Appears in the 2nd Workshop on Architectural Reliability (WAR-2)
Orlando, Florida, December, 2006

Quantifying the Impact of Process Variability on Microprocessor Behavior

Bogdan F. Romanescu, Sule Ozev, and Daniel J. Sorin
{bfr, sule, sorin}@ee.duke.edu
Department of Electrical and Computer Engineering
Duke University

Abstract—**Architects and chip makers are worried about the impact of increasing CMOS process variability. This variability can impact a processor's performance and, depending on how aggressively the design is pushed, its reliability. We perform the first quantitative analysis of the impact of process variability on an RTL-level specification of a microprocessor core. For each pipeline stage, we compute the expected latency, as well as the standard deviation of this latency. We show that with even modest amounts of process variability, the impact on performance can be significant, and this impact can increase when using dynamic voltage scaling.**

I. Introduction

A major problem facing the semiconductor industry is the increasing amount of process variability [4, 13]. As transistor and wire dimensions continue to shrink, the variability in these dimensions—across chips and within a given chip—has a greater impact. Four parameters in particular exhibit significant variability that can greatly impact circuit behavior [19]: transistor gate length (L), gate width (W), gate oxide thickness (T_{ox}), and threshold voltage (V_{t0}). Variability in these parameters complicates system design by introducing uncertainty about how a fabricated chip will perform. Although a circuit or chip is designed to run at a nominal clock frequency, the fabricated implementation may stray far from this expected performance. There are useful transistor-level and circuit-level techniques, such as adaptive body biasing (ABB), gate sizing, Razor flip-flops [9], and X-Pipe flip-flops [23], that can help to mitigate the impact of variability, but they cannot solve the problem entirely.

For the small amounts of process variability that we have historically experienced, the simple solution has been to "speed bin" chips by how they perform. That is, we test them and then sort them into performance bins; however, binning at the chip level unfortunately reduces performance to that of the slowest path in the chip. As process variability is increasing and the number of transistors and circuit paths is also increasing, there is a greater likelihood that one or more of the paths is significantly slower than its expected delay. Consider a microprocessor pipeline stage, as a simple example. As pipeline widths have increased, the number of transistors and circuit paths per pipeline stage has grown. Combine this trend with increasing variability and it becomes even more likely that one or more circuit paths will be significantly slower than nominal and thus slow down the entire chip. Moreover, consider the trend towards placing multiple processor cores (each with many paths) on the same chip, and this issue becomes even more critical.

For computer architects, the increasing amount of process variability presents some exciting new challenges. Now, for a given chip design, there is a probability distribution of performance that is not necessarily tight around the mean (nominal) performance. Variability will cause performance and/or reliability problems, because chips will not behave as intended. As architects, we would like to develop designs that mitigate the impact of variability. To design variability-aware microarchitectures, we must first quantify the impact of low-level process variability on high-level system behavior. We believe that this problem must be quantified before it can be solved appropriately. To date, however, most architectural papers that discuss process variability have been qualitative.

Our goal in this paper is to quantify the impact of process variability on microprocessor behavior. Within the past few years, researchers have developed statistical static timing analysis (SSTA) tools that can provide a statistical analysis of a circuit's performance [1, 3, 5, 24, 25, 26]. Because no existing SSTA tool is publicly available, we developed our own SSTA tool that is an extension of previously developed models. We do not claim that our SSTA tool is a major contribution; rather, our contribution is the analysis of a microprocessor using our SSTA tool. The tool's inputs are the circuit's netlist and the expected process variability (e.g., the standard deviation of transistor gate length). Given these inputs, the tool computes the mean and standard deviation (σ) of the delay through each circuit path, and it is accurate

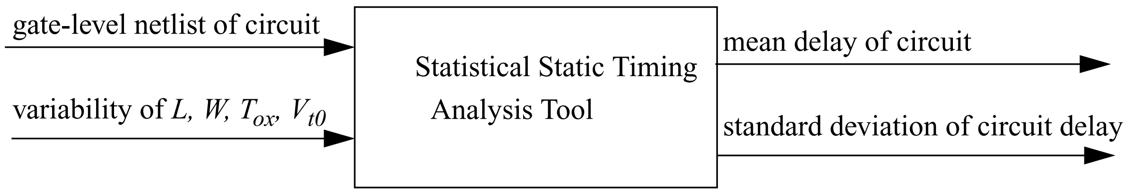
gate-level netlist of circuit
variability of L, W, T_{ox}, V_{t0}
Statistical Static Timing Analysis Tool
mean delay of circuit
standard deviation of circuit delay

Fig. 1. High-Level Overview of SSTA Tool

to within 2-3% of full Monte Carlo simulation for non-trivial circuits. We analytically combine the results from the most critical paths to determine the results for a complete circuit.

We apply our SSTA tool to an open-source microarchitecture specification, the Illinois Verilog Model [12]. IVM implements an Alpha-like core at the RTL level.

