Revamping Timing Error Resilience to Tackle Choke Points at NTC Systems

Aatreyi Bal, Shamik Saha, Sanghamitra Roy, Koushik Chakraborty

USU BRIDGE LAB, Electrical and Computer Engineering, Utah State University
bal.aatreyi@gmail.com, shmk_saha@yahoo.co.in, {koushik.chakraborty, sanghamitra.roy}@usu.edu

ABSTRACT

Process variation is a conspicuous predicament for sub-micron VLSI circuits. In this paper, we illustrate "choke points" as a vital consequence of process variation in the Near Threshold Computing (NTC) domain. Choke points are process variation affected sensitized logic gates with increased delay deviation. They dominate the choice of critical paths post-fabrication. To mitigate the timing errors induced thereby, we propose *Dynamic Choke Sensing (DCS)*. This technique senses the timing error causing opcode sequences, and uses the knowledge to prevent similar sequences from causing errors in future. Our scheme provides 25%-160% improvement in performance and 50%-90% improvement in energy efficiency as compared to contemporary timing error mitigation schemes, with minimal area and power overheads.

1. INTRODUCTION

Near-Threshold Computing (NTC) has emerged as one of the promising directions in the pursuit of improving the energy-efficiency of computer designs [5,12]—a growing concern in the current geopolitical landscape. A device operating at NTC sets its supply voltage close to its threshold voltage, while still maintaining a positive difference between them. Consequently, the energy consumption is dramatically reduced, both due to the quadratic impact of supply voltage reduction, as well as, a linear reduction from the operating frequency. Therefore, NTC plays a pivotal role in mitigating dark silicon [16]. Overall, an NTC device delivers orders of magnitude improvement in energy efficiency [8], combating the power wall in modern many-core systems, albeit at a significant cost of performance.

NTC's tremendous energy efficiency benefit comes with its own set of challenges. Besides 10X or more performance degradation, NTC suffers from exacerbated process variation (PV) sensitivity [7]. PV at NTC can vary the gate delays by as large as 20X of their nominal values [15]. Due to this massive delay variation, NTC circuits have a substantially higher reliability threat from *choke points*—a small set of PV affected gates that can dominate the selection of critical paths in the post-silicon circuit—than their Super-Threshold Computing (STC) counterparts [4].

In this paper, we use an extensive circuit-architectural analysis to illustrate how choke points are formed and their resulting impact on conventional STC circuits and rapidly evolving NTC circuits, respectively. We observe that a choke point can be formed with as small as 0.17% of the total gates in an NTC ALU—a circuit with a large logic depth—causing a delay degradation of 23.7%. These intriguing characteristics can substantially degrade the critical path delay at NTC, while also radically altering the composition of critical paths in a fabricated circuit by massively degrading the delay in short paths. In addition, choke points cannot be estimated pre-fabrication. A batch of identical chips may have a large variation in the distribution of choke points, post silicon. Thus, existing physical design techniques are rendered inefficient in mitigating this problem. To overcome these limitations, we propose a low overhead and dynamically adaptive timing error mitigation technique, called Dynamic Choke Sensing (DCS).

Our main contributions in this paper are as follows:

- We focus on the gate-delay deviation caused by process variation at NTC and its role as a major source of timing errors. We establish choke points as a significant outcome of this delay variance (Section 2).
- We propose a low overhead dynamic timing error prediction and mitigation technique, called Dynamic Choke Sensing (DCS). Our technique performs an early choke point detection and, thereby, uses the knowledge to avoid recurrent timing errors. As a result, penalty cycles incurred to recover from timing errors are reduced (Section 3).
- We demonstrate that our scheme provides 20%-50% improvement in performance and 50%-83.7% improvement in energy efficiency as compared to representative timing error recovery technique, Razor [6] (Section 5). The area, wire-length and power overheads of DCS are 0.23%, 0.77% and 0.85%, respectively.

