Statistical Fault Injection for Impact-Evaluation of Timing Errors on Application Performance

Jeremy Constantin[1], Andreas Burg[5]
TCL, École Polytech. Féd. de Lausanne
{jeremy.constantin,andreas.burg}@epfl.ch

Zheng Wang[2], Anupam Chattopadhyay[4]
Nanyang Technological University
{wangz,anupam}@ntu.edu.sg

Georgios Karakonstantis[3]
EEECS, Queen's University Belfast
g.karakonstantis@qub.ac.uk

ABSTRACT

This paper proposes a novel approach to modeling of gate level timing errors during high-level instruction set simulation. In contrast to conventional, purely random fault injection, our physically motivated approach directly relates to the underlying circuit structure, hence allowing for a significantly more detailed characterization of application performance under scaled frequency / voltage (including supply noise). The model uses gate level timing statistics extracted by dynamic timing analysis from the post place & route netlist of a general-purpose processor to perform instruction-aware fault injections. We employ a 28 nm OpenRISC core as a case study, to demonstrate how statistical fault injection provides a more accurate and realistic analysis of power vs. error performance.

1. INTRODUCTION

Shrinking transistor sizes and increased static and dynamic parametric variations, as well as the need to reduce pessimistic design margins renders circuits more prone to timing errors, caused for example by supply voltage noise. Such timing errors may permeate various parts of the micro-architecture, propagate to the system software layer and eventually lead to catastrophic program failure. The severe impact of such errors on system functionality has led to new design paradigms that either try to predict the potential errors and apply voltage/timing guardbands at design time or try to detect the incurred timing errors at run time and take corrective actions at the micro-architecture level [1, 2].

To circumvent the large power and performance penalties of such approaches, the approximate computing paradigm has emerged as an alternative, where output-quality is traded-off against power by exploiting the error-resilient nature of various applications [3]. However, the efficiency of this design approach largely depends on the identification of the real impact of timing errors on system operation, which is usually evaluated through models and system simulators at design time. Inaccurate prediction/simulation of the errors at design time may not only lead to the design of inefficient techniques that waste resources and power, but may also lead to complete failure, if the impact of errors is underestimated. Therefore, an essential step in coping with variations and the resulting timing errors is the development of accurate characterization approaches that consider the statistical nature and real impact of such errors at the micro-architecture level.

Several high-level timing models and simulators exist for injecting errors and studying their impact on system performance [4–7]. Although emulation of a faulty environment at the gate level or at real hardware may be more accurate in capturing the impact of faults, such approaches are not widely used due to the prohibitively long simulation time and high setup cost [8]. Instead, fault injection (FI) at the micro-architecture level by flipping register bits in a cycle-accurate simulator or at the software layer by altering memory states have prevailed, due to the reduced simulation time [9]. However, such approaches suffer from low accuracy since errors at various registers are either injected randomly without any view of the actual gate level implementation or timing [10–13] making the fault injection further unrealistic. A compromise between speed and accuracy lies at modeling the gate level timing behavior of the underlying circuits carefully in an instruction set simulator.

Contributions: In this paper, we propose a novel approach to modeling of gate level timing errors during high-level instruction set simulation based on accurate characterization of the statistical nature of the timing of an open-source processor. Our approach involves the following contributions:

- In contrast to conventional, almost purely random fault injection, the proposed approach directly relates to the underlying circuit, since characterization of timing errors is performed at gate level on a post place & route netlist.
- The characterization accuracy of timing errors is improved by conditioning the error statistics on the instruction type using gate level dynamic timing analysis (DTA).
- The initially fixed characterization of the DTA for different operating conditions is extended to also model the dynamic impact of (high frequency) supply voltage noise, which is one of the most critical timing uncertainties since it is difficult to compensate for with more conventional process compensation techniques.
- The characterized statistical instruction-based timing behavior of the underlying processor is used to inject faults in a cycle-accurate simulator. This allows accurate evaluation of the impact of faults at the application layer. The impact is quantified in terms of output quality, as well as energy and performance.
- The proposed approach is applied to a 32-bit 6-stage OpenRISC core in 28nm CMOS and the impact of timing errors on various application kernels with different characteristics (computation, control) in terms of output quality and point of first failure (PoFF), is assessed.

Overall, the proposed approach does not only provide an alternative and accurate approach for rapidly evaluating the impact of errors on system performance, but can also prove as an essential tool to identify and mitigate reliability bottlenecks in hardware implementations (e.g., by pointing out structures that lead to timing walls that cause frequent fatal errors) as well as to determine the timing margins required to achieve a desired quality metric.