This paper makes three primary contributions:

- We provide a quantitative analysis of the impact of process variability on a microprocessor.
- We quantitatively show process variability's even greater impact on microprocessors with dynamic voltage scaling (DVS).
- We discover that even small amounts of process variability can have a significant impact on microprocessor behavior.

The rest of this paper is as follows. In Section II., we present the key features of our SSTA tool, including the validation of its accuracy. In Section III., we describe the IVM microprocessor that we are analyzing. In Section IV., we present our experimental evaluation, in which we apply our tool to the IVM microarchitecture. In Section V., we explain how variability analysis can be incorporated into the architectural design cycle. We discuss related work in Section VI., and we conclude in Section VII..

II. Statistical Timing Analysis Tool

The focus of this paper is not the SSTA tool itself, but rather to use the tool to quantify the impact of variability. However, we must provide enough information about it to justify its use and provide basic insight into how it works. We also identify its strengths and weaknesses in this section.

A high-level overview of our tool is illustrated in Figure 1. The tool's inputs are a gate-level netlist of a circuit and the means and variances[1] of the low-level process parameters (L, W, T_{ox}, V_{t0}). When we generate circuit netlists from Verilog, we constrain the netlist generator tool to only use flip-flops, 2-input NANDs, 2-input NORs, and NOT gates, and we also restrict fan-out to 10 gates. The tool's outputs are the mean delay and variance of the delay through the circuit.

A. Hierarchical Modeling

We analyze each circuit in four steps. First, as we explain in Section A.1, we identify the paths in the circuit that are most likely to be critical. Second, as we discuss in Section A.2, we divide each of these paths into small *patterns*, each of which is small enough to be easily simulated with the HSpice circuit simulator and Monte Carlo selection of input parameters. The results of a Monte Carlo pattern simulation are the mean delay and delay variance of each gate in the pattern, as well as the covariances between gates.[2] The simulation of all the patterns takes the majority of the analysis time, but it is still orders of magnitude less time-consuming than Monte Carlo analysis of the entire circuit. Our tool's runtime is linear in the number of gates on the path, whereas full Monte Carlo is polynomial. Third, as we describe in Section A.3, we analytically compose the results of the pattern simulations to compute the mean delay and variance of the entire path. This analytical step takes very little time (e.g., seconds). Fourth, as we explain in Section A.4, we analytically compose the results of each path analysis to compute the mean delay and variance of the entire circuit. This analytical step is also very fast.

A.1 Identifying Critical Paths

Like many statistical timing analysis tools, our tool analyzes each circuit path individually. We analyze the paths that are likely to be most critical. Although identifying the most critical paths is an NP-Complete problem, there are good heuristics for estimating which paths are most likely to be critical. The gate depth of a path and the fanout of the gates on the path are both directly related to path delay. We use the well-established, indus-

1. Variance is often denoted by σ^2. It is the square of the standard deviation (σ).

2. There is a non-zero covariance between the delays of gates A and B if, when A's delay varies, then B's delay also varies.

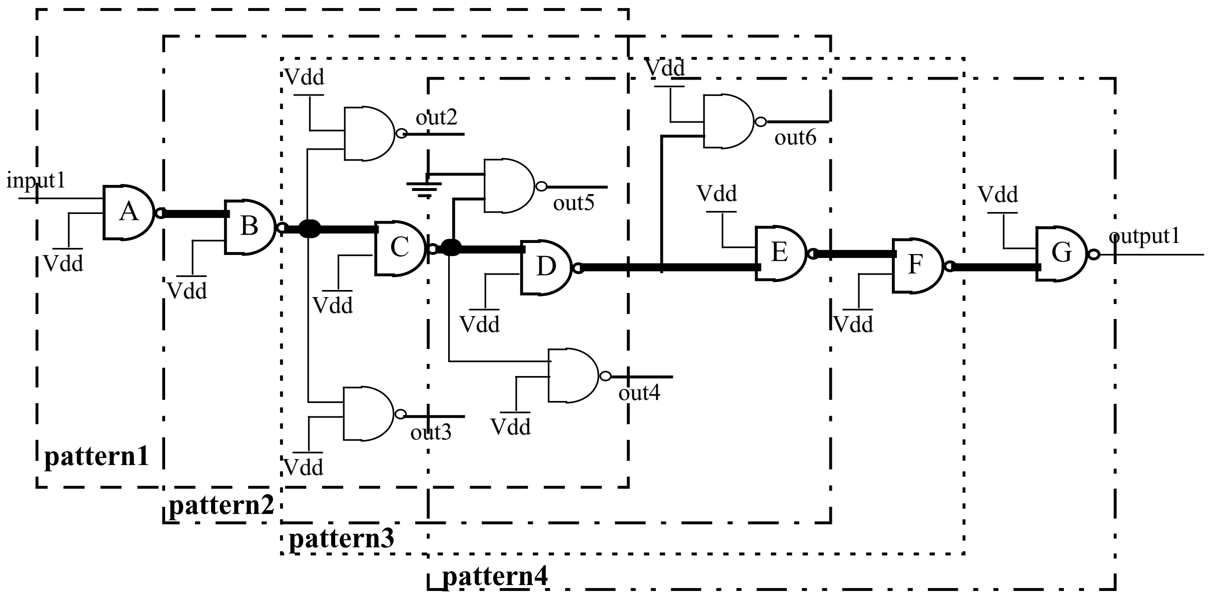

Fig. 2. Dividing a path (gates BCDEF) in a simple circuit into patterns

try-accepted heuristics of Synopsys's Design Compiler tool to identify the *K* most critical paths that it can *sensitize* (i.e., for which it can provide inputs that make the paths switch).

