D2M: DATA-DRIVEN MODEL FOR FAST AND ACCURATE TIMING ERROR SIMULATION IN STATICALLY SCHEDULED MICROPROCESSORS

Yuanbo Fan Russ Joseph

Department of Electrical Engineering and Computer Science
Northwestern University
2145 Sheridan Road
Evanston, IL, USA
yuanbo@u.northwestern.edu rjoseph@eecs.northwestern.edu

ABSTRACT

The simulation of circuit level timing errors has become a critical component in the evaluation of many emerging architectures. However, existing methods for injecting these errors tend to be either excruciatingly slow or rather inaccurate. We show that by dynamically building an error model through the use of supervised learning, excellent speedups can be achieved while maintaining little divergence from cumbersome gate-level simulation. We demonstrate performance improvements as great as 40x while limiting the Root Mean Square (RMS) divergence of estimated and true error rates to 0.5% on a range of the SPEC CINT2006 benchmarks. This is a significant result because it offers great hope for accelerating simulation and easing the evaluation of new architectures.

Keywords: circuit-level timing error, error rate, gate-level simulation, architecture-level simulation

1 INTRODUCTION

Many recently proposed architectures seek to circumvent traditional design margins and conservative design practices in an effort to reclaim performance or energy. These novel architectures push run-time operation past the conventional safety margins for voltage and clock frequency into an operating regime where circuit-level timing constraints cannot always be met. In the first category of these architectures detect and recover from the errors, allowing the system to produce architecturally correct results at discounted energy costs. (Ernst, Kim, Das, Pant, Rao, Pham, Ziesler, Blaauw, Austin, Flautner, and Mudge 2003) (Das, Tokunaga, Pant, Ma, Kalaiselvan, Lai, Bull, and Blaauw 2009) In the second category of systems, these errors may be allowed to propagate to the output leading to approximate results. (Sartori, Sloan, and Kumar 2011) (Sampson, Dietl, Fortuna, Gnanapragasam, Ceze, and Grossman 2011) (Esmaeilzadeh, Sampson, Ceze, and Burger 2013) Both approaches promise intriguing energy or performance gains.

High quality timing error models are critical to the evaluation of these systems but are challenging to deploy because they typically imply high complexity and cumbersome simulation overhead. In timing sensitive systems, the distribution of the timing errors has a significant impact on the performance of the system and failure to account for this leads to flawed evaluation and possibly an inadequate design. However, traditional means of injecting high-quality timing errors into simulators require gate-level simulation and therfore make evaluation intractable for all but the smallest designs. With the emergence of timing error sensitive design paradigms, we need tractable mechanisms for accurately modeling errors within complete systems.

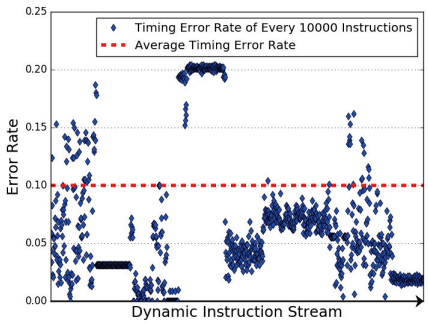
In this paper, we propose an automatic flow which offers high fidelity error models without compromising simulation time. Our approach applies machine learning, detailed hardware representation, and real program input to dynamically construct an error model which can be integrated into an existing microarchitectural simulator. This allows the simulation of timing errors to approach the fidelity of the slow detailed gate-level simulation while achieving simulation speed comparable to the fast but inaccurate analytic models. The contributions of this work include:

- We demonstrate the strong relationship between program-level features and circuit-level phenomenon in statically scheduled microprocessors.
- We build a framework that combines microarchitectural and gate-level simulations to efficiently estimate accurate timing error rates with guidance of preset accuracy target.
- We propose a methodology to automatically build adaptive error models with in-situ data collected from gate-level simulation using supervised learning on statically scheduled pipelines.
- We demonstrate the flexibility of framework with various supervised learning algorithms and quality thresholds, and the robustness over various Process, Voltage and Temperature (PVT) variations and error rate levels.
- We show that the proposed technique achieves on average less than 0.5% Root Mean Square (RMS) divergence from the true error rate (6.49% on average) over a range of the SPEC CINT2006 benchmarks.

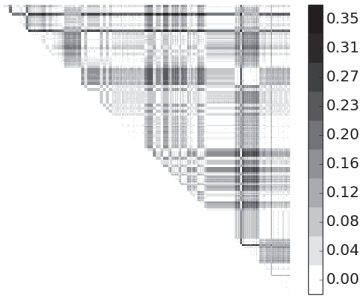
This paper is organized as follows: Section 2 introduces some background and related work and Section 3 describes the proposed timing error simulation framework. Section 4 describes our methodology setup, and Section 5 evaluates our proposed technique compared to other alternatives. Finally, Section 6 concludes.

2 BACKGROUND AND RELATED WORK

The present state of error models for circuit-level timing speculation leaves system evaluators with a difficult choice: (a) sacrifice the fidelity of the error model in exchange for tractable simulation time or (b) apply a more sophisticated error model and abandon either the coverage or reasonable simulation run-time. We divide previous efforts for modeling timing errors into two main categories:



(a) Timing error rates of consecutive program intervals (every 10000 executed instructions) in 401.bzip2 show large variation.