The rest of the paper is organized as follows. Section 2 describes the case study and benchmarks used in the paper. Section 3 discusses different models for timing errors, including our novel statistical approach, while Section 4 analyses the power, performance and output quality results obtained by our proposed model. Conclusions are drawn in Section 5.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

DAC '16, June 05 - 09, 2016, Austin, TX, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4236-0/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2897937.2898095

2. SETUP AND CASE STUDY

Before describing in detail our modeling approach, in the following we discuss the hardware and software environment of our case study for evaluation of the performance of various benchmarks.

2.1 Hardware Processor Core

The hardware is comprised of a modified 32-bit OpenRISC general purpose embedded processor core, which includes instruction and data memories. The micro-architecture of the core has a 6-stage pipeline and achieves close to one instructions per cycle, including single-cycle 32-bit multiplications. Both memories are realized in form of single-cycle latency SRAM macros.

The core is implemented with the constraint strategy proposed by the authors of [14], which avoids a timing wall in the path delay distribution of the circuit to ensure that important control paths are not immediately affected by frequency-over-scaling. For such a core this can be done with limited area and power overhead ($\approx$ 5–13%) [14], enabling a graceful performance degradation beyond the static timing analysis (STA) limit. In our implementation, this optimization ensures that only the ALU endpoints of the execution stage data path limit the maximum clock frequency (here 707 MHz at 0.7 V), while the paths in all other stages are short enough to be safe when operating below a certain much higher threshold frequency (here 1.15 GHz at 0.7 V). Hence, for this case study, we can limit our modeling to timing errors that can be induced in these 32 ALU-endpoint flip-flops, and assume the use of meta-stable hardened flip-flops for this specific pipeline register , and use the fact that non-ALU instructions (e.g., branch, load, store, etc.) are always safe from such errors (below the given threshold).

For the timing characterization required by our proposed model we use the dynamic timing analysis proposed in [14] and apply it to a fully placed and routed test-chip design of the processor, which has been fabricated in a 28 nm FD-SOI CMOS technology.

2.2 Instruction Set Simulator with FI

Simulation is performed using a cycle-accurate instruction set simulator (ISS) that is generated from a custom LISA-model of our OpenRISC implementation. The ISS is enhanced by the FI framework developed by the authors of [15], which allows the injection of faults on the level of the micro-architecture (e.g., into pipeline registers).

While a benchmark is executed, FI is only performed for the kernel part of that benchmark (typically accounting for 99%+ of the runtime cycles), which allows us to analyse the effects on the characteristic parts of the code of the application. Moreover, since FIs can frequently cause wrong branching behavior, we include a basic infinite loop detection in the ISS to abort the execution in case of obvious fatal errors.

2.3 Software Benchmarks

We characterize the application performance for four widely used kernels. As the benchmark properties in Table 1 show, some kernels are more computation (data path) heavy, while others are more control oriented. The performance metrics reported in this paper are assessed by Monte Carlo simulation with at least 100 simulations per parameter configuration (data point).

3. MODELING OF TIMING ERRORS

Modeling of timing errors in high-level instruction set simulators can be performed with different levels of detail to match the underlying circuit-level implementation.

Table 1: Overview of benchmark properties

bench-mark	**median**	**matrix mult. (8- & 16-bit)**	**k-means clustering**	**Dijkstra**
type	sorting	arithmetic	data mining	graph search
compute	-	++	+	-
control	+	-	+	++
size	129 values	16x16 matr.	8 points (2D)	10 nodes
cycles	216 k	60 k	351 k	984 k
output error	relative difference	mean squared error (MSE)	cluster membership	mismatch in min. distance

Table 2 provides an overview of different modeling approaches, and their features. We begin our analysis with the widespread purely random fault injection model A, to show its limitations. Next, model B follows more closely the actual circuit behavior by considering the results of the static timing analysis of the circuit under a given set of operating conditions. We then further refine this model by considering the impact of supply voltage noise on the timing behavior and we refer to this refined model B as model B+. Finally we introduce our novel model C, which further improves the accuracy of the characterization of application behavior with an even more detailed fault injection that accounts also for critical-path activation statistics conditioned on individual instruction types.

Table 2: Overview of timing error models & features

model	*fault injection technique*	*timing data*	*multi-V_{dd}*	*V_{dd} noise*	*gate-level aware*	*instruction aware*
A	fixed probability	none	no	no	no	no
B	fixed period violation	STA	yes	no	partially	no
B+	modulated period violation	STA	yes	yes	partially	no
C	probabilistic period violation (using CDFs)	DTA	yes	yes	yes	yes

3.1 Fixed Probability FI

Model A is based on the introduction of random bit flips into all or a limited subset of logical or physical registers within the processor core (and potentially the memories). Each bit flip occurs with a fixed FI probability. This simple model has originally been motivated by the analysis of single-event-upsets (SEUs) which affect all resources independently of each other and independent of the processor state or timing properties, but even for that it has been shown that accurate modeling of the underlying hardware is essential [7].