A.2 Pattern Construction and Simulation

In each pattern, we consider two gates on the path as the main gates for which we want to capture the mean, variance, and covariance information, and the rest of the pattern consists of their immediate load and driver gates. In Figure 2, we illustrate how we break down a simple example path, gates BCDEF, into patterns. In pattern1, gates B and C are the main gates. By only including immediate neighbor gates and none of the gates that are further upstream or downstream, we are making an approximation for the sake of computational tractability. The patterns are overlapping, which enables us to accurately model the covariances between gates (discussed next).

For each pattern, we collect the mean, variance, and covariance information for the two main gates on the path using HSpice circuit simulation with Monte Carlo selection of input parameters. We empirically determined that we needed 5,000 Monte Carlo simulations per pattern to obtain statistically significant results. After 5,000 simulations per pattern, there were negligible changes in the results. We assume low-level process variability parameters that are similar to those that have been explored in the literature [25]. Future work will explore the impact of changing the means and variances of the low-level parameters. Our process technology is 130nm, because that is the most trustworthy publicly available technology. Our parameter means and variances are in Table I, and we make the common assumption that the parameters have Gaussian distributions. For *L*, we assume a variance of 15%. Because we assume that the *absolute* variance of *W* is the same as for L (24 nm), that leads to a variance for *W* of 4.4% for the PMOS transistors and 9.6% for the NMOS transistors.

Table I. Process Parameters

parameter	**mean**	**variance**
L	160nm	15%
W (PMOS)	550 nm	4.4%
W (NMOS)	250 nm	9.6%
T_{ox}	3.3 nm	10%
V_{t0} ***(PMOS)***	-0.3169 V	10%
V_{t0} ***(NMOS)***	0.365 V	10%

A.3 Analyzing Circuit Paths

To analyze a path in a full circuit (not just a single pattern), we first look at the first two gates in the path and then at subsequent gate pairs along the path. For each pair of gates, we find the corresponding pattern (including driver and load gates) and assign the appropriate mean, variance, and covariance information to these gates. In Figure 2, for example, the path consists of gates BCDEF. In this example, we consider gate A to be the last gate in the previous stage of the circuit and gate G to be the first gate in the next stage of the circuit. Gates A and G are included to represent the limited drive capacity of the previous stage and the loading effect of the next stage, respectively. Because the parameters of A and G vary, we must consider their impact on the path.

For path BCDEF, the first gate pair we consider is BC, and we match this gate pair, along with its drivers and loads, to Pattern 1. We then shift the window by *only one* gate downstream, so the next gate pair is CD, and repeat the pattern matching process. Because we shift by only one gate, the patterns overlap and we may find somewhat different mean delay and delay variance values for a given gate in the two patterns in which it is one of the main gates (e.g., gate B in Patterns 1 and 2). To account for this, we approximate the mean delay of a gate by taking the mean of these two mean delays, and we approximate the delay variance of a gate by taking the mean of these two delay variances.

Once we have the means, variances, and covariances for every gate on the path, we can compute the mean delay and delay variance of the path.

$$mean(\text{path delay}) = \sum_{g \in \text{ gates on path}} mean(\text{delay of g})$$

$$\sigma^2(\text{path delay}) = \sum_{g \in \text{ gates on path}} \left[\sigma_g^2 + cov(g, g-1) + cov(g, g+1)\right]$$

In the variance equation, *cov(g,g-1)* is the covariance between gate *g* and the gate preceding it on the path, and *cov(g,g+1)* is the covariance between gate *g* and the gate immediately after it on the path.

A.4 Analyzing an Entire Circuit

A circuit is composed of many paths, and we use an analytical formulation to determine a full circuit's mean delay and delay variance. Since the delay of each path is a random variable, with a mean and variance, the circuit delay is a random variable that represents the maximum of the path delays. We calculate the mean and variance of the circuit delay using the analytical formulation given by Clark [7]. In this formulation, the mean and the variance of each path delay distribution, as well as the correlations among the path delays are used. We observe that, as *N* increases, the mean of the maximum path delay also increases. The standard deviation, on the other hand, *may* decrease slightly, skewing the overall distribution towards higher values. However, even with the decreased variance, the overall impact of the maximum operation is a slight increase in the mean+3σ value of the distribution, where this increase depends on the correlations among the path delays.

When analyzing a pipeline, we can use this same analytical technique to compose the delays of each pipeline stage. The maximum of the delays of each pipeline stage is a random variable that represents the clock period.

