8-2018

Tackling Choke Point Induced Performance Bottlenecks in a Near-Threshold GPGPU

Tahmoures Shabanian
Utah State University

TACKLING CHOKE POINT INDUCED PERFORMANCE BOTTLENECKS IN A NEAR-THRESHOLD GPGPU

by

Tahmoures Shabanian

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

Koushik Chakraborty, Ph.D. Major Professor	Sanghamitra Roy, Ph.D. Committee Member
Jacob Gunther, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2018

ABSTRACT

Tackling choke point induced performance bottlenecks in a near-threshold GPGPU

by

Tahmoures Shabanian, Master of Science

Utah State University, 2018

Major Professor: Koushik Chakraborty, Ph.D.
Department: Electrical and Computer Engineering

The proliferation of multicore devices with a stipulated thermal envelope has aided to the research in Near-Threshold Computing (NTC). Despite several reliability and vulnerability concerns, NTC operation of VLSI circuits are gaining tractions among researchers due to its inherent energy efficiency. However, operating a Graphics Processing Unit (GPU) at the NTC region has still remained recondite. In this work, an important reliability predicament of NTC is explored, called choke points, that severely throttles the performance of GPUs. Choke points are manifestations of process variation, altering the delays of sensitized logic gates in a fabricated chip. As a result, they can potentially create new critical paths that are virtually impossible to predict during the design of a chip. Upon uncovering the shortcomings of existing timing error mitigation techniques, a holistic circuit-architectural solution is propose, that promotes an energy-efficient NTC-GPU design by gracefully tackling the choke point induced timing errors. The proposed scheme offers 3.18× and 88.5% improvements in NTC-GPU performance and energy delay product, respectively, over a state-of-the-art timing error mitigation technique, with minimal area and power overheads.

(40 pages)

PUBLIC ABSTRACT

Tackling choke point induced performance bottlenecks in a near-threshold GPGPU

Tahmoures Shabanian

Over the last decade, General Purpose Graphics Processing Units (GPGPUs) have garnered a substantial attention in the research community due to their extensive thread-level parallelism. GPGPUs provide a remarkable performance improvement over Central Processing Units (CPUs), for highly parallel applications. However, GPGPUs typically achieve this extensive thread-level parallelism at the cost of a large power consumption. Consequently, Near-Threshold Computing (NTC) provides a promising opportunity for designing energy-efficient GPGPUs (NTC-GPUs). However, NTC-GPUs suffer from a crucial Process Variation (PV)-inflicted performance bottleneck, which is called *Choke Point*. Choke Point is defined as one or small group of gates which is affected by PV. Choke Point is capable of varying the path-delay of circuit and causing different forms of timing violation.

In this work, a cross-layer design technique is proposed to tackle the performance impediments caused by choke points in NTC-GPUs.

This thesis is dedicated to the most awesome people in my life. To my parents for all their love and support in every single stage of my life. To my brother Touraj Shabanian and my sister Tahmineh Shabanian, for their endless support and inspirations. I wouldn't have been able to get to this stage without them.

ACKNOWLEDGMENTS

I am glad to express my gratitude to all the people who have walked alongside me during the last three years, and helped me thorough this journey as a master student. I owe many thanks to my advisor Dr. Koushik Chakraborty and my co-advisor Dr. Sanghamitra Roy for their insightful guidance and financial support. They gave me the fantastic opportunity to join the BRIDGE lab, and to become familiar with different research areas. I would also like to thank Dr. Jacob Gunther for his valuable comments on my research as my committee member and for all his kind support and as the head of the ECE department at USU.

I extend my gratitude to the all the members of the BRIDGE lab, for their guidance, friendship and the stimulating discussions. They have all made me feel at home.I would specially like to thank Prabal Basu and Aatreyi Bal for their amazing help and support. It was a wonderful experience to have them both as co-authors in my paper. I thank all of my other friends and professors at USU, who have influenced my life and my thinking in remarkable ways. I am grateful to the ECE department and to all the staff members for making these years very enjoyable. I am thankful to Tricia Brandenburg for helping out through all of the administrative processes.

I thank my best friend Reza Tavakoli for his support in all the hard times that I have had. Without him, I would not be able to complete this journey. Thank you for being my rock.

Last, but not the least, I thank my wonderful, supportive parents and the best brother and sister in the world. Their endless love and support have sustained me throughout my life. I am forever indebted to them.

Tahmoures Shabanian

CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

ACRONYMS

CPU	Central Processing Unit
GPU	Graphic Processing Unit
GPGPU	General Purpose Graphic Processing Unit
VLSI	Very-Large-Scale Integration
VF	Voltage Frequency
RTL	Register-Transfer Level
NTC	Near Threshold Computing
STC	Super Threshold Computing
SIMD	Single Instruction Multiple Data
CU	Compute Unit
PV	Process Variation
CAM	Content Addressable Memory
UTD	Ultra Threaded Dispatcher
PTM	Predictive Technology Model
ChEST	Choke Error Sensing Table
CMU	Choke-error Monitor Unit
DUI	Decision Unit Interface
BLU	Black List Unit
EDP	Energy Delay Product

CHAPTER 1

INTRODUCTION

Evolution of Graphics Processing Units (GPUs) from specialized graphics chips to more generalized computing devices has ushered a new era in parallel computing. Over the past decade, researchers have made a significant progress in enhancing the computational power and memory bandwidth of General Purpose GPUs (GPGPUs) [1–4]. GPUs[1] offer a substantial performance improvement, over Central Processing Units (CPUs), for highly parallel applications. However, the extensive thread-level parallelism in GPUs are usually accompanied with a large power consumption [5]. To avoid hitting the power wall, while still maintaining the benefits of parallel computing, low-power GPUs are the need of the hour. Consequently, Near Threshold Computing (NTC) has emerged as a promising design paradigm for energy-efficient GPUs [6, 7]. In this paper, a significant reliability concern of the GPUs, operating at the NTC region (NTC-GPUs) is explored.

NTC-GPUs, besides offering a high energy-efficiency, suffer from a substantial process variation (PV) induced delay variation [8, 9]. Moreover, the vast spatial expanse of the GPUs, compared to CPUs, make them more susceptible to PV [3]. The problem is further complicated at NTC due to a manifold increase in the number of cores, compared to traditional GPUs, operating at super-threshold (STC) voltages [6]. This paper focuses on *Choke Points*—a crucial PV-inflicted performance bottleneck in NTC-GPUs.