Nevertheless, this random FI is also used frequently to model the impact of variations which actually manifest in the form of timing errors. Unfortunately, this straightforward approach is obviously highly inaccurate and lacks any physical motivation: The model neglects the fact that timing errors appear selectively only on the endpoints of critical or near-critical paths and only if these paths are actually excited. Moreover, the FI rate has no direct link to the activity of the hardware or to the operating conditions. This is especially problematic, since we aim to characterize the impact of frequency-/voltage-over-scaling and supply voltage noise on the program behavior and application output error.

3.2 Static Timing Based FI

To relate the FI for individual endpoints within the processor core more closely to the underlying hardware, [15] proposes to consider the worst case path delays to each endpoint. These delays can be obtained for any operating condition that is available from the design kit through STA of

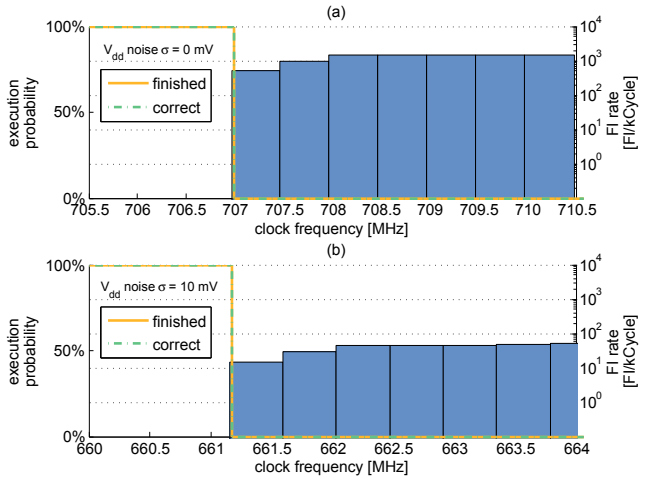

Figure 1: Performance and fault injection rate of the median benchmark for (a) model B based on STA @ $0.7\,V$, and (b) model B+ with supply voltage noise

the gate level netlist of the placed and routed design. The individual endpoints are then related to their location in the processor's execution pipeline and to the set of instructions that has an effect on this pipeline stage. A fault is always injected into a register whenever the pipeline stage is activated (e.g., only ALU instructions can trigger the FI in the execution stage which shows no activity for other instructions) and when the longest path delay to that endpoint exceeds the clock period.

The issue with model B is that it is overly pessimistic, since details of the CPU state and current or previous data values which have an influence on the path excitation are not taken into account. Furthermore, no distinction is made between potentially very different path delays to the same register, depending on the type of instruction that all trigger the same pipeline stage. Finally, factors such as high-frequency supply voltage noise are not considered. Nevertheless, such factors critically determine the behavior of a circuit on the boundary between 100% reliable operation and complete failure and are therefore essential.

To illustrate the pessimism of this model, we apply it to our case study and show the FI rates and program behavior for the median benchmark with different frequencies in Fig. 1(a). It can be seen from the plot that the FI rate immediately rises significantly as soon as the clock frequency just slightly exceeds the static timing limit, since any executed ALU instruction, independent of its type, leads to a timing error in the execution stage. As a result of this high FI rate, the probability for a program to execute correctly and even to finish drops abruptly from 100% down to 0% with almost no transition region that could be exploited. Repeated execution of the same program will not change this behavior since for a given program, there is no randomness in the model. Note that we limit the illustration here to the results of only one benchmark due to space restrictions, but we observe the same behavior also with all other benchmarks.

3.3 Supply Voltage Noise

To recover the link to uncertainties (randomness) in the underlying circuit behavior we extend model B to model B+ by accounting for the influence of supply voltage noise as a primary source of variation of gate delays and timing behavior of a specific instance of the chip. Supply voltage noise can have many sources: it is inherently caused by DC-DC converters and depends strongly on the off-chip and on-chip power delivery network and the circuit switching activity (V_{dd}-droop). In this study, we model this supply voltage noise by a normal distribution, with a mean of 0 V and a standard deviation σ, but other distributions are also possible. The maximum noise level is clipped/saturated at 2σ, to avoid the occurrence of large, physically unrealistic, spikes due to the tails of the distribution.