2. MOTIVATION

In this section, we demonstrate that circuits operating at NTC have a substantially higher chance of manifesting choke points than their STC counterparts. We also present an extensive circuit-architectural analysis to illustrate how

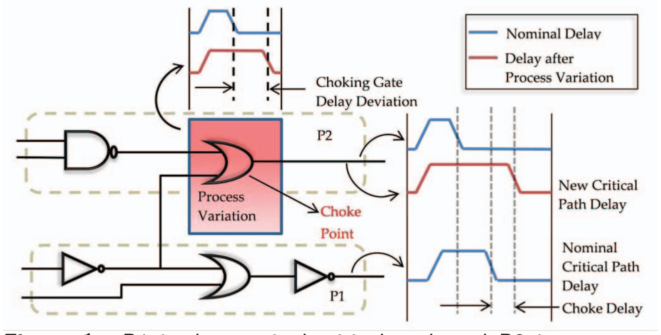
Nominal Delay
Delay after Process Variation
Choking Gate Delay Deviation
P2
Process Variation
Choke Point
New Critical Path Delay
Nominal Critical Path Delay
P1
Choke Delay

Figure 1: *P1 is the nominal critical path and P2 is a non-critical path pre-fabrication. P2 becomes the new critical path post-fabrication owing to the increased gate delay of the choke point.*

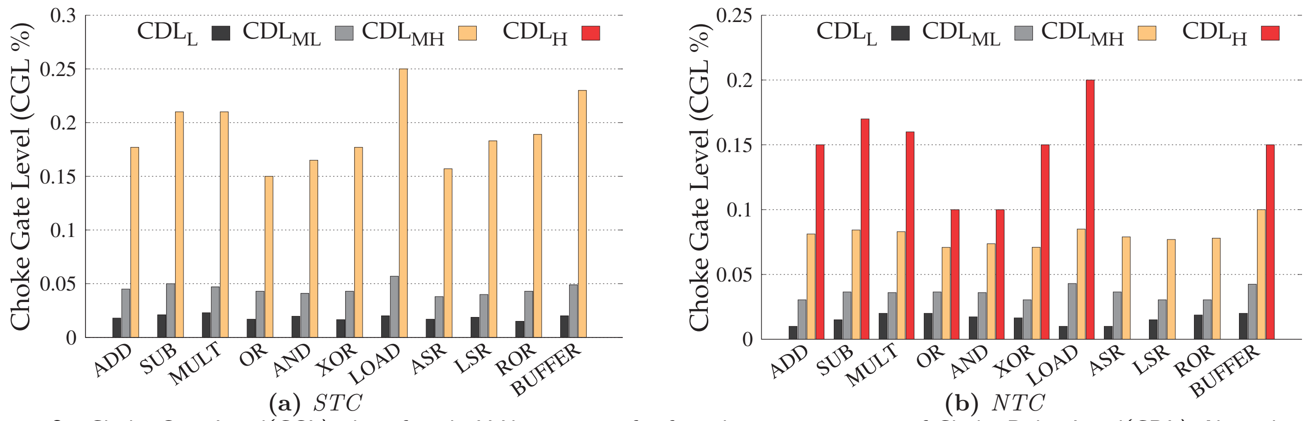

Figure 2: *Choke Gate Level(CGL) plot of each ALU operation for four distinct categories of Choke Delay Level(CDL). Note that the Y-axis scales upto 0.25% and 0.2% at STC and NTC, respectively.*

a small number of PV affected gates at NTC can serve as choke points, potentially transforming a shorter delay path into a critical path, after chip fabrication. Finally, we discuss the unique challenges in robust circuit design, presented by choke points, as we move from STC to NTC. In Section 2.1, we define *choke points* and discuss their potency. Sections 2.2, 2.3 and 2.4 present the methodology, results and significance of our study.

2.1 Choke Points

A choke point comprises a single or a small group of PV affected gate(s) that can transform a shorter delay path into a critical path (Choke Path), when sensitized[1]. Further, choke points can also create critical paths with substantially higher delays than the nominal. Figure 1 illustrates the concept of a choke point in a small circuit. P1 is the nominal critical path. A post-silicon instance of this circuit, however, suffers a high delay variation due to the OR gate in the path P2. The OR gate serves as a choke point, transforming P2—a short delay path—as the new critical path.

Though choke points can be formed anywhere in a circuit, their effects are observed only in sensitized paths. Typically, the distribution of sensitized paths in a circuit depends on the instructions executed on the circuit, as well as, the inputs to those instructions.