A choke point is a small group of PV-affected logic gates in a circuit path, that dominates the path delay, so as to transform a short delay path into a critical path [10]. Choke points are created due to a drastic gate delay variation at NTC. Being an artifact of the fabrication process, the formation of choke points cannot be precisely anticipated or tackled at the design time [11]. As a result, a dynamic adaptive technique is required to tackle choke points in fabricated chips. Recently, Bal et al. have proposed an in-situ timing speculation

[1] In this paper, the terms GPU and GPGPU are used interchangeably.

technique to predict and recover from choke point induced timing errors in a simple out-of-order CPU pipeline at NTC [11]. This work shows, naively applying such an error recovery mechanism to a SIMD[2] pipeline does not boost the NTC performance to a similar extent as [11] (Section 6). Hence, employing a cross-layer methodology, this research proposes a choke point resilient NTC-GPU architecture, referred to as *Adaptive Choke Error-resilient GPU (ACE-GPU)*. Exploiting the recent history of timing errors in the GPU compute units (CUs), ACE-GPU employs a novel thread mapping strategy to minimize the occurrences of future timing errors, while incurring minimal overheads.

To the best of my knowledge, this is the first work that investigates the impacts of choke points in NTC-GPUs, as well as, proposes a timing error mitigation technique to improve the GPU power-performance.

The following are the key contributions of this work:

- This research elaborates why existing timing errors mitigation techniques are grossly ineffective in alleviating the choke point induced performance loss in an NTC-GPU (Section 3.2).
- This research explores the impacts of choke points in GPU pipelines, operating at STC and NTC regions (Section 3.4).
- This paper investigates the choke point induced performance penalties and energy cost, when a GPU operates with different number of CUs, at different operating conditions, with an existing error correction scheme (Section 3.4).
- The proposed NTC-GPU architecture—*Adaptive Choke Error-resilient GPU (ACE-GPU)*—efficiently tackles choke point induced timing errors in GPUs (Section 4.1).
- Using a range of GPGPU benchmarks, the performance benefit, energy efficiency and overheads of ACE-GPU are evaluated, with respect to traditional timing error mitigation techniques. (Section 6).

[2]GPUs are comprised of many single instruction multiple data (SIMD) units, executing several parallel threads simultaneously.

CHAPTER 2

RELATED WORK

Recent studies, related to this work can be broadly classified into three categories: (a) performance and reliability concerns at NTC, (b) tackling timing violations in modern microprocessors, and (c) performance and energy-efficiency of GPUs, operating at NTC.

In the first category, Pinckney et al. have evaluated the limitations of NTC in parallelized systems [12]. Dreslinski et al. have explored the bottlenecks, as well as opportunities, of the NTC operation [13]. Zhang et al. have characterized voltage noise in multicore NTC processors [14]. Gemmeke et al. have investigated the memory bottlenecks in NTC systems [15]. Karpuzcu et al. have explored the impacts of PV on NTC systems [16]. One of the remarkable reliability challenges, posed by PV, at NTC is choke points [10]. Bal et al. have explored this problem and proposed an adaptive technique, to mitigate the performance loss, arising from choke points, in unicore CPUs, operating at NTC [11]. Seo et al. have analyzed the impact of PV on near-threshold SIMD architectures [8]. Aguilera et al. have proposed workload partitioning, to alleviate the impact of PV in GPUs [3]. Miller et al. have analyzed the performance of low-voltage chips under the effect of PV. They have proposed two power supply rails to mitigate the effects of process variatoin on the performance of low-voltage chips [17]. Karpuzcu et al. have proposed a method with single supply voltage which relies on multiple frequency domains to increase the degree of freedom in tackling variations in NTC [18]. Maiti et al. have studied impacts of PV in NTC and have proposed PV aware power management method to mitigate the impacts of PV in NTC multicore systems [19].

In probing timing violations in microprocessors, Roy et al. have proposed a timing error prediction method, using the program counter values [20]. Rahimi et al. have proposed a hierarchically focused guardbanding technique to mitigate the impact of the process and environmental variations [21]. Ernst et al. have presented a dynamic timing error detection

and mitigation technique, using double sampling latches [22]. Ye et al. adopted an online clock skew tuning approach to speculate imminent timing errors [23].

Compared to the previous two categories, very little research has been invested into the third category, NTC-GPUs. Basu et al. have introduced a self-adaptive sprint technique, to mitigate PV induced performance variations in NTC-GPUs [6]. Pal et al. have recommended a dynamic allocation of thread blocks to mitigate the effects of PV on register access latency [7]. *However, to the best of my knowledge, this work is the first work that explore the impacts of choke points on the performance and energy-efficiency of NTC-GPUs.*

CHAPTER 3

MOTIVATION

In this chapter, choke point is illustrated as an important reliability problem in NTC-GPUs. First, the greater vulnerability of GPUs to choke point induced performance impediments, compared to traditional CPU cores, are discussed (Section 3.1). Second, the inefficacy of existing timing error mitigation techniques in alleviating the harmful impacts of choke points in GPUs are explained (Section 3.2). Third, the proposed cross-layer methodology is briefly described (Section 3.3). Finally, the delay variabilities in GPU pipelines is demonstrated, which is caused by choke points (Section 3.4), to motivate the design of ACE-GPU (Section 3.5).

3.1 Background

A *choke point* is defined as a single logic gate (or a group of logic gates) in a circuit that is affected by PV, and dominates the delay of the path in which it occurs. The presence of choke points can create new critical paths by increasing the delays of short delay paths, calculated at the design time. Consequently, choke points can give rise to unanticipated timing violations in the fabricated chips of a design [10].

The impact of PV is exacerbated in NTC circuits with respect to the STC ones [24]. As a result, NTC chips are significantly more susceptible to choke point induced variabilities. Furthermore, Aguilera et al. have demonstrated that GPUs are more impacted by within-die PV, compared to traditional multicore CPUs, due to a large number of cores in the former [3]. To exploit the large thread-level parallelism of the GPGPU applications, an NTC-GPU employs more cores than their STC counterparts [6]. Therefore, the probability of timing violations due to the formation of choke points notably increases in an NTC-GPU, thwarting the energy benefits of NTC operation.