During each cycle in the simulation a new independent random value for the supply voltage noise is drawn and translated into a factor which modulates the timing of each path of the circuit for that cycle. The relation between a small supply voltage change and the corresponding effect on delay is extracted from a fitted V_{dd}-delay curve, which is interpolated from the delay of the worst path scaled over 5 different supply voltages (0.6 V to 1.0 V in 100 mV steps)[1]. These modified delays are then used to determine the injection of faults in the same way as for model B (Sec. 3.2).

The FI behavior under model B+ is shown for $\sigma = 10\,mV$ (maximum V_{dd} noise of $\pm 20\,mV$) in Fig. 1(b). Compared to the no-noise scenario of model B (Fig. 1(a)), the clock frequency at which first faults start to get injected is now significantly lower. The higher the noise σ, the further away from the static timing limit of 707 MHz is the first point of fault injection (at 661 MHz and 588 MHz for $\sigma = 10\,mV$ and $\sigma = 25\,mV$, respectively). However, the observed fault injection rate at the first point of fault injection is significantly lower at only around 10 faults per 1000 cycles, due to the modulated (instead of fixed) path delays, caused by the random characteristic of the supply voltage noise.

Unfortunately, we still observe the same hard threshold in the application behavior as for model B (for the shown and all other benchmarks). The reason for this behavior is that even model B+ does not capture the significant instruction and data dependencies of path delays [14].

3.4 Proposed Dynamic Timing Statistical FI

To improve the accuracy and link to the physical circuit of models B and B+ we further refine the resolution of the fault injection with respect to different instructions and introduce a statistical model C to cope with data dependent and micro-architectural or circuit implementation related delay uncertainties.

To this end, we employ dynamic timing analysis to extract the statistics of the data arrival times (i.e., the dynamic timing slack) on all relevant endpoints inside the processor, as introduced by [14]. This characterization is performed independently for different instructions, even if they affect the same pipeline stage. For our experiments, we use a gate level characterization kernel (here with 8 kCycles), covering all ALU instructions with randomized operands. We further verify that all non-ALU instructions possess a sufficient timing margin to always be non-critical. The extracted dynamic slack statistics are then used to determine the probabilities $P_{E,V,I}(f)$ of an endpoint E to have a timing error at a given frequency f with supply voltage V, while the instruction I is executed (resides in the pipeline stage associated with E). We calculate $P_{E,V,I}(f) = v_f/n_I$, where n_I is the total number of cycles in which DTA encounters instruction I, and v_f is the number of these cycles for which the dynamic path delay to E (including the setup time) is larger than the clock period $1/f$, i.e. cycles in which the endpoint timing is violated by instruction I. Sweeping f provides us with the cumulative distribution functions (CDFs) for the

[1]Although not all paths scale equally in terms of delay (especially over a wide voltage range), due to varying gate compositions of the paths, this is nevertheless a valid approximation for capturing small delay changes around an accurately characterized operating point.

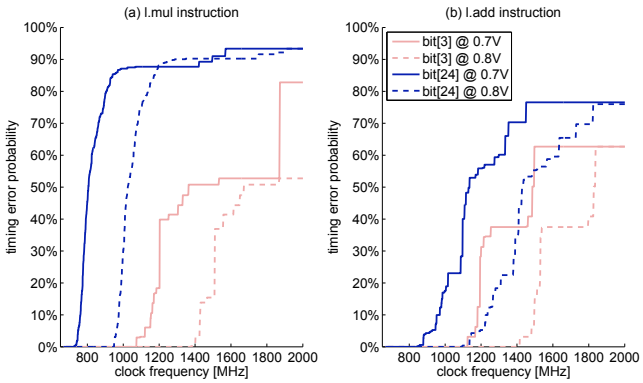

Figure 2: Cumulative distribution functions of timing error probabilities extracted by DTA, for different ALU endpoints and supply voltages

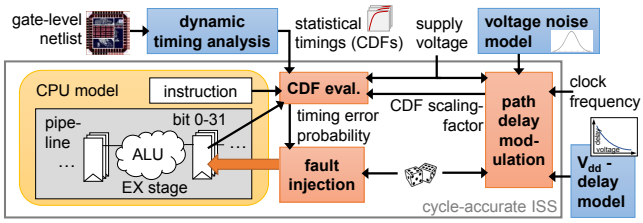

Figure 3: Simulation with statistical FI (model C)

dynamic timing error probabilities, as shown in Fig. 2 for two instructions, two endpoints, and two supply voltages.