B. Limitations of the Tool

The tool makes three approximations to enable reasonable solution times for a problem that is otherwise intractable. These approximations, however, are the sources of *potential* inaccuracy in the tool's results. First, we ignore gates that are not immediate neighbors of gates on the path under analysis. We have validated this approximation for the ISCAS benchmark circuits—it introduces errors of less than 3% in estimates of mean delay and standard deviation. Second, to compute a gate's mean delay, we use the mean of its mean delays in the two patterns in which it appears. Similarly, to compute a gate's delay variance, we use the mean of its delay variance in the two patterns in which it appears. We separately validated this second assumption on the ISCAS circuits and showed that taking the mean of the values in the two patterns achieves better accuracy than using just one of the two values. Third, when we simulate each pattern, we assume that only the critical (last-arriving) input is switching. That is, we assume that the other input does not switch or that it switches much earlier than the critical input. This *single-switching assumption* at each gate can lead to conservative results (i.e., it predicts means and variances that are larger than the actual values), because the delay through a pattern is generally less when both inputs are switching.[3] However, this single-switching assumption is intuitively reasonable for the most critical path in a circuit, and our validation results for the ISCAS85 benchmark circuits confirm this intuition.

C. SSTA Tool Validation

We have validated the accuracy of our tool in estimating the mean and variance of path delay for the ISCAS85 benchmark circuits. For all of the combina-

3. For example, consider a NAND gate switching from 0 to 1. If only one input switches, there is a charging path through one PMOS transistor. If both inputs switch, two PMOS transistors become active in parallel.

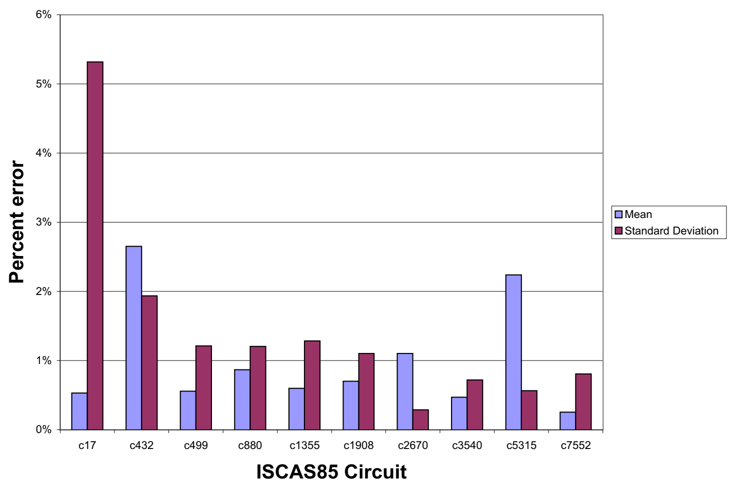

Fig. 3. SSTA Tool Validation Results (percent error with respect to full Monte Carlo Simulation)

tional circuits except c6288,[4] we tested our tool on the single most-critical "true" path (i.e., path that switches) identified by Design Compiler. Each of these most-critical paths satisfy our single-switching assumption. These paths ranged in length from 3 to 51 gates. We compared the results of our tool with the results of full Monte Carlo simulation of the entire circuit. The full Monte Carlo simulations are our reference points because they are accurate, but they took several compute-months to complete.

Our SSTA tool is extremely accurate. We show our validation results in Figure 3. Excluding for a moment the trivial circuit c17, the mean delays produced by our tool are within 3% of full Monte Carlo, and the standard deviations produced by our tool are within 2% of full Monte Carlo. Circuit c17 has a slightly higher error for standard deviation, but it only has 3 gates on its critical path and only 7 gates total; it would normally not be worth studying. The tool's accuracy is remarkable, considering how much faster our tool is than full Monte Carlo. For less critical paths that have one or more gates that violate the single-switching assumption, our tool may produce conservative results that over-estimate delay, for reasons we explained in Section B..

4. The circuit c6288 is a multiplier that is notorious for having thousands of near-critical paths for which finding switching paths requires an exhaustive search (which is impossible due to the input space). Design Compiler does not identify any true paths for this circuit that are anywhere close to the longest path in length. As such, we omit c6288 from our validation experiments.

III. Microprocessor Under Study

The microprocessor under study is the Illinois Verilog Model (IVM), which was developed and generously distributed by Prof. Sanjay Patel's research group at the University of Illinois [12]. The microarchitecture is a superscalar, dynamically scheduled core that executes a subset of the Alpha ISA. Patel's group verified the correctness of the design by comparing it to an instruction-level reference model on the SPEC benchmarks. The pipeline has 12 stages, and 132 instructions can be in-flight at any time. IVM models this microarchitecture at the RTL level, which provides sufficient detail for our study. The IVM architecture is an academic design and, as such, was not optimized by a large industrial team, so we try to be careful not to make sweeping generalizations about all modern architectures based on results obtained from experiments with IVM. We do not claim that IVM is representative of commercial microprocessors, but we believe it is a larger and more sophisticated design that has been studied before. Future work will explore commercial microarchitectures that have recently been made open-source.