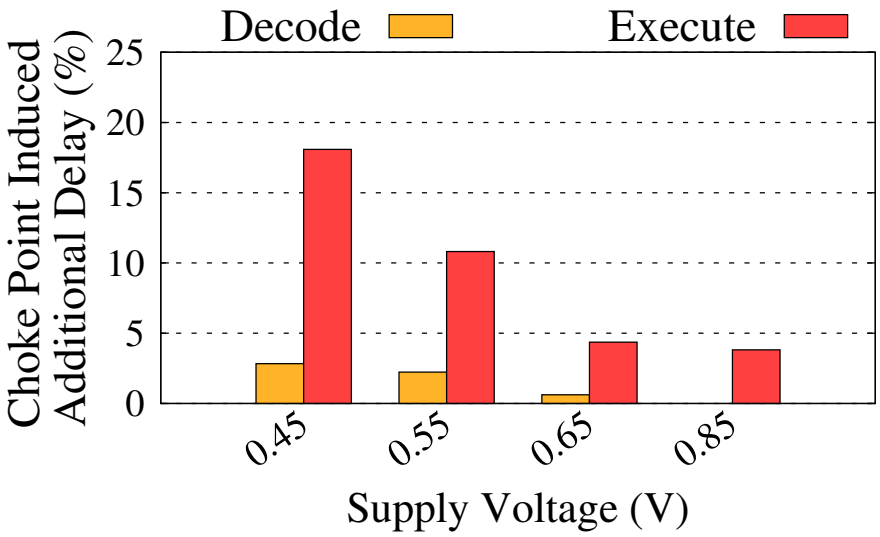


Fig. 3.1: Choke point induced additional delays. Shows that the presence of choke points increases the delays of the decode and execute stages, compared to the respective nominal delays of those stages.

3.2 Can Existing Timing Error Mitigation Techniques Tackle Choke Points in GPUs?

Researchers have proposed several circuit-architectural techniques to tackle timing errors in CPUs and GPUs. This section briefly discusses why such techniques are ineffective in combating choke points induced timing errors in an NTC-GPU.

- **Timing Guardbands:** To ensure a reliable operation, a chip can be conservatively run at a supply voltage, higher than the minimum required voltage, determined by static timing analysis. As a result, the system has enough timing slack to tolerate the worst-case process and environmental variations. However, the worst-case delay being a rare event, a conservative timing guardband drastically increases the power consumption of a system. Rahimi et al. have proposed hierarchically focused guardbanding that speculates the future timing behaviors of the circuit and adaptively adjusts the guardbands to guarantee timing requirements [21]. Despite being more effective than a conservative approach, [21] is extremely energy inefficient in tackling choke point induced timing errors in a unicore processor, operating at NTC [11]. Hence, employing an adaptive timing guardband strat-

egy is also futile in sustaining a reliable and efficient operation in multicore systems, like NTC-GPUs.

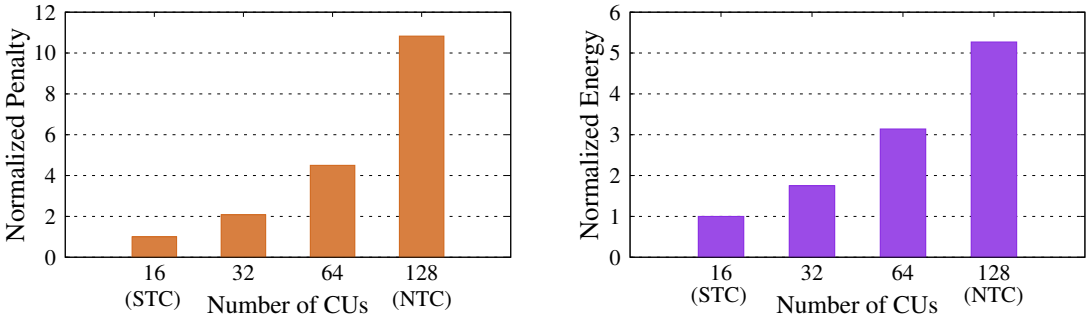
- **Dynamic Error Detection and Correction:** Deploying shadow latches to dynamically detect timing errors, and replaying the errant instructions, is one of the most popular and effective techniques to improve the timing resiliency of a system [22]. However, [22] cannot account for the PV-signature of a circuit which is crucial to effectively predict the recurring timing errors engendered by choke points [11]. Moreover, the large volume of the parallel threads and the prodigious spatial expanse of a GPU, further obscure the nuances of the PV-signatures. Hence, existing adaptive error detection and correction techniques are unlikely to efficiently deal with timing errors in an NTC-GPU.

- **Dynamic Choke Sensing:** Bal et al. have recently explored the effect of choke points in a simple out-of-order unicore system [11]. By uncovering a unique relation between the instruction metadata and the corresponding sensitized choke points, authors of [11] have proposed an adaptive timing error speculation and recovery scheme for scalar processors. However, a naive adoption of Bal's technique in vector processors like a GPU, leads to a significant performance loss (Section 6). This is due to the fact that a timing error in a single execution unit of a GPU, stalls all the SIMD lanes of a pipeline, thus effectively multiplying the error rate by the degree of parallelization [25]. As an NTC-GPU offers a significantly more parallelization (due to an increased number of CUs) than a traditional GPU, operating at STC conditions, deploying [11] in an NTC-GPU is a poor design choice.

Next, the proposed methodology is briefly discussed to demonstrate the choke points induced delay variabilities in an NTC-GPU.

3.3 Methodology

The MIAOW GPU RTL [26], which is modeled on the AMD's Southern Island GPU architecture, is used as the experimental platform. To synthesize the decode and execution units of a CU, the 15-nm FinFET library from NanGate is used [27]. In order to model PV



(a) Variation in performance penalty.

(b) Variation in energy consumption.

Fig. 3.2: Figure 3.2(a) and 3.2(b) exhibit the drastic increase in the performance penalties and energy consumptions respectively, for RecursiveGaussian, as more CUs are employed to cope with the decreasing VF levels towards NTC. For figure 3.2(a) and 3.2(b), the results are normalized to the corresponding STC values.

at STC and NTC, the VARIUS [28] and VARIUS-NTV [9], are used respectively. A home-brewed Statistical Timing Analysis tool is employed to study the delay of the sensitized paths of the decode and the execution unit. For a conservative estimate, PV-induced delays are considered in randomly chosen 1% of the gates of a circuit. A detailed methodology is discussed in Section 5.