As can be seen from Fig. 2, the multiplication instruction starts to fail at a lower frequency than the less complex addition instruction for the same supply voltage and ALU endpoint. Moreover, we observe that bits with higher significance tend to fail earlier than bits with lower significance and a higher supply voltage shifts the CDF to the right.

As illustrated in Fig. 3, the proposed model C integrates the instruction-aware statistical dynamic timing information from the DTA in form of CDFs and combines it with the supply voltage noise model presented in Section 3.3. Specifically, model C performs the following steps in each cycle of the simulation:

1. A CDF scaling-factor is derived from the defined simulator clock frequency together with the randomly distributed supply voltage noise, which allows for the dynamic adjustment of the CDFs.
2. The timing error probability $P_{E,V,I}(f)$ is determined for all the relevant endpoints E, by using the corresponding scaled CDF with matching instruction I, and supply voltage V (without noise).
3. FI is performed on each endpoint E with the respective probability $P_{E,V,I}(f)$.

Our proposed approach is also able to account for parameters that are constant or vary only slowly for a specific die, such as process variations, temperature, and aging. These effects can be modeled accurately by performing DTA on a netlist that is timed with libraries provided by the foundry that are characterized for the desired process corner, temperature, and age. Different sets of CDFs can then be used within the simulation environment to model the effects on the application.

4. APPLICATION OF STATISTICAL FI

In the following we apply the proposed statistical FI (model C) to the considered case study to illustrate how this detailed model can provide interesting insights into the operation and application behavior under operating conditions that may lead to timing errors.

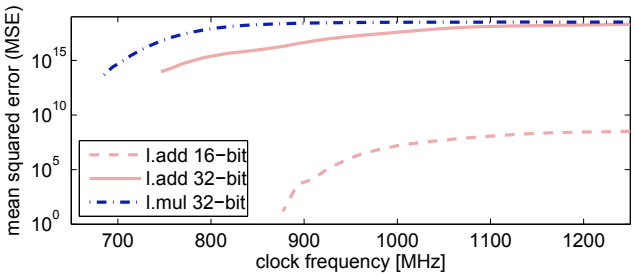

Figure 4: MSE vs. frequency for add. & mult. instructions at $V_{dd} = 0.7\,V$ with $\sigma = 10\,mV$ (model C)

4.1 Instruction Characterization

We first study the behavior of addition (l.add) and signed multiplication (l.mul) instructions across different frequencies. Addition is evaluated in two forms, using input operands with a 16-bit value range and a 16-bit result, and with 32-bit operands giving a 32-bit result. Multiplication is performed with input operands that cover a 16-bit value range and a 32-bit result. All operands are chosen uniformly random and the operating point for the error analysis is a supply voltage of 0.7 V with $\sigma = 10\,mV$ voltage noise. The mean squared error (MSE) due to timing errors is shown in Fig. 4.

The plot shows that first calculation errors (MSE > 0) occur at 877 MHz, 746 MHz, and 685 MHz for the 16-bit addition, 32-bit addition, and multiplication, respectively. This spread illustrates the relatively large difference between the points of first failure (PoFF) for different arithmetic instructions and also highlights the importance of timing error modeling on single-bit granularity, which can be seen by the significant difference in PoFF when comparing 16-bit with 32-bit addition. Moreover we observe that the MSE has moderate magnitude for low frequencies and saturates close to maximum values (corresponding to the used operand bitwidths) after about 15% of further frequency increase beyond the PoFF.

4.2 Impact of Frequency, Voltage, and Noise

A key feature of the proposed statistical fault injection model is that it captures many details of the gate level implementation that are required to study the transition region in which timing errors start to appear due to frequency-/voltage-overscaling or insufficient margins to protect against supply noise. On the application level, the impact of these effects is often characterized by four different metrics: the probability for the application to finish, the probability for the execution to be correct, the rate of injected faults (in FIs per 1000 cycles of kernel execution), and the error of the program output.

Fig. 5 shows these metrics for the median benchmark running at different operating frequencies on the hardware with two different supply voltages and three different levels of voltage noise (sub-figures (a-f)), averaged over 200 Monte-Carlo trials. The graphs only show the interesting transition region between reliable and unreliable operation, while the low-frequency ranges where no errors occurr (no faults are injected) is grayed out and marked with "n/a".

A first interesting observation is that the simulations reveal that the PoFF where the application first does not finish with a 100% correct result is displaced from the pessimistic STA limit. The corresponding possible gain from frequency-over-scaling is indicated in the sub-figures.