We divided the Verilog code of the IVM pipeline into its 12 constituent pipeline stages. This process was largely manual, although it was not difficult since the Verilog is modular and well-structured. We describe the pipeline stages in Table II, in terms of their maximum path depth and their primary contents. For the Execute stage, we only consider the simple ALUs and the complex ALU, and we analyze them separately; we did not have time to synthesize and analyze the multiplier, branch unit, or memory units. We do not consider the Schedule or ROB stages, because their path depths are over 100 gates, which makes them unrealistic.

Table II. Characterization of IVM Pipeline Stages. The two shaded entries are unanalyzed because their path depths were over 100 gates, which made them extremely unrealistic. We split the Execute stage into two parts— simple ALUs and the complex ALU—for our analysis.

pipeline stage	**max path depth (#gates)**	**main contents**
Fetch0	75	L1 I-cache (8KB, 2-way, 8 ports), RAS (16 entries), BTB (8 entries), simple branch predictor (1-cycle), 1024-entry hybrid branch predictor (1st of 2 cycles)
Fetch1	33	hybrid branch predictor (2nd of 2 cycles)
Fetch2	34	fetch queue (32 entries)
Decode	30	decode logic (4-wide)
Rename0	67	speculative rename logic (4-wide), memory dependence predictor
Rename1	23	intra-bundle rename logic, 2nd cycle of memory dependence predictor
Schedule	n/a	scheduler (32 entries)
RegRead	20	register file (80 physical regs, 65 bits/reg, 11 read ports, 7 write ports)
Execute	31 (simple ALU) 39 (complex ALU)	2 simple ALUs, complex ALU, multiplier, branch unit, 2 address generation units, load queue, store queue
ROB	n/a	reorder buffer (64 entries, 8-wide)
ArchRATFile	28	arch. register alias table, arch. free list
NextPC	10	multiplexor for selecting the next PC

IV. Experimental Evaluation

In this section, we present our evaluation of the impact of process variability on the IVM pipeline. We discuss our methodology (Section A.) and then present results for each pipeline stage (Section B.). Last, we present a study of the impact of process variability on dynamic voltage scaling (DVS).

A. Experimental Methodology and Assumptions

We analyze each pipeline stage in isolation, to determine the impact of variability on each, because the delays of each stage are independent. For each pipeline stage, we examine the five most critical paths, as determined by Design Compiler. If there is a steep drop-off after, say the third most critical path, we ignore the fourth and fifth most critical paths, because they will never affect our timing analysis. Ideally, for each pipeline stage, we would have analyzed every path that could reasonably be critical in the presence of variability, but time constraints forced us to focus on the five paths most likely to be critical. Analyzing a subset of all paths hurts our case (i.e., makes variability seem less important), because it causes the tool to under-estimate the impact of variability. Intuitively, considering more paths leads to more potential variability.

We use the same low-level process variability parameters that we discussed in Section A. and that are listed in Table I. We make the following two assumptions to enable tractable analysis times:

Assumption #1. We assume that the 5 most-critical paths in a given pipeline stage are independent. For multiple paths within a given structure, say an ALU, this assumption is somewhat unlikely and can lead to underestimation of path delay variability.

Assumption #2. We make the same single-switching assumption that we did when analyzing circuit patterns in Section II.. Thus, our results may be conservative (over-estimate delay) for those not-most-critical paths for which the assumption does not hold for *any* input switching pattern. Without an exhaustive search of the input space, we cannot know how often this situation occurs, although we do not expect it to occur often. Unfortunately, for a circuit with N inputs, there are $2^N * 2^N = 2^{2N}$ possible input switching combinations, which is not tractable for non-trivial circuits.

B. Pipeline Stage Analyses

We first look at the delay of each pipeline stage. In Figure 4, for each pipeline stage, we plot two bars. The leftmost bar is the delay for just the single most-critical path. The rightmost bar is the delay for the maximum of the five most critical paths. The height of each bar represents the mean delay, and the “error bars” represent plus/minus three standard deviations (3σ), which is a common confidence interval for statistical analysis. The