3.4 Results

Figure 3.1 shows the percentage increase in the delays due to choke points, for decode and execution stages of a CU. The values are calculated with respect to the corresponding delays of an ideal case of no PV. Following are the three crucial observations from Figure 3.1.

1. The impact of choke points varies across different pipe stages. For example, at 0.45V, the decode stage exhibits an additional delay of ∼2.8%, while the execute stage has an additional delay of ∼18%. This variation is observed at other operating voltages too.

2. The choke point induced delays monotonically increase as the operating voltage approaches NTC values. For example, the additional delay of the execute stage at 0.45V

is ∼4.7× compared to that at 0.85V. This result confirms that the choke point induced delay variations are more aggressive at NTC, than at STC.

3. There is no choke point induced additional delay for the decode stage at 0.85V. This is because, choke points may not always give rise to new critical paths in a fabricated circuit.

An increase in the number of CUs can improve the NTC performance by effectively exploiting the thread-level parallelism of the GPGPU benchmarks [6]. However, timing errors due to choke points can introduce a staggering performance penalty in an error correction scheme, severely throttling the performance of NTC. Figure 3.2(a) shows the variation in the performance penalties, as *RecursiveGaussian*, a GPGPU benchmark, executes with different number of CUs (with 100% utilization), endowed with *Razor* as the timing error mitigation scheme. The leftmost and the rightmost bars in Figure 3.2(a) represent STC and NTC operating conditions, respectively. The results are normalized to the penalty for 16 CUs. It is noticed that an 8× increase in the number of CUs, increases the performance penalty by ∼8×. This result reveals the inefficacy of Razor [22] in tackling aggravated choke point induced timing errors at NTC.

For the same experiment, Figure 3.2(b) demonstrates a ∼5.5× increase in the energy consumption at NTC (128 CUs), with respect to STC (16 CUs). A large fraction of this high energy consumption at NTC comes from leakage energy. As leakage energy is proportional to the application execution time, the high performance penalties at NTC (Figure 3.2(a)) diminishes the energy-efficiency benefit of an NTC-GPU.

3.5 Significance

The motivational results demonstrate a tremendous impact of choke points on the performance as well as energy consumption of the NTC-GPUs. Hence, in order to efficiently exploit the vast SIMD resources of an NTC-GPU, it is needed to explore a design paradigm that can dynamically predict and avoid imminent timing errors in the CUs, while incurring

minimal performance and power overheads. Next, such a novel GPU design paradigm is explored, that reclaims the energy-efficiency advantage of NTC.

CHAPTER 4

DESIGN PARADIGM

4.1 Adaptive Choke Error-resilient GPU (ACE-GPU)

In this chapter, ACE-GPU a novel design paradigm to tackle choke point induced timing errors in NTC-GPUs is discussed. The design overview is presented in Section 4.2, and the design components are elaborated in Section 4.2.1.

4.2 Design Overview

Figure 4.1 portrays the conceptual overview of ACE-GPU. The Ultra-Threaded Dispatcher (UTD), an integral component of a GPU, is responsible for assigning the thread blocks to the CUs [29]. In ACE-GPU, the baseline thread block assignment policy of the UTD is altered, to a choke point aware assignment strategy. This approach is fundamentally different than stalling the thread block execution to avoid imminent timing errors [11]. The baseline GPU architecture is augmented with a *Choke-error Monitor Unit (CMU)* and a *Black List Unit (BLU)*. CMU is responsible for the identification, correction and prediction of choke point induced timing errors. The primary components of CMU are *Choke Error Sensing Table (ChEST)*, *Decision Unit Interface (DUI)* and *Threshold Comparator*. On the other hand, BLU works in tandem with CMU and the Ultra-Threaded Dispatcher (UTD), to detect and avoid the CUs that are severely impaired by choke point induced timing errors. Different components of CMU and BLU, along with their respective working principles are described next.

4.2.1 ACE-GPU Components

CMU and BLU dynamically monitor the timing errors occurring in different CUs, and communicate with the UTD to make an efficient thread block assignment in order to improve the NTC-GPU performance.

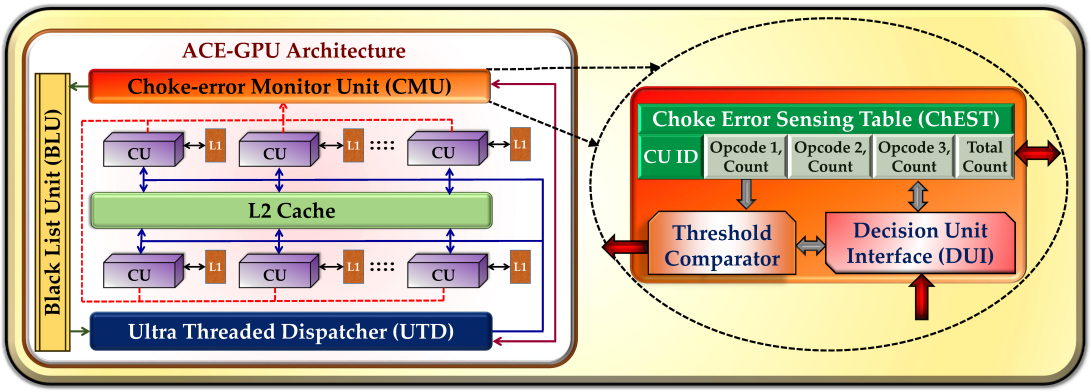


Fig. 4.1: The CMU and BLU form the backbone of the ACE-GPU architecture. The CMU performs the error management, including error logging and avoidance. The magnified version of the CMU shows its components. The Choke Error Sensing Table (ChEST) contains the error logs for the previously encountered error instances, and facilitates error avoidance. The Decision User Interface (DUI) is the brain of the CMU making all the error management decisions. The Threshold Comparator marks the CUs that are severely affected by choke points and logs them in the Black List Unit (BLU). The CUs listed in the BLU are not considered for imminent thread block assignment by the Ultra-Threaded Dispatcher (UTD).