A second observation relates to the impact of voltage noise. From Fig. 5(a-c) and 5(d-f) we can clearly see the im-

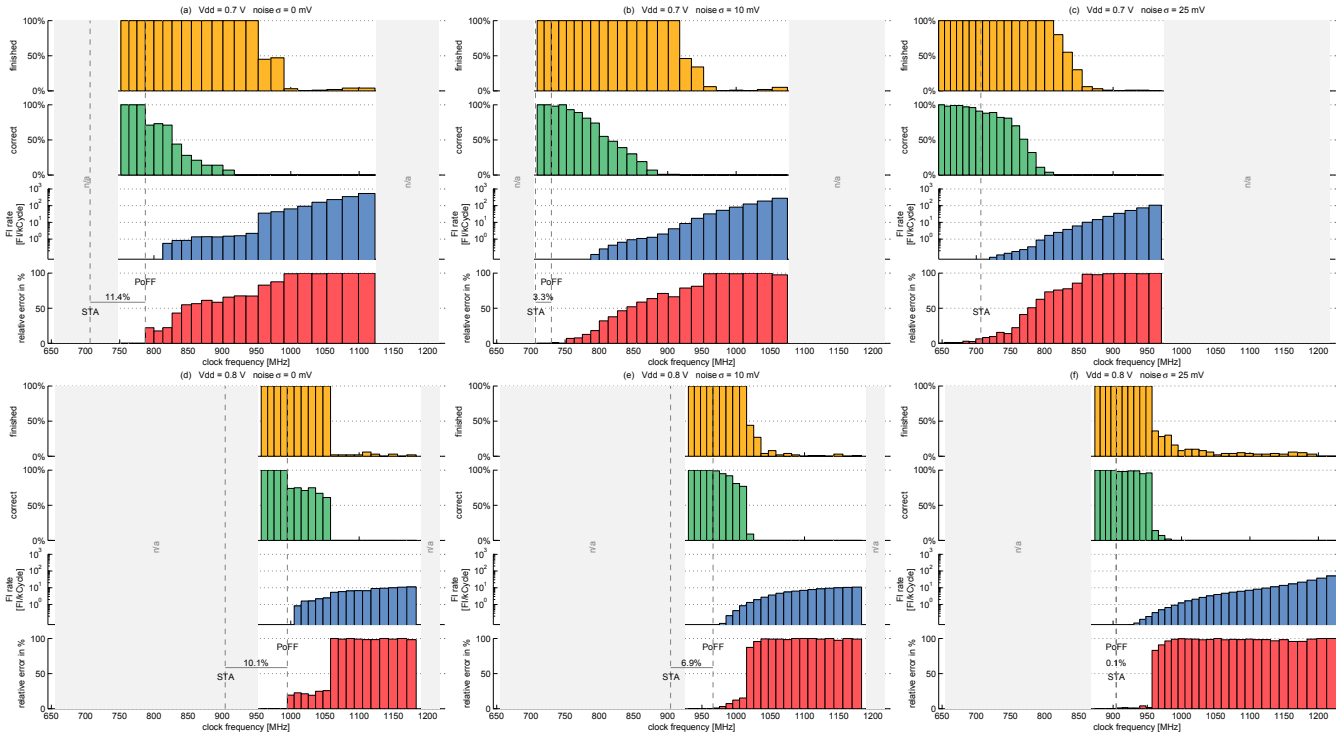

Figure 5: Program performance for the median benchmark for different V_{dd} and V_{dd}-noise (model C)

pact of the amount of noise on the transition ranges causing a shift for all four metrics down to lower frequencies. This can also clearly be seen by the impact of the voltage noise on the gain over the STA limit at the PoFF, which already disappears at a noise level of $\sigma = 25\,mV$. It can furthermore be noted that increased voltage noise causes the transition regions to be smoothened out, especially for the output error metric, mainly caused by the more gradual increase in the FI rate.

As can be seen in all configurations, as soon as the probability of the program to finish reaches low values, the output error of the remaining successful runs quickly saturates.

Looking at the effect of supply voltage, one can observe that a higher supply voltage results in sharper changes in the transition regions, which also means that the application error explodes more rapidly after the PoFF. Hence our analysis indicates that lower supply voltage seems to favor gradual failure behavior, which is often desired in approximate computing applications.

4.3 Performance Comparison of Benchmarks

Another key aspect of our model C is the distinction between different operations. This distinction is important when considering different types of kernels which rely on different instruction types and sequences. We demonstrate how this behavior is exposed by comparing different benchmarks in Fig. 6 at an operating point of 0.7 V with a supply noise of $\sigma = 10\,mV$.