(b) Similarity matrix of interval error rates for ten benchmarks. The diagonal of matrix represents the execution of benchmarks concatenated. Each point is the error rate difference between two intervals (projected horizontally and vertically to the diagonal). The darker, the more different.

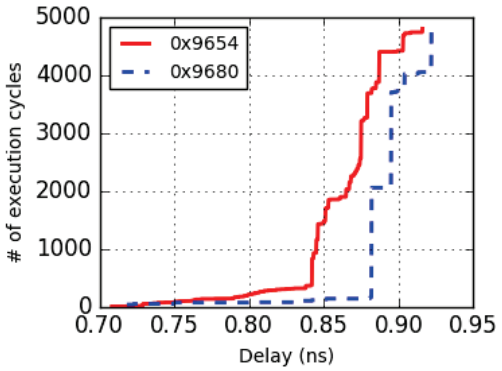
Figure 1: Error rates of continuous intervals show large variations both within and across benchmarks.

Input Agnostic Models which use analytic or empirically constructed models to estimate the probability of an error for a given instruction or execution unit as a function of voltage, temperature, and frequency. These an be evaluated quickly within the micro-architectural simulation, but do not consider the effect of data values on error rate and hence may not give the most accurate results. VARIUS is a well-known example of this approach. (Sarangi, Greskamp, Teodorescu, Nakano, Tiwari, and Torrellas 2008)

Gate-level Models which evaluate detailed gate-level descriptions of the hardware, essentially simulating the switching of gates to construct a comprehensive picture of the error rate, albeit with significant performance slowdown due to the complexity of the model. These models offer faithful distribution of timing errors including data-dependent effects, but are several orders of magnitude slower than current microarchitectural simulators, which may not be practical for evaluating full-size benchmarks on real, complex designs. To mitigate prohibitively slow simulation speed, researchers often use abbreviated instruction execution streams of benchmarks as representative workloads, or smaller input sets (i.e. the test or train sets rather than all of the reference sets in the SPEC CINT2006). However, due to large variation in error rate distribution and input-dependent behavior, partial simulation on narrow input sets of benchmark may give inaccurate or misleading results. (Wunderlich, Wenisch, Falsafi, and Hoe 2003) (Lafage and Seznec 2001) (Sherwood, Perelman, Hamerly, and Calder 2002)

```
BZ2_blockSort:
   ... /* skipped instructions */

   9640:    ldr    r0, [r5, lr, lsl #2]
   9644:    sub    r2, r2, #1
   9648:    str    r0, [r3]
   964c:    str    r4, [r5, lr, lsl #2]
   9650:    sub    lr, lr, #1
   9654:    cmp    r6, r2
   9658:    bgt    9688 <BZ2_blockSort+0x97c>

   965c:    add    r3, r5, r2, lsl #2
   9660:    ldr    r4, [r3]
   9664:    add    r0, r4, r1
   9668:    ldrb   r0, [r8, r0]
   966c:    cmp    r0, r7
   9670:    beq    9640 <BZ2_blockSort+0x934>
   9674:    bcc    95c0 <BZ2_blockSort+0x8b4>

   9678:    sub    r2, r2, #1
   967c:    sub    r3, r3, #4
   9680:    cmp    r6, r2
   9684:    ble    9660 <BZ2_blockSort+0x954>

   ... /* skipped instructions */
```

(a) Slice constructed for example instructions from two basic blocks in Function "blocksort" of 401.bzip2.



(b) Cumulative Delay Distribution of same static instruction, "cmp r6, r2" at 0x9680/0x9654 (marked in Table 2a) from two basic blocks in 401.bzip2.

Figure 2: Example of instruction timing delay.

Figure 1 shows that in real programs, it is common to have a significant temporal variation in error rates even if the voltage, frequency and temperature remain constant. Our experiments on a gate-level model of an ARM pipeline show that wide variations in timing error rates exist both within and across benchmarks. These variations become pronounced under extreme voltage/clock scaling. As Figure 1a shows, for one execution window (1M instruction) in 401.bzip2, error rates of intervals may vary about 20% (while the average of whole benchmark is only 5%). Moreover, the variation across benchmarks could be as high as 35%, as shown in Figure 1b. Such large variation makes it extremely difficult for any single simulation window to capture the true error rate.

While recent work has identified some of these problems and has offered an interesting alternative, scalability problems persist when studying whole processor designs. Jiao et al. proposed logistic regression based models to predict timing errors at bit-level. (Jiao, Rahimi, Narayanaswamy, Fatemi, de Gyvez, and Gupta 2015) (Jiao, Jiang, Rahimi, and Gupta 2017) It predicts timing errors with an average accuracy of 95% for small FPUs, such as adder, multiplier and square

root circuits. Tziantzioulis demonstrates the effect of history and correlation among output at bit-level (Tziantzioulis, Gok, Faisal, Hardavellas, Ogrenci-Memik, and Parthasarathy 2015). However, both methodologies are hard to scale to either larger designs (e.g. a full processor pipeline) or all possible application inputs.