Choke-error Monitor Unit (CMU)

The CMU is the central component of the ACE-GPU architecture. It spearheads two main tasks of error management. First, it receives timing error details from each of the CUs, and records them for future references. Second, it populates the BLU with the IDs of the CUs that are rendered unreliable due to the severe impact of choke points. The role of each of the constituent components in the entire design flow is explained next.

- **Choke Error Sensing Table (ChEST):** The ChEST records the errors occurring in each CU, in terms of tuples of errant opcodes. In the implementation, each tuple corresponds to a CU ID, three errant opcodes encountered by that CU, and their number of occurrences (i.e., the number of times that opcode is executed on the CU, not the number of timing errors it caused). Additionally, each tuple has the total error count, which records the number of the timing errors detected in that CU.

  The ChEST is implemented as a Content Addressable Memory (CAM), to facilitate the table lookup. The number of entries in the ChEST is a trade-off between the associated

Algorithm 1 Working Principle of DUI

```
1:  procedure DUI(timing_info[IDs][opcodes])
2:      for all IDs ∈ timing_info do
3:          if (ID ∈ ChEST) then
4:              if (timing_info[ID][opcode] ∈ ChEST[ID]) then
5:                  ChEST[ID][opcode].count++
6:              else
7:                  if (opcode_field.full()) then
8:                      opcode_field.evict(least_occurring_opcode)
9:                  end if
10:                 opcode_field.insert(opcode)
11:             end if
12:         else
13:             ChEST.evictPseudoLRU(ID)
14:             ChEST[ID][opcode].count = 1
15:             ChEST[ID].error_count++
16:         end if
17:         PDCP(ID)
18:     end for
19: end procedure
```

overhead and the accuracy of sensing recurrent timing errors. A large entry size ensures a relatively high sensing accuracy, at the cost of relatively high area and power overheads. On the other hand, a smaller ChEST reduces the sensing accuracy thereby increasing the incurred penalty cycles. In the evaluations, the entry size for ChEST is considered to be 10 (for 128 CUs). The UTD looks up the ChEST before assigning thread blocks to CUs, in order to avoid imminent recurrent timing errors.

- **Decision Unit Interface (DUI):** The DUI manages the error sensing, as well as, error correction inside the CMU. Algorithm 1 shows the working principle of the DUI. Upon receiving timing error information from the CUs, it updates the record of the encountered errors in the ChEST. The errors are detected with the help of double-sampling latches [22].

The DUI logs the errant opcodes, along with the corresponding CU ID, in the ChEST. If the ID is already listed in the ChEST, the DUI logs the opcode in the corresponding tuple of the ChEST. In case the tuple is already full, a new errant opcode replaces an existing opcode with least number of occurrences. Simultaneously, the DUI updates the total error count of the corresponding CU. If the ChEST becomes full, a new CU ID can replace an existing CU in the ChEST using a pseudo-LRU policy. Once an error is detected at the CU, the DUI stalls the current execution of the CU, and re-assigns the thread block to ensure an error-free execution (Section 4.2.1).

To adapt to high-error situations, the DUI employs a *Performance Degradation Control Procedure (PDCP)*, described in Algorithm 2. If the number of CUs in the BLU exceeds 10% of total CUs, the DUI increases the threshold value, considered by the Threshold Comparator. After that, it flushes the existing entries of the BLU, and stores the current threshold for which, the BLU is flushed. This stored threshold value is required to calculate the total number of timing errors, lest the flushed CUs once again appear in the ChEST. If the current threshold is beyond a preset maximum, it is no longer incremented. In that case, a CU in the ChEST, with the total error being more than the current threshold, can replace one of the existing CUs in the BLU, chosen randomly. However, this situation in the simulations have not been encountered.

- **Threshold Comparator:** The Threshold Comparator continuously monitors the total error counts of all the CUs in the ChEST. As soon as the total error count of a CU exceeds the threshold pre-defined in the comparator, the CU ID is removed from the ChEST, and added to the BLU. In the implementation the threshold is updated dynamically, if a certain percentage of the total CUs are blacklisted. However, the threshold value cannot be increased beyond a preset upper limit.

Black List Unit (BLU)

The BLU records the CU IDs whose total error count exceeds a given threshold of errors. Any CU ID listed in the BLU is not considered for thread block assignment. The

Algorithm 2 Performance Degradation Control Procedure

```
1: procedure PDCP(ID)
2:     error_count = ChEST[ID].error_count + 1
3:     if (BLU[ID].tag == 0) then
4:         error_count += BLU[ID].old_error_threshold
5:     end if
6:     if (error_count > error_threshold) then
7:         if (BLU.size() == 0.1 * total_num_CU) then
8:             if (error_threshold < max_error_threshold) then
9:                 for IDs ∈ BLU do
10:                    if (BLU[ID].tag == 1) then
11:                        BLU[ID].tag = 0
12:                        BLU[ID].old_error_threshold =
       error_threshold
13:                    end if
14:                end for
15:                error_threshold += delta_threshold
16:            else
17:                BLU.randomReplaceCU(ID)
18:            end if
19:        end if
20:        BLU[ID].tag = 1
21:        ChEST.evict(ID)
22:    end if
23: end procedure
```

UTD looks up the BLU, before assigning a thread block to a CU. The size of the BLU is fixed at 10% (empirically determined) of the total CU count. If the number of blacklisted CUs exceeds the BLU size, the BLU is flushed by setting the valid tags of all of the CU IDs to zero, and the threshold is increased in the comparator. Like the ChEST, the BLU is also implemented as a CAM.

Ultra-Threaded Dispatcher (UTD)

The baseline UTD is modified to work harmoniously with the CMU and the BLU. Algorithm 3 displays the remodeled UTD assignment. First, each thread block is considered to be composed of SIMD threads of only one type of opcode [30]. Second, before assigning a thread block to a CU, the UTD looks up the BLU. A CU, listed in the BLU, is discarded by the UTD. Next, it searches for the CU in the ChEST entries. Upon getting a match, the CU is considered for the thread block assignment, only if the opcode, corresponding to the thread block under consideration, is not present in that CU's tuple in the ChEST. If this condition is not met, the UTD then randomly chooses one of the remaining CUs in the pool. When a thread block encounters a timing error in a CU, the UTD reschedules the errant thread block based on Algorithm 3. Rescheduling a thread block, although infrequent, incurs a performance penalty that is dictated by the number of pipeline stages in the GPU.