2.1.1 *Potency of a Choke Point*

We define two key parameters—Choke Delay Level (CDL) and Choke Gate Level (CGL)—to quantify the potency of a choke point. CDL is the amount of additional delay introduced by the choke point to create the new critical path, overshooting the nominal critical path delay. It is expressed as a percentage of the nominal critical path delay. CGL is the number of gates forming the choke point, expressed as a percentage of the total gates in the whole circuit. A low CGL signifies a highly potent choke point, where, a small percentage of gates in the circuit can transform the critical path. Similarly, a high CDL also indicates high potency of a choke point.

2.1.2 *Threat of a Choke Point*

Analyzing the effects of choke points in the sensitized paths is of paramount importance for the reliable operation of a NTC system. To understand the extent of threat presented by choke points in modern processors, we pose a few key research questions. How can we compare choke point significance in STC and NTC circuits? What is the likelihood of choke points in transforming critical paths in popular processor pipelines while running real applications ? To answer these compelling questions, we employ a rigorous cross-layer methodology, outlined next.

[1]A path is sensitized when the applied input changes the output state of the path.

2.2 Methodology

Analyzing sensitized choke points in STC and NTC presents a methodological challenge. As instructions are executed in a circuit component, they sensitize different paths, and therefore observe different logic computation delays. Further, the sensitized path in a circuit depends on two consecutive instructions [18]. For this analysis, we model the PV in the logic gates at STC and NTC, on the basis of VARIUS [14] and VARIUS-NTV [7], respectively. Next, we synthesize a 64-bit ALU using a 15nm FinFET library from NanGate [10]. We use the PV-affected logic gate models in our in-house Statistical Timing Analysis (STA) tool to study the cycle accurate delay timings of all the sensitized paths for 11 different arithmetic and logic operations. The operands are chosen to cover a typical range seen in real applications. We extensively study the sensitized path delay distribution for different combinations of operations and operands to analyze potential choke paths. A detailed description of methodology is presented in Section 4.

2.3 Results

Figure 2a depicts the correlation of CDL and CGL for each operation of the ALU at STC. We present the data for four categories of CDL: CDL_L (0-5%), CDL_{ML} (>5%-10%), CDL_{MH} (>10%-20%) and CDL_H (>20%-30%). We find choke points to be formed typically in the range of 0 to 12% CDL. This trend can be attributed to the fact that individual gate delay deviation at STC is not large enough to surpass a CDL larger than 12%, even when all the gates in the choke path are affected by PV.

Figure 2b explores the same correlation at NTC. We find that choke paths can be formed at higher CDL with substantially smaller CGL at NTC. For example, in case of LOAD operation, a CGL of only 0.2% is sufficient to surpass a CDL of 27.45% while for AND, only 0.1% CGL can exceed a CDL of 23%. However, the choke point sentitization in the higher range CDL_H, varies across operations. In a few operations (like ASR, LSR and ROR), choke points fail to create new critical paths with higher CDL. The reason being, in the higher categories of CDL, typically the path with the maximum logic depth forms the critical path. Thus, other sensitized paths, with lesser logic depth, fail to emerge as potential choke paths in CDL_H category for these operations. On a deeper analysis, we find that choke point sensitization depends on the following architectural factors:

Nature of Operations: The nature of operations is a determining factor in the creation of choke points. The computation complexity involved in an operation significantly affects the path sensitization at a given cycle. The greater the number of intermediate calculation steps involved, more is the number of sensitized gates in the circuit. Consequently, the probability of most of the PV affected gates appearing in the sensitized paths, and thereby the potency of choke point creation, increases. This characteristic is clearly depicted by the ADD and BUFFER operations. Since ADD sensitizes more paths while calculating sum and carry, chances of sensitizing more PV affected gates in the process is high. BUFFER on the other hand simply passes the input to the output after one clock cycle. Therefore, in every category of CDL, ADD records lower CGL than BUFFER, depicting its higher potency in creating choke paths.

Significant Width of Operands: The significant width of the operands of an operation—the number of set bits in the operand—affects the probability of choke point formation. If the significant width is smaller than half the total width of operand, determined by the Instruction Set Architecture (ISA), we term it "low"; otherwise it is considered "high". Higher the significant width, more gates are likely to be sensitized across the paths, and greater is the possibility of the PV affected gates occurring in those paths. Therefore, higher significant width of operands have greater potency to create choke paths, even in the higher categories of CDL. We use Operand Width Marker (OWM) to denote if either of the operands in an operation has high significant width. If none of the operands has high significant width, OWM has reset value; else, it has set value. The potency of significant operand width in forming choke paths is depicted in Figure 3.