Comparing 8-bit and 16-bit matrix multiplication (Fig. 6(a) & 6(b)), we see very similar application behavior. The lower bit-width helps in the region below the STA frequency to have a significantly higher rate of fully correct benchmark executions where timing errors are still induced due to the supply voltage noise. The MSE develops similarly for both bit-widths, with a factor of about 10^3 between them, due to the different operand and result ranges.

Compared to the matrix-multiplication benchmark, the k-means benchmark (Fig. 6(c)) experiences a fault injection rate that is almost one order of magnitude lower at the same operating frequency, which can be explained by the significantly lower number of more timing critical multiplications. Nevertheless, the kernel shows a considerable performance degradation (30-40%) of the quality metric, even though the code is still able to finish execution for a similar frequency range above the STA limit.

Finally, the Dijkstra benchmark (Fig. 6(d)) is characterized by only a very narrow transition region (hence it was simulated with a higher resolution). We can observe that although the kernel shows a frequency gain at the PoFF (which the others do not, apart from the median benchmark), 4% of further frequency increase beyond the PoFF already causes the application to fail completely, while still having a very low FI rate (below 1 FI per kCycle).

Fig. 6 furthermore contrasts the high characterization detail of our proposed model C with the observed behavior under model B+. As can be seen from the plot, the hard failure threshold at 661 MHz (also see Sec. 3.3) applies to all benchmarks equally, which means that model B+ does not allow for any of the provided analysis in the relevant transition region.

4.4 Error vs. Power Consumption Trade-Off

An important motivation for avoiding unnecessary voltage margins at the risk of errors or quality degradation are power savings. The proposed simulation model C improves the corresponding trade-off analysis by accounting for the underlying hardware structure.

To illustrate this analysis for the interesting transition region between nominal/safe voltage levels and complete system failure, we characterize achievable power savings in relation to the output quality for the median benchmark. The system operates at a fixed nominal frequency of 707 MHz (the STA limit at 0.7 V). The quality metric is computed by using our simulation model C, while the power savings are obtained by translating potential frequency-over-scaling

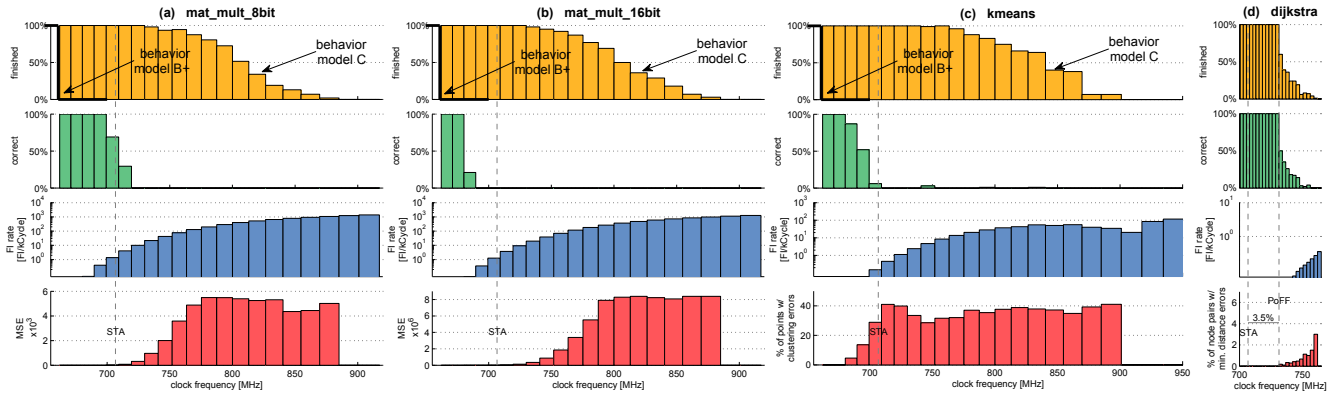

Figure 6: Program performances for various benchmarks at $V_{dd} = 0.7\,V$ with V_{dd}-noise $\sigma = 10\,mV$ (model C)

performance gains into an equivalent reduction of the supply voltage[2].

Fig. 7 details the trade-off analysis. The bottom right of the figure indicates the nominal operating point at a normalized core power of 1, and fully error free behavior. At a relative core power of 0.93 (due to a V_{dd} reduction of 33 mV) we reach the PoFF if no supply voltage noise is present. Scaling the voltage further shows a relative power of 0.88 with an average relative output error of 22%.

Considering also the impact of supply voltage noise, we observe that at $\sigma = 10\,mV$ the error-vs-power curve still relatively closely follows the no-noise configuration, with some higher power costs for equal quality. At $\sigma = 25\,mV$ however we observe a significantly earlier rise in output error, indicating that only marginal power gains are possible for a reasonable output quality.

