6 shows graphs for each recovery system matched with each CMOS design style. For the CMOS design styles, we show only one technology point since the results from Section 7.1 showed that, for each design style, the curves were very similar irrespective of process technology and whether frequency or voltage scaling was applied. Hence, we show only results for 45nm using frequency scaling.

Figure 6 shows that timing speculation provides, at best, a 23% reduction in EDP_{sys} for high performance and low power CMOS and a 47% reduction in EDP_{sys} for ultra-low power CMOS operating at sub-threshold V_{dd}. In all cases, the optimal error rate is around 2×10^{-2} errors per cycle. This high error rate can only be sustained with a fine-grained system like Razor. With checkpoint-based recovery, optimal error rates are around 10^{-4} with at most 13% energy efficiency improvement for high performance and low power CMOS, and 32% energy efficiency improvement for ultra-low power CMOS. Our conclusion is that large energy efficiency gains are only possible for extreme low-power designs, such as sub-threshold CMOS, irrespective of future technology process generation.

8. Limitations

While our model is end-to-end in covering hardware designs and system organization, it has limitations because it abstracts away some details.

Hardware efficiency model. First, to avoid complexities of considering area, which is required for leakage energy, we consider only dynamic energy in our hardware efficiency model. At today's technologies, leakage power presents an optimization problem in the choice of using low-leakage and high-leakage gates. The choice gives the designer freedom to determine at design time what fraction of total power is leakage power. Hence, our results for dynamic energy can be adapted assuming leakage power can be fixed to a certain percentage of total power. Second, our quantitative results are driven by path distributions from only one processor design. We believe this distribution is typical, but even so our model is easily applied to other designs as the path distribution is an input to the model.

System recovery model. First, we assume that the cost of checkpoint restoration is fixed and that execution of an application region does not perturb the microarchitecture sufficiently to change the number of cycles to re-execute the region. Second, for simplicity, we assumed execution time is linearly proportional to frequency, which is an optimistic assumption because of fixed memory delays. Third, as with other timing speculation proposals, we assume that detection coverage is perfect, detection latencies are short, and the recovery itself does not fail. Relaxing these assumptions may reduce the potential gains of timing speculation.

Combining the models. When determining optimal error rates and overall system energy efficiency, we assume all errors are detectable at all frequencies and error rates. However, certain systems like Razor place a bound on the range of timing speculation, and below a certain clock period they no longer work. Hence, our results present an upper-bound on timing-speculation improvements.

Other variations. In addition to process variation, there are other sources of dynamic variation such as voltage and temperature variation. The sources of voltage variations include voltage regulator variations, IR drops along supply rails, and di/dt noise. These effects could also be modelled as distributions affecting path delay. Our framework can be easily extended if the distributions are known.

9. Related work

Patel proposes the Critical Operating Point hypothesis for large CMOS circuits [12]. The hypothesis states that the ability to trade-off reliability for energy efficiency is extremely limited for high-performance processors. Our model confirms the hypothesis for a simple processor model and shows that recovery overheads make very high error rates impractical.

Regarding models for PVT variations, individual circuit-level models have been previously proposed. Among them, Mukhopadhyay et al. model the failure probabilities of SRAM cells due to process-parameter variations [10], and Memik et al. develop a model for error probability in a register files at a given clock frequency [8]. For timing speculation specifically, circuit and architecture techniques have been proposed to achieve additional gains [4, 15, 20]. The models developed in this paper have been extended for use with the Relax framework for software recovery as well [2].

Finally, a variety of work exists in producing measurements for different technologies for use by models such as ours. Pang and Nikolic measure and analyze process variability on a 45nm test chip [11]. A detailed characterizations of the Intel 45nm High-K/metal-gate process is also presented by Kuhn et al. [6].

10. Conclusion

Due to fundamental device properties, energy efficiency from CMOS scaling is showing diminishing or practically no improvements. Timing speculation provides the opportunity to speculatively ignore worst-case circuit timing conditions and optimize systems for common-case behavior, thus providing energy efficient execution.

In this paper, we built a hardware-level model to capture hardware-level efficiencies extracted from timing speculation. We also built a general system-level model for backward error recovery. The models were combined to yield an end-to-end model for timing speculation. The model enables a high-level analysis of timing speculation without the need for detailed architectural simulation, and we used the model to explore technology projections for CMOS scaling down to the 11nm technology node, a spectrum of CMOS design styles, and a spectrum of error recovery systems.

Our results showed that the improvements remain essentially fixed as technology scales. We also found that ultra-low power designs operating in sub-threshold region are able to obtain substantial improvements due to the exponential impact of process variations on gate delays. Finally, we found that very fine-grained recovery systems can provide significantly better energy efficiency for timing speculation than checkpoint-based recovery systems.

11. Acknowledgments

We thank the anonymous reviewers, the Vertical group, and Nam Sung Kim for comments, and the Wisconsin Condor project and UW CSL for their assistance. Support for this research was provided by NSF CAREER award #0845751 and Toshiba corporation.

References

[1] Process integration, devices, and structures. In *International Technology Roadmap for Semiconductors*. 2008 edition, 2008.

[2] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *ISCA '10*.

[3] D. Ernst et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO '03*, pages 7–18.

[4] B. Greskamp et al. Blueshift: Designing processors for timing speculation from the ground up. In *HPCA '09*, pages 213–224.

[5] B. Greskamp and J. Torrellas. Paceline: Improving single-thread performance in nanoscale CMPs through core overclocking. In *PACT '07*, pages 213–224.

[6] J. Hicks et al. 45nm Transistor Reliability. *Intel Technology Journal*, 12, 2008.

[7] Intel. Enhanced Intel speedstep technology for the Intel Pentium M processor. *White Paper*, 2004.

[8] G. Memik and A. Mallik. Engineering over-clocking: Reliability-performance trade-offs for high-performance register files. In *DSN '05*, pages 770–779.

[9] S. S. Mukherjee et al. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *MICRO '03*.

[10] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(12):1859–1880, 2005.

[11] L.-T. Pang and B. Nikolic. Measurement and analysis of variability in 45nm strained-Si CMOS technology. In *CICC '08*, pages 129–132.

[12] J. Patel. CMOS process variations: A critical operation point hypothesis. *Online Presentation*, 2008.

[13] J. Rabaey. *Low Power Design Essentials*, chapter 2. Springer, 2009.

[14] S. Sarangi et al. VARIUS: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):3–13, 2008.

[15] S. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas. EVAL: Utilizing processors with variation-induced timing errors. In *MICRO '08*, pages 423–434.

[16] J. C. Smolens et al. Reunion: Complexity-effective multicore redundancy. In *MICRO '06*, pages 223–234.

[17] D. Sorin. *Fault Tolerant Computer Architecture*. Morgan & Claypool, 2009.

[18] E. Sprangle and D. Carmean. Increasing processor performance by implementing deeper pipelines. In *ISCA '02*, pages 25–34.

[19] J. Tschanz et al. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *IEEE Journal of Solid-State Circuits*, 37(11):1396–1402, 2002.

[20] L. Wan and D. Chan. Dynatune: Circuit-level optimization for timing speculation considering dynamic path behavior. In *ICCAD '09*, pages 172–179, 2009.

[21] A. Wang, B. H. Calhoun, and A. P. Chandrakasan. *Subthreshold Design for Ultra Low-Power Systems*. Springer, 2006.