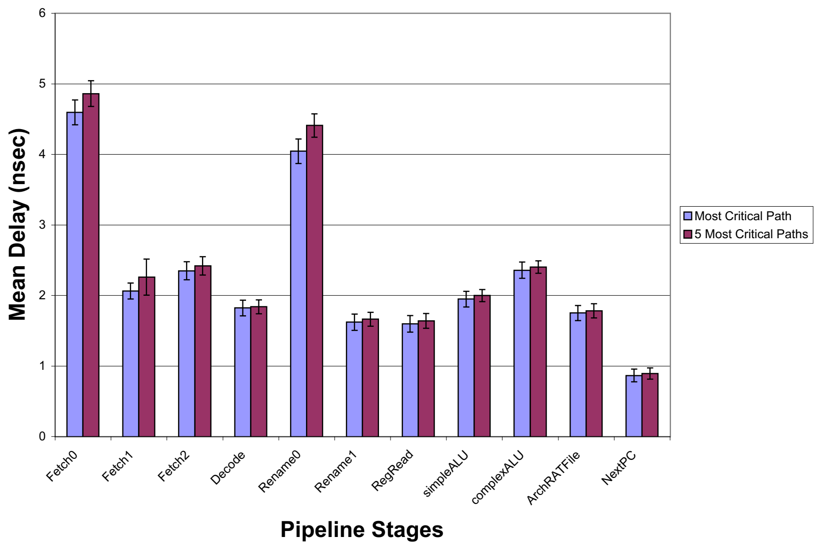

Fig. 4. Delay (mean plus/minus 3σ) of Each Pipeline Stage

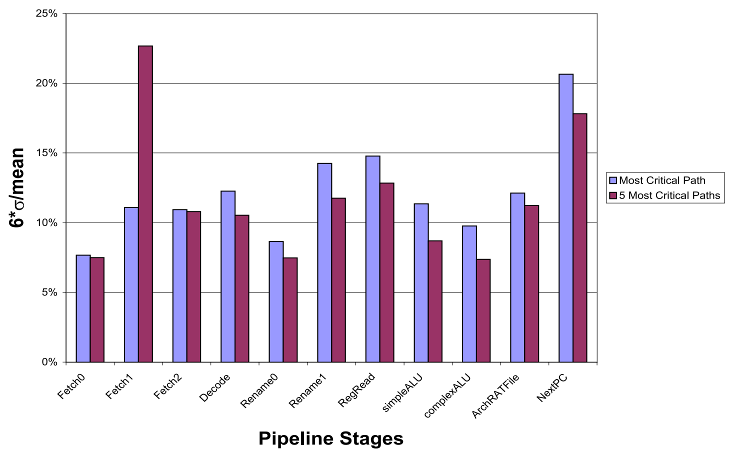

Fig. 5. Normalized Variability (6σ/mean) of Each Pipeline Stage

amount of variability is easier to discern in Figure 5. In this figure, we plot the *normalized variability* of each pipeline stage, which we define as 6σ/mean*100%. This metric, 6σ/mean, represents the 3σ confidence interval in each direction from the mean, as a fraction of the mean.[5] We make several observations and conclusions from this data.

First, before discussing variability, we note that the mean delay differs significantly across the pipeline stages. This pipeline imbalance is an artifact of IVM having been developed for studying reliability, rather than being fine-tuned for performance. Because of this unrealistic imbalance, we do not attempt to analyze the impact of variability on the pipeline as a whole.

Second, we observe from Figure 4 that architects cannot simply study the single most-critical path when analyzing a circuit in the presence of process variability. Other paths with near-critical mean delays often cause the overall pipeline stage delay to have a greater mean and/or mean plus 3σ. While the mean delay increases as we consider more than the single most-critical path, the delay variance usually decreases. Clark's formula for

5. Note that this metric is also equal to six times the *coefficient of variation*.

computing the maximum of multiple random variables implies this result for paths with equal or similar delay distributions [7]. One exception is the Fetch1 pipeline stage, which exhibits greater variance when considering multiple paths, because those paths have extremely different variances.

Third, we conclude that process variability does indeed lead to non-trivial variability in pipeline delay. The results in Figure 5 reveal 7-17% normalized variabilities. For future technologies, for which process variability is expected to become even more pronounced, this impact on microarchitectural performance variability will be even greater.

Fourth, we observe a striking correlation between path length and delay variability. For pipeline stages with longer paths, such as Fetch0 and Rename0, the normalized variability is less. Intuitively, this is because the delay variabilities of each gate along the path are more likely to cancel each other out. That is, some gates will be faster and some will be slower, and as we add more gates the normalized variability will decrease. This result has a large potential impact on microarchitecture design. As designs have trended towards deeper pipelines, each pipeline stage has become shorter. This trend, however, unfortunately amplifies the problem of process variability. Consider two processors, P1 and P2, where P1 has a deeper pipeline (i.e., more pipeline stages, with shorter paths in each stage). It is possible that the distribution of clock frequencies of fabricated chips of P1 and P2 actually favors P2. Furthermore, this result suggests that recent research to determine the optimal pipeline depth (e.g., Hrishikesh et al. [10]) might have to be re-visited to account for process variability.

C. The Impact of Variability on Dynamic Voltage Scaling (DVS)

DVS is used by a microprocessor to dynamically adapt its power consumption. By lowering the supply voltage, the power consumption is reduced. DVS is a well-known and commercially accepted mechanism, yet we suspected that it could exacerbate the impact of process variability. Lowering the supply voltage reduces the drive current. With less drive current, the sensitivity of gate delay to process parameters increases. Since there is a higher sensitivity, the variation in the gate delay will increase for the same variation in the process parameters.