The performance and hardware overheads are considered from the components of ACE-GPU in the evaluations (Section 6).

Algorithm 3 UTD Assignment

```
1:  procedure UTD(BLU, ChEST)
2:      ID_list = [1,...,128]
3:      for ID ∈ ID_list do
4:          opcode = opcode_to_be_assigned
5:          if ID ∈ BLU then
6:              next
7:          end if
8:          if ID ∈ ChEST && opcode ∈ ChEST[ID] then
9:              next
10:         end if
11:         ID.assign(opcode)
12:         break
13:     end for
14: end procedure
```

CHAPTER 5

METHODOLOGY

In this section, the comprehensive cross-layer methodology used to implement as well as evaluate the potency of the proposed design, is discussed. Figure 5.1 portrays the proposed methodology. The details of each layer are elaborated next.

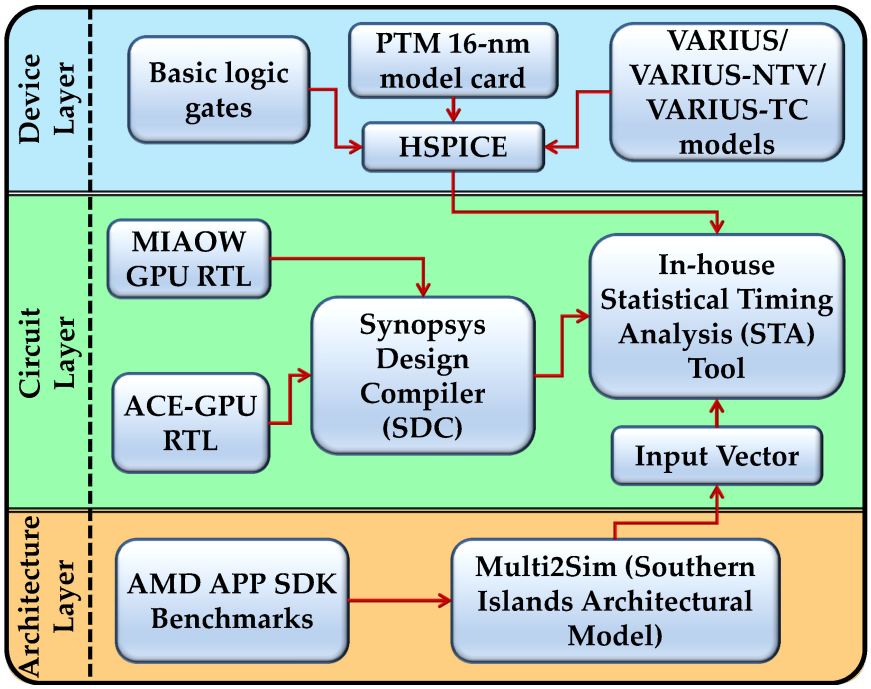


Fig. 5.1: Cross-layer methodology for ACE-GPU.

5.1 Device Layer

The HSPICE models of the basic logic gates (NAND, NOR, Inverter) are simulated to estimate their delay distributions, in the presence of PV, at different supply voltages. For these simulations the 16-nm Predictive Technology Models (PTM) is used. To estimate within-die PV, the VARIUS [28] and VARIUS-NTV [9] models for STC and NTC, respec-

tively are considered. Further, the VARIUS-TC model is utilized to incorporate the FinFET characteristics [**?**]. Monte Carlo simulations are performed to gauge the propagation delay variations for 10,000 instances of each logic gate. The delay values are used in the circuit layer (Section 5.2) to realize choke points in the circuit.

5.2 Circuit Layer

In this layer, two principle tasks are considered. First, an open-source reference GPU RTL [26] is augmented to implement the components of the proposed ACE-GPU architecture. The reference and augmented GPU RTLs are synthesized by using Synopsys Design compiler (SDC) [31], to estimate the area and power overheads associated with the proposed scheme. The VF values considered for synthesis at STC and NTC are (0.85V, 900MHz) and (0.45V, 400MHz), respectively. Second, the synthesized netlists and the input vectors (Section 5.3), are fed into an in-house Statistical Timing Analysis (STA) tool. The STA tool contains a library of the delay distributions for the basic logic gates at different operating voltages, obtained from HSPICE simulations (described in Section 5.1). The STA tool performs a timing analysis of the sensitized paths in the circuit netlist, for the given input vectors. Consequently, a clear idea of the impact of choke points on the path delays of a fabricated chip at the runtime is obtained. The resultant delay reports are used to evaluate the efficacy of the comparative schemes. (Section 6.1).

Parameters	**Configurations**
No. of CUs	128
Supply Voltage	0.35 V
CU Frequency	400 MHz
L2 Cache	8×768 KB, latency: 20 ns
Global Memory	B/W: 264 GB/s, latency: ∼300 ns

Table 5.1: GPU configurations.

5.3 Architecture Layer

An AMD Southern Island GPU is modeled on Multi2Sim architectural simulator [29]. The GPU architectural parameters considered in this work are listed in Table 5.1. The Multi2Sim codebase are instrumented to automatically extract the cycle-wise instruction metadata (viz., opcodes and operands) from an execution unit, while running the GPGPU benchmarks from AMD's APP SDK suite [32]. These metadata serves as the input vectors to the STA tool for performing a dynamic path sensitization analysis and evaluating the impacts of choke points (Section 5.2).

CHAPTER 6

EXPERIMENTAL RESULTS

In this chapter, the performance and energy-efficiency benefits of ACE-GPU are compared, with respect to other timing error recovery schemes in GPU. Section 6.1 discusses the considered comparative schemes, while Section 6.3 and 6.4 present the relative performances and energy efficiencies of the schemes, respectively, across 8 GPGPU benchmarks. Section 6.5 presents the hardware overheads of ACE-GPU.

6.1 Comparative Schemes

- **Razor:** This is a popular timing speculation technique that sporadically trims the timing guardband to allow intermittent timing errors in the pipelines [22]. The errors are detected by employing double-sampling latches at the pipeline boundaries. A thread-block reassignment is triggered to correct the timing errors.