Instruction timing delay is a strong function of instruction sequence and data usage, not only the static single instruction. As shown in Figure 2a, two basic blocks from 401.bzip2 have exactly the same static instruction marked by black dot (Line 3 on the left and Line 6 on the right). They have very different delay distribution within the execution window, as shown in Figure 2b. The connection between certain high-level instruction patterns and low-level circuit timing characteristics provides an opportunity to effectively accelerate timing error models on larger design using supervised learning algorithm. This high-level timing error model is trained to identify critical transitions on paths that may lead to timing errors, which can be used to drastically improve simulation.

3 TIMING ERROR SIMULATION FRAMEWORK WITH D2M

This section presents the proposed framework for timing error simulation using the Data-Driven Model (D2M), as shown in Figure 3. We use the term *Data-Driven* because the models in our approach are dynamically built on top of in-situ data labeled by circuit-level timing simulation which capture real program data and computation. Supervised learning is used to extract the correlation between high-level program pattern and low-level circuit timing characteristics.

This adaptive model is able to enhance microarchitectural simulation by inserting timing errors in a manner which retains high fidelity with full-blown gate-level simulation. Moreover, the proposed framework dynamically monitors the quality of timing error models to guide the decision of when to rebuild a new model if divergence is significant. The continuous fidelity tracking helps the simulation to maintain quality throughout the run. In addition, quality thresholds can be adjusted to allow tradeoffs in accuracy versus total simulation time.

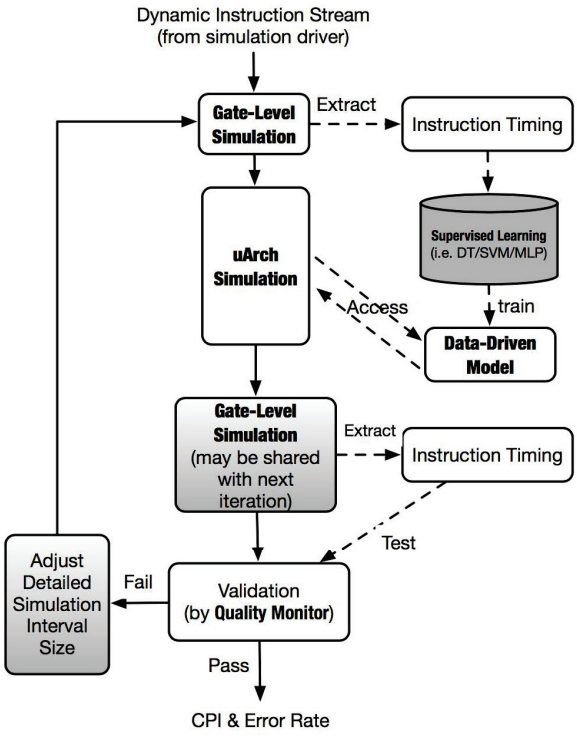


Figure 3: Proposed framework for timing error simulation with D2M.

3.1 Achieving accuracy and speed with both gate-level and microarchitectural models

Our framework consists of two simulation modes which respectively use gate-level and microarchitecture-level models of the processor design. Gate-level models are used for detailed timing simulation which remains a gold-standard for the error models. Despite the accuracy, gate-level simulation is a time consuming proposition. Event-driven simulation suffers from very low performance because of its inherently sequential nature and the sheer volume of activity in the entirety of the processor pipeline. Microarchitectural models, on the other hand, are widely used for cycle-level performance measurement and offer good scalability even

for large and complex designs. However, they on their own are incapable of timing simulation. They can be augmented by error models that mimic the delay faults that would appear in gate-level simulation.

In our approach we use brief periods of gate-level simulation to build accurate error models that can be used during the fast microarchitectural simulation. The gate-level simulation intervals provide accurate instruction timing information which is used as training data for supervised learning. Adaptive timing error models are trained based on this data and applied to microarchitecture-level simulation. Thus, the microarchitectural model is able to estimate timing errors while simulating at much faster speeds than traditional gate-level approaches.

3.2 Supervised Learning Based Approach: Data-Driven Model

Timing errors are strong functions of both instruction sequence and data usage in pipeline logic. This is because various input transitions for combinational logic will exercise different critical paths. In statically scheduled pipelines, this correlation is even stronger due to the fact that instruction sequence directly determines the execution order in the pipelines. By taking advantage of this property, supervised learning may be used to extract important "features" from labeled data and build adaptive timing error models accordingly. However, for moderate or large designs like a microprocessor, it quickly becomes an overwhelming task to build a single error model for the whole system and every possible input due to complexity of the circuit and input space. Therefore, even if there is a strong connection between input program and timing errors, it is hard to build a single comprehensive model for every benchmark under every possible dataset. We instead apply supervised learning algorithms to extract features in the timing simulation periodically which can identify dynamic critical transitions during given execution windows and rebuild a new model whenever the current model has low quality.

3.2.1 Model Training

We specifically select features: (1) that are visible at the instruction level during microarchitecture-level simulation and (2) that may trigger timing errors. Feature selection plays an important role in building a timing error model. One of our goals is to be able to identify relatively high-level program/instruction characteristics that can be easily tracked within a microarchitectural simulation and yet correlate well to timing errors which are rooted in circuit-level structure. One reasonable way to do this is to use a recent segment from the instruction stream which represents instructions which are currently present in the pipeline.