2.4 Significance

Our study shows that the heightened effect of PV on gate delay at NTC can massively degrade a short delay path into a choke path or substantially degrade the critical path delay in a fabricated circuit. In addition, choke points—an artifact of PV and NTC operation—cannot be estimated pre-fabrication. In fact, a batch of identical chips may have a large variation in choke paths, post silicon. This intriguing characteristic can render conventional physical design techniques of timing error mitigation, like gate-sizing and multiple threshold voltage (V_{th}) assignment, inefficient. Likewise, existing techniques like hierarchical guardbanding [13] or timing speculation can suffer from substantial energy efficiency loss in mitigating timing errors from choke paths. *Then, a key question is can we design a low overhead hardware inside every chip that can learn and adapt to its own choke paths?* In the next section we discuss our proposed scheme, Dynamic Choke Sensing (DCS), to mitigate timing errors from choke points at NTC.

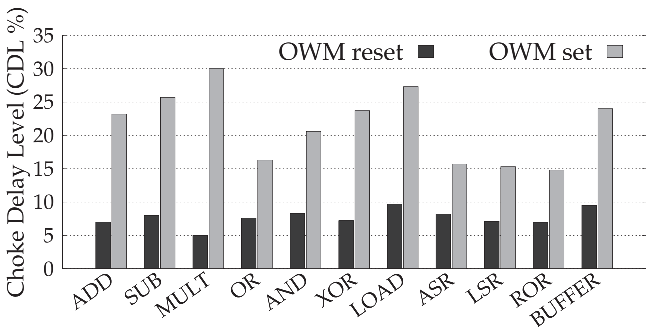

Figure 3: *Choke Delay Level (CDL) variation with Operand Width Marker (OWM) for each operation at NTC. Choke paths with higher CDL can be created when OWM is set, signifying higher significant width of either or both operands of an operation. Reset value of OWM signifies both the operands have low significant width and is less potent in creating choke paths.*

3. DCS DESIGN

In this section, we present a robust technique, Dynamic Choke Sensing (DCS), which exploits the factors discussed in Section 2.3, to mitigate timing errors. Choke points vary from chip-to-chip within the same design, due to variance in the degree of PV; but they comprise a permanent characteristic of a particular chip instance. Timing violations caused by choke points emerge as an inherent property of the chip. Though, newer timing violations may arise or existing violations may magnify due to aging, yet, an existing choke point will continue to cause timing violations for the entire lifetime of the chip. Our proposed technique, adaptively mitigates these dynamic timing errors. In Section 3.1 we present a brief overview of the technique. Sections 3.2 and 3.3 highlight on the methodology of the technique and the error handling mechanisms, respectively.

3.1 DCS Overview

Figure 4 portrays an overview of the components and flow of the DCS technique. DCS focuses on sensing errors and avoiding their recurrent occurrences. The two major components facilitating this tecnique are the Choke Controller and the Choke Sensor Lookup Table (CSLT). The CSLT serves as a record of the unique timing error instances. The Choke Controller performs the pipeline flush and instruction replay, when an error is detected for the first time, and inserts the erroneous opcode into the CSLT. When the same errant opcode is detected in the decode stage of the pipeline, the Choke Controller avoids the error by inserting a stall cycle. These two components perform in unison to reduce the penalty cycles for repeated timing error recovery, thereby improving the overall performance of the system.

3.2 DCS Stages

DCS mechanism operates in three interlinked stages, that are discussed next.