To test our hypothesis, we performed an experiment on just the simple ALU from the IVM pipeline. We started with a supply voltage of 1.2V and then decreased it in steps of 0.12V down to 0.84V. In Figure 6, we plot the normalized variability (6σ/mean) of the delay of just

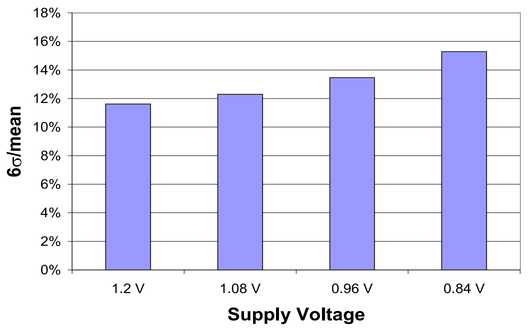

Fig. 6. Normalized Variability of Simple ALU Delay as a Function of Supply Voltage

the simple ALU (considering just the single-most critical path). The data shows that the normalized variability does indeed increase as the supply voltage decreases. The same experiment on the next most critical paths displayed similar results (not shown).

These results, albeit for just one component in a microprocessor, suggest that architects planning to use DVS in the future may need to be especially careful of the impact of process variability. Moreover, it is possible that paths that are not critical at one supply voltage setting may become critical in another. Thus, it is possible that DVS will lead to degraded performance and/or reliability, unless architects consider the impact of process variability.

V. Variability Analysis in the Design Cycle

In general, variability analysis has two primary purposes during the design cycle of a microarchitecture. First, variability analysis is an important tool for the design team. They can use this kind of analysis to identify and optimize critical paths. In the presence of process variability, designers cannot just look for the paths with the longest mean delay—they must look for the paths with the greatest statistical delay (e.g., mean delay plus three times the standard deviation). Moreover, they cannot just consider a single longest path because, depending on variability, a nominally non-critical path (i.e., a path that would not be critical if there was no variability) could be critical.

Second, variability analysis can be used to help predict expected performance and reliability across all fabricated chips. In effect, it enables us to accurately predict the distribution of chips into speed bins. This prediction is useful in at least two ways. It can obviously help predict how many chips of each speed bin will be available to be sold. It can also be used for comparing to the actual speed binning results—if the actual chips do not follow the same distribution, that discrepancy can potentially identify a fabrication problem.

VI. Related Work

There has been some work on quantitatively evaluating the impact of variability on selected parts of microprocessors, especially storage structures. Agarwal et al. [2] study the impact of variability on cache performance. Liang and Brooks [17, 18] explore the performance impact of variability on execution units and SRAM structures, such as register files. Venkatesan et al. [22] developed a DRAM placement policy that considered the impact of variability in required refresh rates. By placing data in DRAM pages that require less frequent refreshes, they save refresh power. Our work here differs in that we study the impact of variability on an entire pipeline.

Some related work has explored the impact of variability on a simplified model of a processor. Datta et al. [8] assume that the delay distribution of each pipeline stage is already known and then show how to analytically combine these stage delay distributions into an overall processor delay distribution. Humenay et al. [11] developed a model of variability's impact on a pipeline, in which they assume that each pipe stage is either all SRAM or all logic. If the stage is logic, it is modeled as an adder instead of the actual logic. Kim et al. [15] discuss the impact of process variability on the power consumption of an abstract pipeline that is modeled in terms of the number of FO4 delays per stage. Our work differs by looking at the actual circuitry in each pipeline stage, instead of abstracting it away or assuming its delay is known.

There have been many SSTA tools like ours. The most similar tools are the ones by Agarwal et al. [1], Amin et al. [3], Chang and Sapatnekar [5], Visweswariah et al. [24], Zhan et al. [25], and Zhang et al. [26]. These analytical tools consider gate-level details, and they are scalable to large benchmark circuits. They produce probability distributions for path delays, and they have been validated against Monte Carlo. Our work differs in that we have used our tool to analyze a full microprocessor. There are also some subtle differences in our SSTA methodology, but these are second-order effects, and we do not claim to innovate in the tool itself.

Some recent research has explored techniques for mitigating the impact of variability at both the architectural level [2, 17, 18, 22] and circuit level [9, 23, 14, 20, 6, 16, 21]. We do not discuss this work in detail because it is orthogonal to the issue of analysis.

VII. Conclusions

Process variability is becoming an important challenge for computer architects. We cannot just assume that circuits will perform at their nominal frequencies, because variability in process parameters will lead to variability in overall performance. Even modest amounts of process variability affect how we should design a microarchitecture.

The primary goal of this paper was to present the quantitative impact of process variability on microarchitectural performance and reliability. We believe that designing variability-aware microarchitectures is a vitally important problem for architects to consider, and we look forward to future work in this area.

Acknowledgments

This work is supported in part by the National Science Foundation under grant CCF-0444516, the National Aeronautics and Space Administration under grant NNG04GQ06G, an equipment gift from Intel Corporation, and a Warren Faculty Scholarship.