Due to the connection between high-level instruction patterns and low-level circuit timing characteristics, comprehensive input features should consist of all instructions that are present in the pipeline at the same time. However, only the instructions that have internal influence on each other may affect the circuit timing. To reduce the size of input features, we can pick the instructions that are internally relevant due to data and control dependence as input features of the model. For our hardware model, the timing delay of the most recent instruction may be influenced by maximally three preceding instructions due to control and data dependency logic. Thus, four consecutive instructions are used as a single input instance to make a prediction for timing error property in a cycle.

During detailed gate-level simulation, timing error information for each instruction is collected and used to label the input instance as either "Error" or "No Error". These labeled input instances are then used as training dataset to build a timing error model with a supervised learning algorithm. Data collected from gate-level simulation reflects timing characteristics of both hardware design and dynamic simulation environment such as Process, Temperature and Voltage (PVT) variations, thereby timing error models that are trained by

this data do not only make predictions for timing errors introduced by program, but also dynamic simulation conditions that the program is currently running under.

Our approach is general enough that various learning algorithms may be utilized with different corresponding tradeoffs in terms of accuracy and training/testing cost. Because the proposed framework is orthogonal to supervised learning algorithms, optimizations in algorithms would also help the framework to reduce cost from model training/testing or improve the accuracy of timing error simulation. We will demonstrate the tradeoffs of various algorithms in Section 5.4.

3.2.2 Model Evaluation

Good in field assessment of model quality is critical because it helps to direct the retraining and construction of better models. There are a few common pitfalls for evaluation of the model. Straightforward classification rate is not a good metric, because it is affected by the true error rate. The difference between the model estimate and true error rate is not a good metric to evaluate the model either due to the False Positive (FP) and False Negative (FN) in classification result. The Receiver Operating Characteristic (ROC) Curve cannot be used in this case either, because it is affected by imbalanced data set. According to our studies and analysis, we choose F_1 score (Sokolova, Japkowicz, and Szpakowicz 2006) as the metric to evaluate the performance of timing error model and will show the evaluation of F_1 score as metric in Section 5.3.

3.3 Guiding mode transitions with a quality monitor

In order to maintain the accuracy of the proposed framework, We dynamically monitor the quality of models to guide the decision of when to switch between two simulation modes, and F_1 score is used as the metric to evaluate the model quality. The evaluation of models is done periodically on separate testing set collected/labeled by gate-level simulation. Then, the next relevant question is: How should the threshold F_1 score be set? Our experiments show that the optimal threshold for different benchmarks may vary. This also provides an opportunity for us to trade off accuracy with simulation speed. We will show detailed analysis in Section 5.3.

3.4 Putting It All Together

The simulation repeatedly alternates between detailed gate-level and fast microarchitecture-level modes with guidance of the Quality Monitor. For a stream of instructions, a subset is simulated in detail while instruction timing is extracted. This timing information is then used to train an adaptive timing error model with supervised learning which is applied during fast microarchitecture-level simulation to estimate error rates. We then use the detailed simulation on a separate stream of instructions to produce testing data. The Quality Monitor then uses this to validate the current model. Since this procedure is repeated, we can always re-use the instruction timing extracted to validate the quality of previous trained models.

4 EXPERIMENTAL METHODOLOGY

This section describes our methodology in detail, including the hardware model, D2M training and some other important components in the proposed framework.

4.1 Hardware Model

Table 1: Full Gate-level Simulation (Ground Truth) Error Rates (%)

Cond.	perlbench	bzip2	gcc	mcf	hmmer	sjeng	libquantum	h264ref	omnetpp	astar	Avg.
PVT0; Overclock: 10%	6.01	9.12	4.48	7.90	5.07	8.71	3.11	2.79	10.17	7.55	6.49
PVT1; Overclock: 20%	15.16	18.95	11.44	21.94	12.66	15.12	9.48	9.65	23.85	16.66	15.49

To evaluate the proposed framework, we model logic timing errors in the execution pipeline by constructing a detailed gate-level model of ARM microprocessor. It supports the vast majority of user-mode integer ARM ISA instructions (Seal 2000). More specifically, we implemented a Register-Transfer-Level (RTL) model for 6-stage pipelined ARM microprocessor with timing speculation support using the Verilog Hardware Description Language (HDL), and it is synthesized using Synopsys Design Compiler (Kurup and Abbasi 2011) and target GLOBALFOUNDRIES 55nm ARM standard cell library (Online). The design is optimized for a frequency of 1 GHz, which we believe is reasonably aggressive target for a low-power design. The whole design is back annotated using Standard Delay Format (SDF). After constructing this detailed gate-level model, we link it to the gem5 simulator (Binkert, Beckmann, Black, Reinhardt, Saidi, Basu, Hestness, Hower, Krishna, Sardashti, Sen, Sewell, Shoaib, Vaish, Hill, and Wood 2011), so that two simulation modes (detailed gate-level and fast micro-architectural simulation) can be interleaved.