3.2.1 Choke Sensing

When a timing error is sensed for the first time, it is recorded as a tuple in the CSLT. The table entries are man-

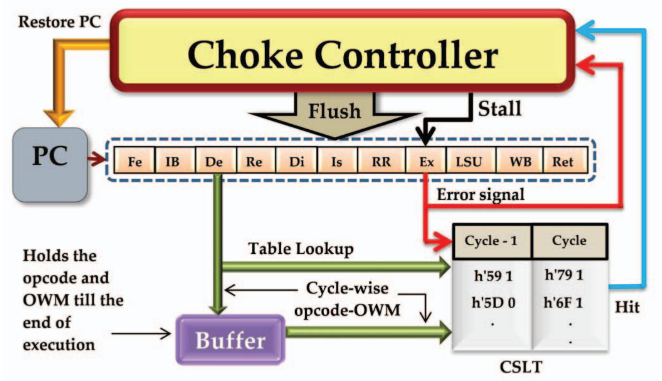

Figure 4: *Every cycle, the decoded opcode is looked up in the table. If there is a match, in both current and previous cycle opcode-OWM, the Hit signal is set high. If there is no match and an error is detected in the execute stage (Ex), the Error signal is set high and the 8-bit opcode and 1-bit OWM for the errant cycle and previous cycle from the buffer are latched to the CSLT. The Choke Controller regularly checks both the Hit and Error signal. If the Hit is high, it inserts a stall cycle in Ex-stage for corresponding opcode. If Error signal is high, pipeline flush and instruction replay are initiated by the controller.*

aged dynamically, in the form of a Random Access Memory (RAM). For the sensing mechanism, we use double-sampling flip-flops at the output of potential sensitized non-critical paths, similar to Razor [6]. The potential paths are identified by the method suggested by Lak et al. [9]. The tuple in the table consists of the errant opcode and the corresponding OWM (discussed in Section 2.3). However, instead of using a single opcode-OWM pair, we use a sequence of opcode-OWM pairs from consecutive cycles, the errant cycle and the previous one, to tag a particular timing error instance. The sequence characterizes the changes in the output states of the paths, over this one cycle time period, which trigger the timing error. A buffer is implemented to hold the opcode-OWM values from decode (De) stage till the writeback (WB) stage, in order to preserve the previous cycle data until the current opcode completes execution (Ex) stage. The unique tag allows us to monitor the timing error instances at a finer granularity. When the CSLT becomes full, pseudo-LRU (Least Recently Used) technique is used to evict existing tags and make space for new entries.

3.2.2 *Choke Error Recovery*

After a timing violation has been recorded, the Choke Controller initiates a pipeline flush to erase the contents of all the pipe stages. The flush is followed by an instruction replay to repopulate the stages and resume the flow of execution. This recovery mechanism incurs a penalty of P cycles, where P is the number of pipe stages.

3.2.3 *Timing Error Avoidance*

Repeated timing errors caused by same sequence of opcode-OWM pairs are avoided by the table lookup mechanism. For each cycle, the corresponding opcode-OWM is looked up in the CSLT during the decode stage. An error causing opcode-OWM sequence is likely to repeat its behavior in subsequent cycles, under same operating conditions. The Bloom Filter [2] mechanism is implemented to lookup the table using the tag described in Section 3.2.1. If a match is found, we expect a timing error to occur again. Hence, the Choke Controller stalls the subsequent opcode before the execution stage for an additional cycle and guarantees the propagation of correct results thereafter.

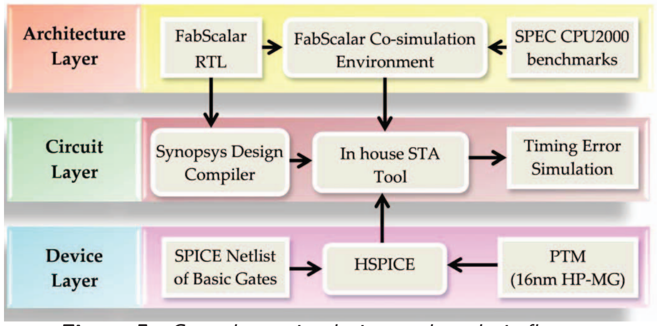

Figure 5: *Cross-layer simulation and analysis flow.*

3.3 Error Handling

Depending on the type of error encountered during the table lookup, the penalty cycle count varies. A false positive match returned by the CSLT results in an additional stall cycle penalty. However, a false negative match incurs higher penalty cycles, as the Choke Controller initiates a pipeline flush and instruction replay when the error is finally encountered in the execution stage.

4. METHODOLOGY

In this section, we discuss the rigorous cross-layer methodology that we employ to establish the potency of our technique in a 11-staged pipeline. Figure 5 depicts the multiple layers that we broadly outline in the following sections. Section 4.1 discusses how we estimate the delay distribution of FinFETs as well as our methodology to estimate process variation. In Section 4.2, we use real world benchmarks to gather inputs for our in-house STA tool. Section 4.3 discusses the circuit level implementation of our technique.