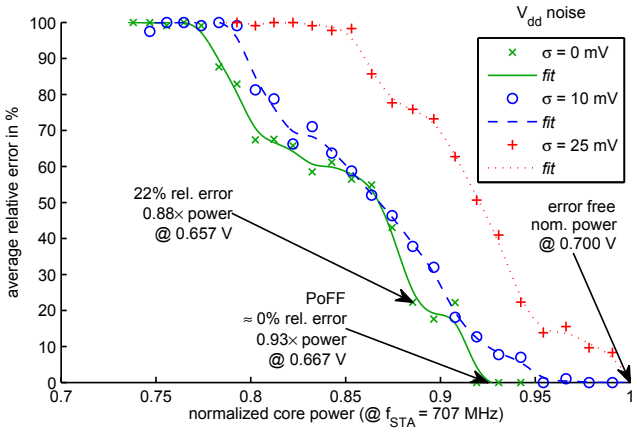

Figure 7: Relative error vs. core power consumption trade-off for the median benchmark (model C)

5. CONCLUSION

Modeling of timing errors on high-level simulators enables the rapid evaluation of application performance in terms of output quality, while overscaling frequency and/or supply voltage. However, the evaluation of the impact and identification of the reliability bottlenecks heavily depends on the accuracy of the employed characterization model. We show that in contrast to conventional methods, based on purely random FI or static timing based FI, our proposed statistical fault injection approach provides a significant improvement in modeling accuracy for the important transition regions between fully error-free operation and total circuit failure. The proposed approach achieves these improvements through the use of instruction based timing statistics, obtained by dynamic timing analysis at the gate level of a placed & routed processor.

[2]The power savings from this voltage-over-scaling are calculated by quadratic scaling of the consumed active core power between two reference points obtained by vcd-based gate level post layout simulations. The reference points are $10.9\,\mu W/MHz\,@\,0.6\,V$ and $15.0\,\mu W/MHz\,@\,0.7\,V$, while leakage (of the core) only consumes 2% and 3%, respectively.

6. REFERENCES

[1] D. Ernst *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *IEEE/ACM MICRO*, pp. 7–18, Dec 2003.

[2] K. Bowman *et al.*, "A 45 nm resilient microprocessor core for dynamic variation tolerance," *IEEE JSCC*, pp. 194 –208, Jan. 2011.

[3] V. Chippa *et al.*, "Approximate computing: An integrated hardware approach," in *IEEE Asilomar*, pp. 111–117, Nov 2013.

[4] V. Sieh *et al.*, "VERIFY: Evaluation of reliability using vhdl-models with embedded fault descriptions," in *Fault-Tolerant Computing*, pp. 32–36, IEEE, 1997.

[5] D. Kammler *et al.*, "A fast and flexible Platform for Fault Injection and Evaluation in Verilog-based Simulations," in *IEEE SSIRI*, 2009.

[6] M. Sugihara *et al.*, "A simulation-based soft error estimation methodology for computer systems," in *IEEE ISQED*, march 2006.

[7] H. Cho *et al.*, "Quantitative evaluation of soft error injection techniques for robust system design," in *IEEE DAC*, 2013.

[8] J. A. Clark *et al.*, "Fault injection: a method for validating computer-system dependability," *IEEE Computer*, vol. 28, no. 6, pp. 47–56, 1995.

[9] B. Nicolescu *et al.*, "Detecting soft errors by a purely software approach: Method, tools and experimental results," in *IEEE DATE*, IEEE Computer Society, 2003.

[10] M. de Kruijf *et al.*, "Relax: an architectural framework for software recovery of hardware faults," in *IEEE ISCA*, 2010.

[11] S. K. S. Hari *et al.*, "mSWAT: Low-cost hardware fault detection and diagnosis for multicore systems," in *IEEE/ACM MICRO*, pp. 122–132, 2009.

[12] J. Wei *et al.*, "Quantifying the accuracy of high-level fault injection techniques for hardware faults," in *IEEE DSN*, 2014.

[13] P. Ramachandran *et al.*, "Hardware fault recovery for I/O intensive applications," *ACM Trans. Archit. Code Optim.*, vol. 11, pp. 33:1–33:25, Oct 2014.

[14] J. Constantin *et al.*, "Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment," in *IEEE DATE*, pp. 381–386, Mar 2015.

[15] Z. Wang *et al.*, "Fast reliability exploration for embedded processors via high-level fault injection," in *IEEE ISQED*, pp. 265–272, Mar 2013.