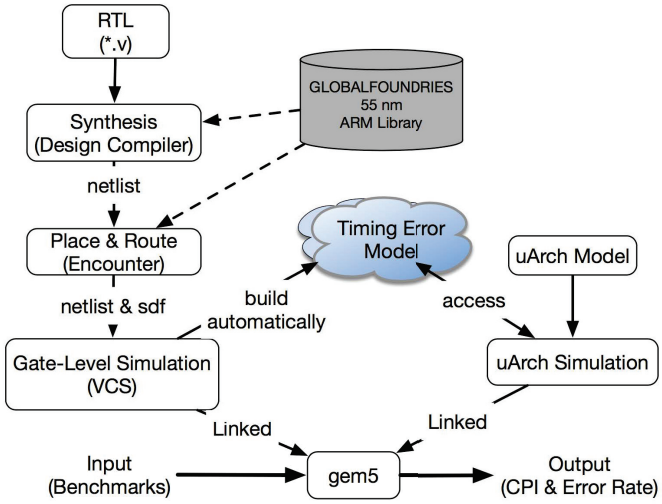


Figure 4: Experiment setup. Gate-level simulation is set up following the RTL design flow and the gem5 simulator is responsible for microarchitectural simulation.

In order to evaluate the proposed approach under different Process, Voltage and Temperature (PVT) conditions and error rates, the simulation is operated under two different environment conditions: PVT0: {Process: Typical; Voltage: 1.2V; Temperature: 25C} and PVT1: {Process: Slow; Voltage: 1.08V; Temperature: 125C}, we over-scaled the clock frequency by 10% and 20% respectively, as described in Table 1. The experiment setup is described in Figure 4.

4.2 Benchmarks

Due to the lack of Floating-point unit in our hardware model, we selected ten integer benchmarks from SPEC CINT2006 (Henning 2006) that are compatible with our infrastructure to evaluate our proposed technique. All benchmarks are cross-compiled for the ARM architecture using the LLVM (Lattner and Adve 2004) with the highest optimization level. We used SimPoint (Hamerly, Perelman, Lau, and Calder 2005) with the interval size of 100M instructions to identify phases in each benchmark and measured the overall benefits with weights of different phases.

Table 2: Comparison of different simulation mechanisms

Mechanism	Timing Error Model	Model Fidelity	Simulation Type	Benchmark Coverage	Simulation Cost	Overall Accuracy
Full Gate-Level (Ground Truth)	RTL/Netlist	High	Gate-Level	Whole	High	High
Independent Failure Rate (IFR)	Input Agnostic Model	Low	uArch	Whole	Low	Low
Sampled Gate-Level (SGL)	RTL/Netlist	High	Gate-level	Partial	Medium	Medium
Data-Driven Model (D2M)	RTL/Netlist + uArch	High	Gate-level + uArch	Whole	Medium	High

4.3 D2M Training

For D2M training, we applied the Decision Tree (DT) algorithm to build timing error models due to its low training/testing cost. We also other several commonly used supervised learning algorithms: Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP) in Section 5.4. We adapted the implementations from WEKA (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten 2009).

For feature selection, the information that is relevant to timing error includes the static machine code, and operand values used in the instruction. The static instruction may provide information such as condition codes, opcode, register indices and intermediate values. In our hardware model, the timing delay of the most recent instruction may be influenced by maximally three preceding instruction due to control and data dependency logic. Thus, we used four consecutive instructions as well as timing error label ("Error" or "No Error") collected from gate-level simulation, as a single input instance to make a prediction for timing error property of the last instruction. Since there is no weight for bit-wise transitions in circuit-level, each input data was converted into binary format and each bit was used as single input feature for both model training and accessing. For example, static instruction: 0xe3500001 (cmp r0, #1) would be converted to 32 input features: "1 1 1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1". And the label for timing error is either "1" (Error) or "0" (No Error).

5 RESULTS

In this section, we provide detailed comparison of the different simulation mechanisms in Section 5.1, followed by the evaluation of our approach with the impact of Process, Voltage and Temperature (PVT) variations in Section 5.2. We evaluate F_1 score as metric for the Quality Monitor and the tradeoff between accuracy and simulation time for various quality thresholds. We also examine the characteristics of various supervised learning algorithms (DT, SVM and MLP) in Section 5.4.

5.1 Accuracy and Speed Comparison

In order to evaluate our proposed technique, as well as some other alternatives, we use error rates measured using full Gate-Level Simulation (GLS) for benchmarks as a gold standard and our basis for normalization as shown in Table 1. Next we compare our proposed framework with Independent Failure Rate (IFR) and Sampled Gate-Level (SGL) simulations, as illustrated in Figure 5 and Table 2. **IFR** simulation uses offline timing error data to determine failure probability of instructions and hardware components, and then injects failures with given penalties during execution. **SGL** simulates an appropriate subset of benchmark in detailed mode and uses sample error rate to estimate the error rate of the whole benchmark, similar to the approach introduced by SMARTS (Wunderlich, Wenisch, Falsafi, and Hoe 2003).