4.1 Device Layer

In order to estimate the gate delay distribution of FinFETs at different operating voltages, we simulate HSPICE models of fundamental logic gates based on the Predictive Technology Model (PTM) for 16nm high-performance multi-gate devices [17]. We consider VARIUS [14] and VARIUS-NTV [7] for PV at STC and NTC, respectively. To model the impacts of PV in FinFETs, we take into account the analysis presented in [11]. We perform Monte Carlo simulation to evaluate the means and standard deviations of propagation delay distributions of the basic gates at STC and NTC regimes. These propagation delay values are then used in the circuit layer simulation, as discussed in Section 4.3.

4.2 Architecture Layer

We use FabScalar infrastructure [3] to perform the architectural simulation. We choose the Core-1 configuration. It has a 11-stage out-of-order superscalar pipeline that is capable of fetching, issuing and commiting 4 instructions in each cycle. In the execution stage, we employ a choke sensing mechanism and a tactic to insert stall cycles. For all the other stages, we apply a pipe stage flush and instruction replay procedure similar to Razor [6]. We use six

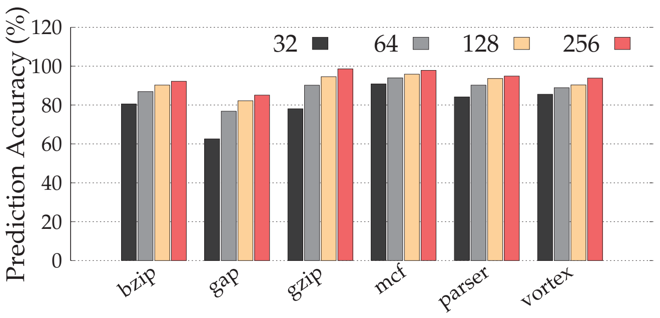

Figure 6: *Prediction accuracy of DCS across all benchmarks for 32, 64, 128 and 256 entries in CSLT.*

SPEC CPU2000 benchmarks that typify real world applications. We obtain the cycle-by-cycle input values for all these benchmarks that are used for the static timing analysis discussed in Section 4.3.

4.3 Circuit Layer

The circuit layer simulation has three stages. In the first stage, we synthesize the ALU RTL using Synopsys Design Compiler. For our study, we focus singularly on the choking in the execution stage of pipeline. We use 15nm NanGate Open Cell Library for FinFETs [10] to perform the synthesis. We also design and synthesize the components of the DCS scheme, to estimate the energy consumption, for energy efficiency analysis later (discussed in Section 5.4). In the second stage, we feed the synthesized netlist and the input values for all the benchmarks into our in-house STA Tool. The process variation induced gate delay values obtained from HSPICE simulation are also incorporated into the STA Tool. By the end of this stage, we procure the propagation delay values of the sensitized paths in each cycle for all the benchmarks. In the third stage, we utilize these delay values to perform the timing error simulation for diverse schemes. We calculate the runtime and number of penalty cycles encountered by each benchmark for each scheme to present a comparison of efficacy among them.

5. EXPERIMENTAL RESULTS

In this section, we illustrate our experimental results in comparison to popular existing techniques like Razor and HFG. Section 5.1 highlights the comparative schemes, while Sections 5.2, 5.3 and 5.4 present the detailed discussion of the prediction accuracy and relative performance and energy-efficiency gains of our technique, respectively. Section 5.5 discusses the area and power overheads of our technique.

5.1 Comparative Schemes

- **Razor**: This timing speculation based error detection and recovery scheme [6], detects a timing error by a double-sampling flip-flop at the end of each pipeline stage. The recovery mechanism flushes the pipeline stages and initiates an instruction replay.
- **HFG**: This scheme proactively prevents timing error [13]. It adaptively modifies the guardband, to account for PVTA (Process, Voltage, Temperature, Aging) variations throughout the device lifetime.
- **DCS**: Our proposed scheme senses timing errors caused by choke points. It then uses this knowledge to avoid similar potential timing errors in future. Moreover, it also reduces the timing penalty caused by repeated pipeline flush and instruction replay.