We thank Sanjay Patel and Nick Wang for their help with IVM. We thank Alvy Lebeck and Anita Lungu for helpful comments on this work. We thank Fred Bower for helping us with an experiment. We thank Hisham Massoud for consultations on semiconductor physics. We thank Jennifer Miller for her early work on this project.

References

[1] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations. In *Proceedings of IEEE ICCAD*, pages 900–907, Nov. 2003.

[2] A. Agarwal et al. A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies. *IEEE Transactions on Very Large Scale Integration Systems*, 13(1):27–38, Jan. 2005.

[3] C. Amin et al. Statistical Static Timing Analysis: How Simple Can We Get? In *Proceedings of the 42nd Design Automation Conference*, pages 652–657, June 2005.

[4] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, Nov/Dec 2005.

[5] H. Chang and S. S. Sapatnekar. Statistical Timing Analysis Considering Spatial Correlations Using a Single Pert-like Traversal. In *Proceedings of International Conference on Computer Aided Design*, pages 621–625, Nov. 2003.

[6] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester. Parametric Yield Maximization Using

Gate Sizing Based on Efficient Statistical Power and Delay Gradient Computation. In *Proceedings of the International Conference on Computer Aided Design*, Nov. 2005.

[7] C. E. Clark. The Greatest of a Finite Set of Random Variables. *Operations Research*, 9(2):145–162, Mar.-Apr. 1961.

[8] A. Datta, S. Bhunia, S. Mukhopadhyay, N. Banerjee, and K. Roy. Statistical Modeling of Pipeline Delay and Design Pipeline Under Process Variation to Enhance Yield in Sub-100nm Technologies. In *Proceedings of Design, Automation, and Test in Europe*, pages 926–931, Mar. 2005.

[9] D. Ernst et al. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2003.

[10] M. S. Hrishikesh, K. I. Farkas, N. P. Jouppi, D. Burger, S. W. Keckler, and P. Shivakumar. The Optimal Logic Depth Per Pipeline Stage is 6 to 8 FO4 Inverter Delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 14–24, May 2002.

[11] E. Humenay, D. Tarjan, W. Huang, and K. Skadron. Impact of Parameter Variations on Multicore Architectures. In *Proceedings of the Workshop on Architectural Support for Gigascale Integration*, June 2006.

[12] Illinois Advanced Computing Systems Group. Illinois Verilog Model. http://www.crhc.uiuc.edu/ACS/tools/ivm/about.html.

[13] International Technology Roadmap for Semiconductors, 2003.

[14] T. Kehl. Hardware Self-Tuning and Circuit Performance Monitoring. In *Proceedings of the International Conference on Computer Design*, Oct. 1993.

[15] N. Kim et al. Total Power-Optimal Pipelining and Parallel Processing Under Process Variations in Nanometer Technology. In *Proceedings of the International Conference on Computer Aided Design*, Nov. 2005.

[16] X. Li, J. Le, M. Celik, and L. Pileggi. Defining Statistical Sensitivity for Timing Optimization of Logic Circuits with Large-Scale Process and Environmental Variations. In *Proceedings of the International Conference on Computer Aided Design*, pages 844–851, Nov. 2005.

[17] X. Liang and D. Brooks. Latency Adaptation of Multiported Register Files to Mitigate Variations. In *Proceedings of the Workshop on Architectural Support for Gigascale Integration*, June 2006.

[18] X. Liang and D. Brooks. Mitigating the Impact of Process Variations on Processor Register Files and Execution Units. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2006.

[19] S. Nassif. Design for Variability in DSM Technologies. In *Proceedings of First International Symposium on Quality of Electronic Design*, pages 451–454, Mar. 2000.

[20] D. Patil et al. A New Method for Design of Robust Digital Circuits. In *Proceedings of the International Symposium on Quality of Electronic Design*, pages 676–681, Mar. 2005.

[21] D. Sinha, N. Shenoy, and H. Zhou. Statistical Gate Sizing for Timing Yield Optimization. In *Proceedings of the International Conference on Computer Aided Design*, pages 1037–1041, Nov. 2005.

[22] R. K. Venkatesan, S. Herr, and E. Rotenberg. Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM. In *Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture*, Feb. 2006.

[23] X. Vera, O. Unsal, and A. Gonzalez. X-Pipe: An Adaptive Resilient Microarchitecture for Parameter Variations. In *Proceedings of the Workshop on Architectural Support for Gigascale Integration*, June 2006.

[24] C. Visweswariah et al. First-Order Incremental Block-Based Statistical Timing Analysis. In *Proceedings of the 41st Design Automation Conference*, pages 331–336, June 2004.

[25] Y. Zhan, A. J. Strojwas, X. Li, and L. T. Pileggi. Correlation-Aware Statistical Timing Analysis with Non-Gaussian Delay Distributions. In *Proceedings of the 42nd Design Automation Conference*, pages 77–82, June 2005.

[26] L. Zhang et al. Correlation-Preserved Non-Gaussian Statistical Timing Analysis with Quadratic Timing Model. In *Proceedings of the 42nd Design Automation Conference*, pages 83–88, June 2005.