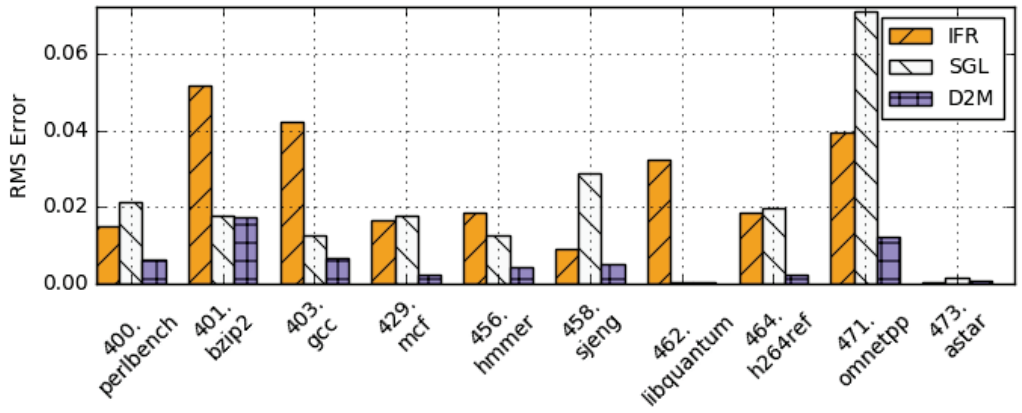


Figure 6: The Root Mean Square (RMS) difference of estimated and true error rates for different mechanisms across benchmarks.

Our proposed technique shows 4x $\sim$ 5x improvement in estimation accuracy, compared to IFR and SGL. Figure 6 illustrates that, compared to IFR, our proposed technique improves RMS divergence between estimated and true error rates by 5x (from 2.5% to 0.5%), while compared to SGL simulation our approach also outperforms by reducing Root Mean Square (RMS) divergence 4x (from 2% to 0.5%) with the exact same samples. Moreover, our technique shows much better consistence and stability in estimation. While IFR and SGL result in RMS divergence as high as 5% and 7.5% respectively, our approach achieves less than 1% RMS difference in 9 out of 10 benchmarks and only 2% RMS difference maximally.

Our proposed technique and conventional sample-based approaches both require detailed gate-level simulation on a subset of the benchmark. For sample-based approaches (e.g. SMARTS) confidence level and simulation intervals can be used to tune accuracy versus simulation time trade offs. In our approach, we use the Quality Threshold to guide this tradeoff. In addition, we capture error rate in the entirety of the benchmark by coupling the microarchitectural simulation with the timing error model. Compared to sample-based techniques applying proportion estimation in statistics, our proposed technique requires only 1/2000 sample size for detailed gate-level simulation to achieve the comparable estimation accuracy, as shown in Figure 7a.

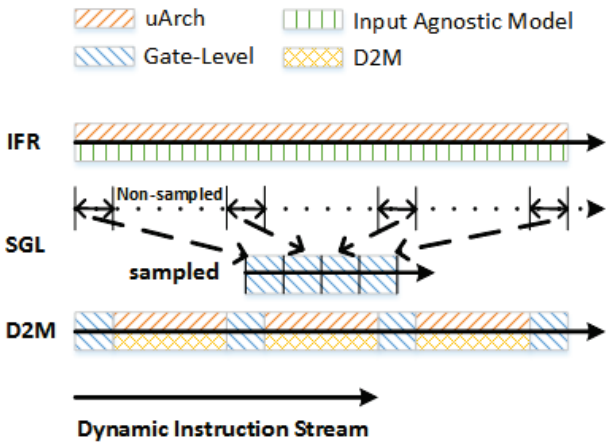
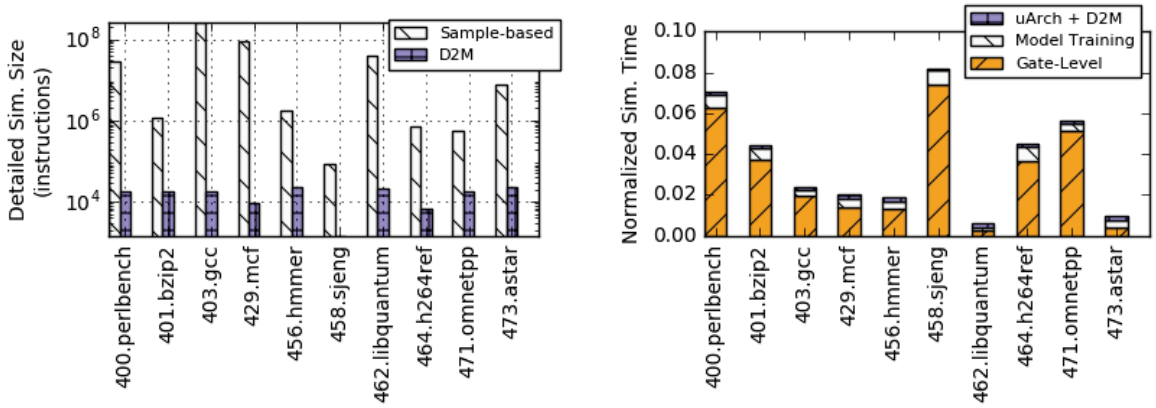


Figure 5: Example of three simulated mechanisms.

Our proposed framework introduces extra cost for model training/testing and micro-architectural simulation. However, in our experiments, this cost is very small since detailed gate-level simulation still dominates the overall simulation time. Also, as the design complexity grows, the model training/testing and micro-architectural simulation have much better scalability compared to gate-level simulation, we believe the relative cost would be small even for more complex designs.