- **Dynamic Choke Sensing (DCS):** This scheme offers detection, correction, as well as prediction of choke point induced timing errors [11]. Originally proposed for a scalar CPU pipeline, DCS is implemented for vector processors like GPUs. DCS employs a lookup table, in the form of a RAM, to store and lookup recurrent timing errors. A choke controller is used to manage the error detection, correction and prediction. For the error avoidance, the choke controller inserts a single stall cycle in the pipeline to allow the errant opcode to finish its execution without any error. The proposed scheme (ACE-GPU) is fundamentally different than DCS, because, instead of stalling the CUs for one complete cycle, ACE-GPU employs an efficient thread block mapping strategy for an error-free execution.

- **Adaptive Choke Error-resilient GPU (ACE):** This is the proposed scheme that also employs double-sampling latches to detect timing errors. ACE is designed to mitigate

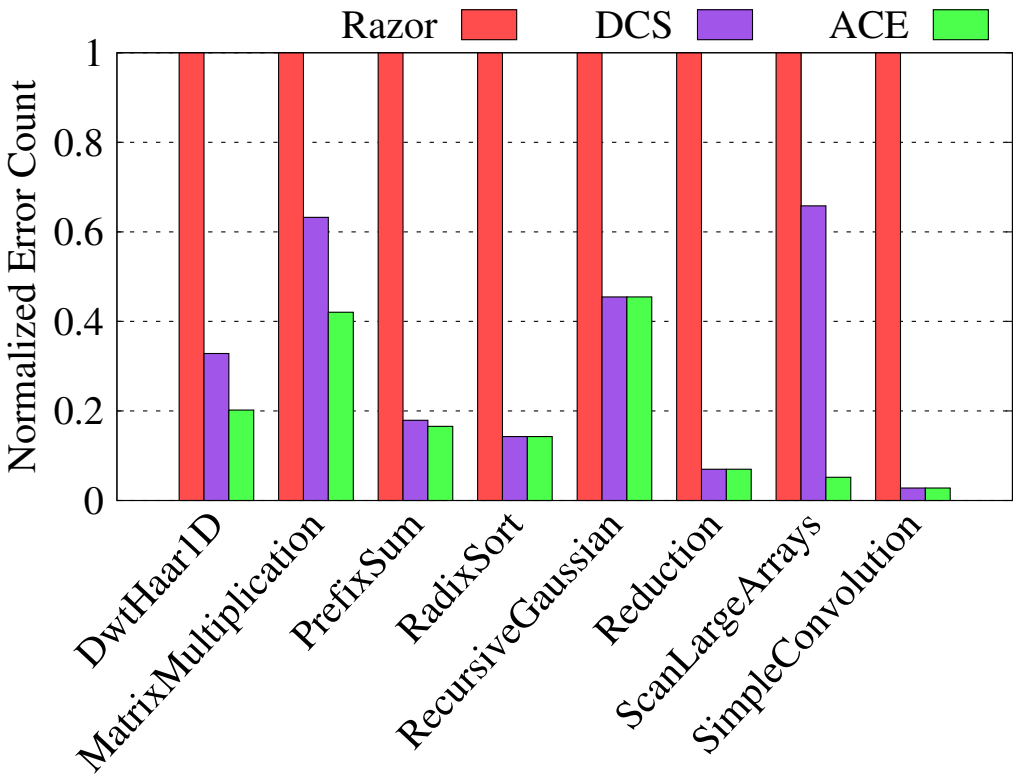


Fig. 6.1: Error comparison (lower is better).

choke point induced timing errors in NTC-GPUs. The design of ACE is described in Section 4.1.

6.2 Error Comparison

Figure 6.1 demonstrates the relative number of timing errors that each benchmark encounters under different comparative schemes. The Y-axis values are normalized with respect to the number of timing errors with Razor [22]. The efficient error prediction and thread block assignment policy in ACE, lead to significantly lower number of timing error events, compared to Razor and DCS, across all the benchmarks. The large difference in the number of timing errors between DCS and ACE for some benchmarks (e.g., $\sim$12.6$\times$ in *ScanLargeArrays*), is due to the more efficient topology of ChEST in ACE, compared to the LUT in DCS.

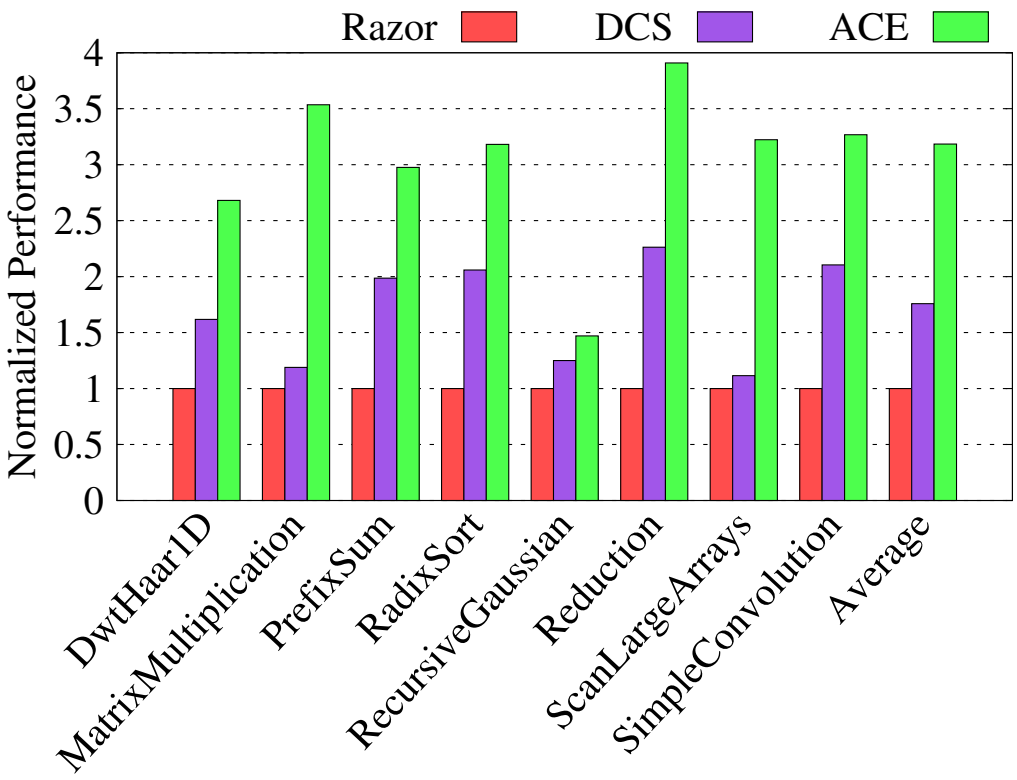


Fig. 6.2: Performance comparison (higher is better).