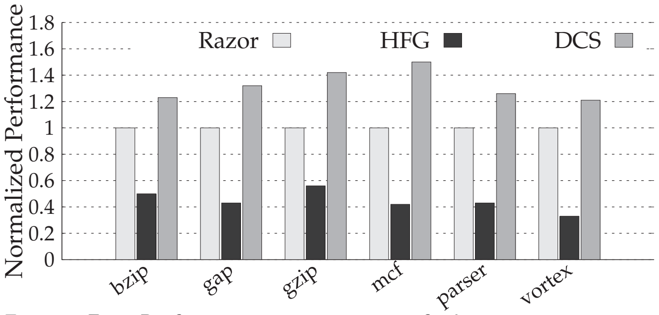

Figure 7: *Performance comparison of three comparative schemes for different applications (higher is better).*

5.2 Prediction Accuracy

Figure 6 shows the prediction accuracy of our scheme for different entry-sizes of the CSLT. We simulate each of the benchmarks for 1 million cycles and record the error and prediction counts for CSLT with different sizes of entries. The varied prediction accuracies displayed by different benchmarks is owing to the variance in number of unique error instances among them. The figure clearly shows that prediction accuracy varies minimally from 128 to 256 entries for most of the benchmarks. So, for a fair trade-off between prediction accuracy and space efficiency, we consider the CSLT size to be 128-entries for our further evaluations.

5.3 Performance Gain

Figure 7 depicts the performance gains achieved by our technique as compared to Razor and HFG. The performances of all the schemes are normalized with respect to Razor. HFG is seen to have the worst performance, among the three schemes, for all the applications. The reason being, HFG simply increases the clock period based on the guardband range. So even for a few instances of potential timing error, the overall runtime is increased in HFG. Razor, however, performs better as it allows the timing errors to occur and then initiates the recovery mechanism to avoid faulty data propagation. DCS shows the best performance among all the schemes, for all the applications. It uses the knowledge of previous timing error instances to foretell potential instances in subsequent cycles. Apart from the opcode itself, DCS also considers the operand width for choke point sensing, thereby harvesting fine-grained knowledge of the errant opcode.

Among the benchmarks, mcf–with the minimum number of unique tuples–displays 50% performance improvement, compared to Razor. On an average, DCS shows 30% performance improvement as compared to Razor and 150% as compared to HFG.

5.4 Energy Efficiency Gain

Figure 8 displays the energy efficiency improvement achieved by our technique, for all the applications, as compared to the other schemes. The energy efficiency values are measured as the inverse of energy-delay products (EDP) and normalized to the energy efficiency values of Razor. Among all the applications, gzip is observed to be most energy efficient with about 83% improvement in DCS over Razor. The anomaly

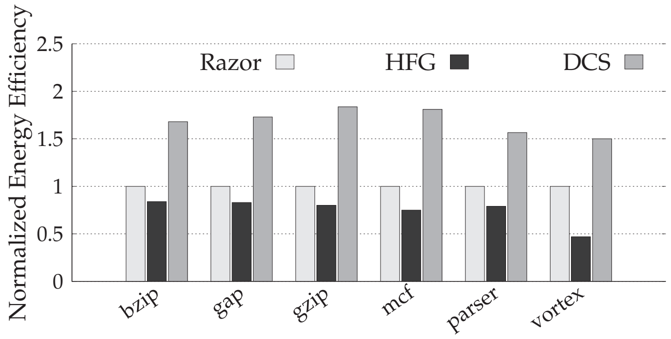

Figure 8: *Comparison of energy efficiency of the comparative schemes for different benchmarks (higher the better).*

in the relative performance and energy efficiency values of gzip and mcf is owing to the fact that total error count of gzip is lesser than mcf, but the number of unique error instances is more. The greater number of tuples reduce the scope of performance gain from DCS as compared to mcf. However, the overall execution time being lesser than mcf, the energy efficiency improvement is slightly higher. On an average, DCS exhibits about 60% improvement in energy efficiency over Razor and 90% improvement over HFG across all the benchmarks.

5.5 Overheads

The area and power overheads incurred by DCS are negligibly small. While the area and wire-length overheads are 0.23% and 0.77% of that of the entire processor pipeline, respectively, the power overhead is 0.85% of the core power. The power overhead results have been included in the energy efficiency results in Figure 8.