Compared to gate-level simulation on whole benchmark, our approach speeds up simulation by 40x while resulting in RMS divergence within 0.5%. As the simulation time breakdown shows in Figure 7b, it consists



(a) Our proposed technique requires much fewer samples for detailed simulation to achieve the same confidence level and interval.

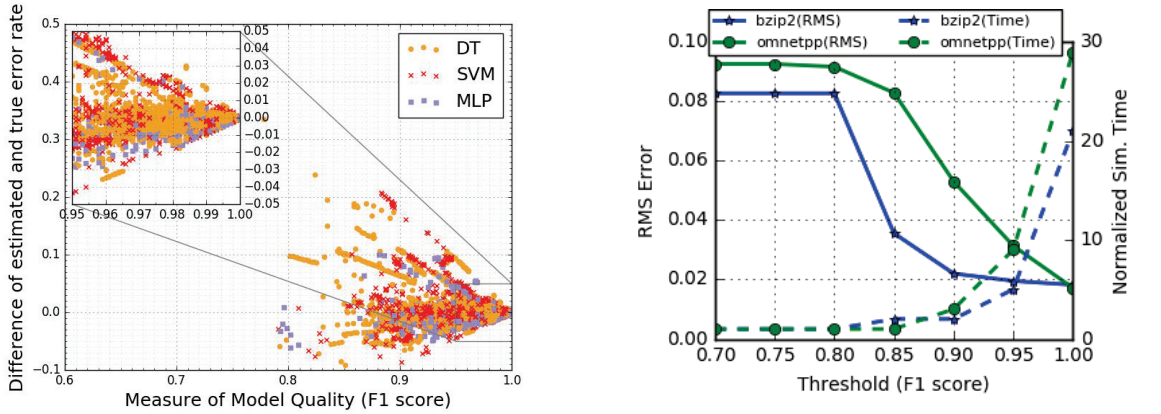
(b) Our simulation method adds tiny extra overhead due to model training/testing and micro-architectural simulation.

Figure 7: Simulation Speedup and Breakdown Compared to Sample-based Techniques.

of three major components including time consumed by detailed, fast simulations and the model building time. In practice, since the model is applied during fast simulation, it may add a tiny overhead to fast simulation as well. However, it is totally negligible. Our technique is also orthogonal to algorithm optimization and micro-architectural simulation improvement which could possibly further reduce this overhead.

5.2 D2M under PVT variations

To validate our proposed technique under extreme conditions, we simulated the design under PVT1 and 20% overclocking as described in Section 4.1 which results in much higher error rate than normal cases. Our proposed technique achieves high accuracy. The RMS difference between estimated and true error rates is less than 1.0%, compared to the RMS difference of 2% and 5% for IFR and SGL respectively.



(a) Correlation between model quality and estimation error.

(b) Tradeoffs between accuracy of simulation time for different quality threshold (F_1 score) across benchmarks.

Figure 8: Quality Metric (F_1 score).

5.3 Measure of Quality

F1 score shows consistency and stability as the metric to evaluate performance of models using various supervised learning algorithms. We experimentally verified the correlation between F_1 score and estimation error across various supervised learning algorithms. Models with a range of performance levels are built using different supervised learning algorithms (DT, SVM and MLP). The estimation error is defined as the difference between estimated and true error rate. As shown in Figure 8a, as the performance (metric value) improves, estimation error of models changes. When F_1 score is larger than 0.99, the estimation error is always within 1%, while the other metrics may have some noise even for high metric value. This criteria (F_1 score) also offers an opportunity to trade off model accuracy with simulation time. As shown in Figure 8b, when threshold of F_1 score is increased, it may require more detailed simulation to build more accurate timing error model which result in less speedup.

5.4 Comparison of Various Supervised Learning Algorithms

The name Data-Driven Model (D2M) implies that the quality of model depends on the training data set. Theoretically, most of supervised learning algorithms can be used to train a timing error model. However, different algorithms may have different characteristics. For example, as the complexity of algorithm grows, it may require a larger training data set to build a more accurate model, or the model it builds may be more complex which may require more build time. To compare the accuracy and cost of different algorithms, We select three widely-known supervised learning algorithms, DT, SVM and MLP. The accuracy of algorithms is measured by the RMS difference of estimated and true error rates on a separate testing set (different instruction streams). As shown in Figure 9, all three algorithms show better accuracy when training sample size increases. However, SVM and MLP require nearly 100x and 10000x longer training time compared to DT for the same sample size of 10000. However, SVM and MLP show better accuracy when the sample size is small. Therefore, for small sample size, probably more complex algorithms/models are preferred. When a large number of samples can be obtained, a more scalable algorithm, like DT may offer better accuracy.

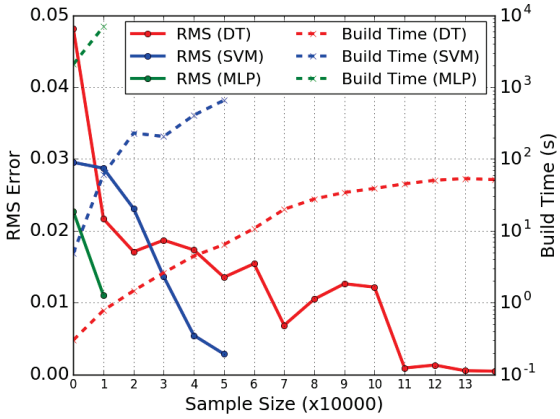


Figure 9: Accuracy and build time comparison for DT, SVM and MLP.

6 CONCLUSION

In recent years, architects have shown interest in systems that have some tolerance for circuit-level timing errors. Unfortunately, current methods for evaluating these architectures either engender very long simulation times or introduce considerable inaccuracy. In this work, we propose a novel flow for modeling timing errors in a complete processor pipeline. We show that by periodically training and applying machine learning models it is possible to achieve fast simulation times without compromising accuracy. This allows us to achieve very small RMS error rate divergence around 0.5% while producing a 40x simulation speedup over full gate level simulation. This offers great hope for evaluating an exciting class of architectures.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grants CCF-1116610 and CCF-1618065.

REFERENCES

Binkert, N., B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. 2011, August. "The Gem5 Simulator". *SIGARCH Comput. Archit. News* vol. 39 (2), pp. 1–7.

Das, S., C. Tokunaga, S. Pant, W. H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, and D. T. Blaauw. 2009, Jan. "RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance". *IEEE Journal of Solid-State Circuits* vol. 44 (1), pp. 32–48.

Ernst, D., N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. 2003. "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation". In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pp. 7–. Washington, DC, USA, IEEE Computer Society.

Esmaeilzadeh, H., A. Sampson, L. Ceze, and D. Burger. 2013, May. "Neural Acceleration for General-Purpose Approximate Programs". *IEEE Micro* vol. 33 (3), pp. 16–27.

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. 2009, November. "The WEKA Data Mining Software: An Update". *SIGKDD Explor. Newsl.* vol. 11 (1), pp. 10–18.

Hamerly, G., E. Perelman, J. Lau, and B. Calder. 2005. "Simpoint 3.0: Faster and more flexible program analysis". In *Journal of Instruction Level Parallelism*.

Henning, J. L. 2006, September. "SPEC CPU2006 Benchmark Descriptions". *SIGARCH Comput. Archit. News* vol. 34 (4), pp. 1–17.

Jiao, X., Y. Jiang, A. Rahimi, and R. K. Gupta. 2017. "SLoT: A Supervised Learning Model to Predict Dynamic Timing Errors of Functional Units".

Jiao, X., A. Rahimi, B. Narayanaswamy, H. Fatemi, J. P. de Gyvez, and R. K. Gupta. 2015. "Supervised learning based model for predicting variability-induced timing errors". In *NEWCAS*.

Kurup, P., and T. Abbasi. 2011. *Logic Synthesis Using Synopsys*. 2nd ed. Springer Publishing Company, Incorporated.

Lafage, T., and A. Seznec. 2001. "Workload Characterization of Emerging Computer Applications". Chapter Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream, pp. pp. 145–163. Norwell, MA, USA, Kluwer Academic Publishers.

Lattner, C., and V. Adve. 2004, Mar. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*. Palo Alto, California.

Online. *SC12 Standard Cell Library - GLOBALFOUNDRIES 55 nm 55LPX*. Available from https://www.design-reuse.com/sip/sc12-standard-cell-library-globalfoundries-55-nm-55lpx-ip-38111/.

Sampson, A., W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. 2011. "EnerJ: Approximate Data Types for Safe and General Low-power Computation". In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pp. 164–174. New York, NY, USA, ACM.

Sarangi, S. R., B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. 2008, Feb. "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects". *IEEE Transactions on Semiconductor Manufacturing* vol. 21 (1), pp. 3–13.

Sartori, J., J. Sloan, and R. Kumar. 2011, Oct. "Stochastic computing: Embracing errors in architecture and design of processors and applications". In *2011 Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, pp. 135–144.

Seal, D. 2000. *ARM Architecture Reference Manual*. 2nd ed. Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.

Sherwood, T., E. Perelman, G. Hamerly, and B. Calder. 2002. "Automatically Characterizing Large Scale Program Behavior". In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, pp. 45–57. New York, NY, USA, ACM.

Sokolova, M., N. Japkowicz, and S. Szpakowicz. 2006. "Beyond Accuracy, F-score and ROC: A Family of Discriminant Measures for Performance Evaluation". In *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI'06, pp. 1015–1021. Berlin, Heidelberg, Springer-Verlag.

Tziantzioulis, G., A. M. Gok, S. M. Faisal, N. Hardavellas, S. Ogrenci-Memik, and S. Parthasarathy. 2015, June. "b-HiVE: A bit-level history-based error model with value correlation for voltage-scaled integer and floating point units". In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6.

Wunderlich, R. E., T. F. Wenisch, B. Falsafi, and J. C. Hoe. 2003. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling". In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pp. 84–97. New York, NY, USA, ACM.

AUTHOR BIOGRAPHY

Yuanbo Fan is a graduate student in the Electrical Engineering and Compute Science Department at Northwestern University (Evanston, IL), advised by Prof. Russ Joseph. Hist primary research interest in computer architecture, specializing in architecture-level modeling, design and implementation power efficient architectures.

Russ Joseph is an associate professor in the Electrical Engineering and Computer Science Department at Northwestern University (Evanston, IL). His primary research interest is in computer architecture, focusing on the design and implementation of power-aware and reliability-aware computer systems.