6. RELATED WORK

In the past few decades, researchers have extensively explored the NTC regime, especially in the context of processor architecture design. Kaul et al. present the opportunities and challenges of NTC design space, as opposed to STC [8]. However, NTC also offers a list of design bottlenecks. Karpuzcu et al. propose VARIUS-NTV, a model that explores this phenomenon [7].

The most conspicuous manifestation of process variation happens in the form of timing error. Agarwal et al. present a statistical timing analysis method to evaluate the effects of inter- and intra-die process variation [1].

Existing works aiming at mitigating timing errors primarily focus on timing speculation techniques. Ernst et al. propose Razor as a timing speculation based error recovery scheme [6]. Rahimi et al. propose Hierarchically Focussed Guardbanding (HFG), to adaptively mitigate timing errors caused by PVTA variations [13].

However, to the best of our knowledge, this paper is the first work to present choke points as a crucial threat in the NTC regime and to propose a dynamic error sensing and avoidance technique to mitigate the effects of choke points.

7. CONCLUSION

We present an analysis of choke points at NTC and experimentally prove its potency as a crucial threat to timing behavior of a system. We highlight the factors controlling the formation of choke points. To mitigate the timing violations caused as a result of choke points, we propose a novel sensing mechanism, called Dynamic Choke Sensing (DCS). This technique not only identifies the occurrences of timing errors caused by choke points, but also uses the knowledge to avoid them in future. In comparison to the other contemporary schemes like Razor and HFG, DCS demonstrates almost 30% and 100% improvement in performance, respectively.

8. ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation grants (CAREER-1253024, CCF-1318826, CNS-1421022, and CNS-1421068). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation

9. REFERENCES

[1] Agarwal, A. and others Statistical timing analysis for intra-die process variations with spatial correlations. In *Proc. of ICCAD* (2003), pp. 900–907.
[2] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. *Comp. Arch. News 13* (1970), 422–426.
[3] Choudhary, N. K. and others FabScalar: composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *Proc. of ISCA* (2011), pp. 11–22.
[4] De, V. Fine-Grain Power Management in Manycore Processor and System-on-Chip (SoC) Designs. In *Proc. of ICCAD* (2015), pp. 159–164.
[5] Dreslinski, R. G. and others Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits. *Proceedings of the IEEE 98*, 2 (2010), 253–266.
[6] Ernst, D. and others Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. In *Proc. of MICRO* (2003), pp. 7–18.
[7] Karpuzcu, U. R. and others VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *DSN* (2012), pp. 1–11.
[8] Kaul, H. and others Near-threshold voltage (NTV) design—Opportunities and challenges. In *Proc. of DAC* (June 2012), pp. 1149–1154.
[9] Lak, Z., and Nicolici, N. In-system and on-the-fly clock tuning mechanism to combat lifetime performance degradation. In *Proc. of ICCAD* (2011), pp. 434–441.
[10] NanGate. `http://www.nangate.com/?page_id=2328`.
[11] Paul, B. C. and others Impact of a process variation on nanowire and nanotube device performance. *T. Electron Devices 54*, 9 (2007), 2369.
[12] Pinckney, N. R. and others Assessing the performance limits of parallelized near-threshold computing. In *DAC* (2012), pp. 1147–1152.
[13] Rahimi, A. and others Hierarchically focused guardbanding: an adaptive approach to mitigate PVT variations and aging. In *Proc. of DATE* (2013), pp. 1695–1700.
[14] Sarangi, S. and others VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects. *IEEE Tran. on Semicond. Manufac. 21*, 1 (Feb 2008), 3–13.
[15] Seok, M. and others CAS-FEST 2010: Mitigating Variability in Near-Threshold Computing. In *J. Emerg Selec. Topics Cir. Sys* (May 2011).
[16] Shafique, M. and others The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives. In *Proc. of DAC* (2014), pp. 185:1–185:6.
[17] Sinha, S. and others Exploring sub-20nm FinFET design with predictive technology models. In *Proc. of DAC* (2012), pp. 283–288.
[18] Xin, J., and Joseph, R. Identifying and predicting timing-critical instructions to boost timing speculation. In *Proc. of MICRO* (2011), pp. 128–139.