6.3 Performance Comparison

Figure 6.2 illustrates the relative performances of the comparative schemes (Section 6.1). The results are normalized to the performance of Razor [22]. It is noticed that, on an average, ACE performs 3.18× better than Razor, across all the GPGPU benchmarks. DCS has a better performance than Razor, as the latter encounters significantly more timing errors due to a lack of error prediction mechanism, incurring severe penalties. On the other hand, ACE performs remarkably better (1.81×, on an average) than DCS. This is because: (a) ACE employs an efficient thread block to CU mapping strategy, obviating the need for a single stall cycle to avoid timing errors. (b) The BLU in ACE prevents assigning thread blocks to CUs that are severely impaired by choke points. For example, in *MatrixMultiplication*, many CUs are highly affected by choke points and hence, get listed in the BLU. As UTD avoids those CUs, listed in the BLU, for immediate thread block assignments, ACE saves an appreciable performance loss. However, DCS, lacking a

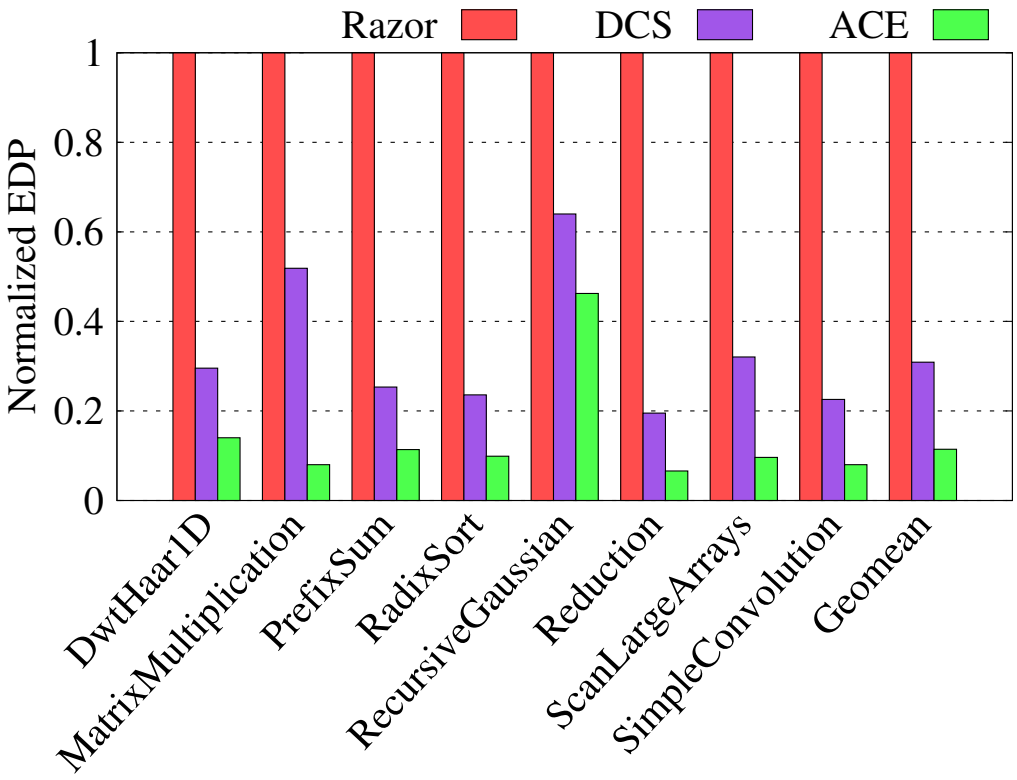


Fig. 6.3: EDP comparison (lower is better).

tailored hardware unit like the BLU, is more susceptible to choke point induced performance penalties. (c) Unlike a RAM-based lookup table in DCS, ACE uses a CAM-based one (ChEST) to record the history of recent timing errors, thus resulting in a much faster lookup.

6.4 Energy-Efficiency Comparison

The energy-efficiencies of the schemes are measured by using the energy delay product (EDP) metric. Figure 6.3 shows the EDPs of the schemes are normalized to the EDP of the Razor. The proposed scheme ACE is the most energy-efficient, offering an average of ∼88.5% improvement in EDP over Razor. It is imperative to note that the relative power footprint of the hardware for ACE, is one order of magnitude more than those of the DCS and Razor. However, this high power footprint is amortized by an appreciable performance gain in ACE. Moreover, at NTC, a significant fraction of the total energy is contributed by

the leakage energy, which is proportional to the application execution time. As the choke point induced timing penalties is the least in ACE (Section 6.3), it dramatically improves the leakage energy consumption of the NTC-GPU, for all the benchmarks. These performance and energy benefits aid the improvement in EDP, making ACE a very energy-efficient GPU design paradigm at NTC.

6.5 Hardware Overheads

The area and power overheads of ACE-GPU, obtained from synthesis at the NTC operating conditions (Table 5.1), are 1.42% and 6.17%, respectively. The overheads are calculated compared to the respective values of a baseline CU, with no timing error mitigation scheme. The power overheads are considered in the evaluation of EDP.

CHAPTER 7

CONCLUSION

Choke points pose a tremendous threat to meeting timing constraints in NTC circuits. Hence, tackling choke point induced timing errors in an NTC-GPU is crucial in order to sustain a reliable and energy-efficient operation. In this work, the significant delay variabilities of different pipelines of an NTC-GPU have been demonstrated, as well as, severe performance loss, engendered by the formation of choke points. By uncovering the limitations of existing timing speculation and error recovery techniques, ACE-GPU—a novel NTC-GPU design paradigm is proposed which combats choke point induced performance bottlenecks. Using a cross-layer methodology, it is shows that ACE-GPU offers 3.18× and 88.5% improvements in performance and EDP respectively, over a Razor-based timing error detection and correction scheme, across 8 GPGPU benchmarks, while incurring minimal hardware overheads.

REFERENCES

[1] B. Wang, B. Wu, D. Li, X. Shen, W. Yu, Y. Jiao, and J. S. Vetter, "Exploring hybrid memory for gpu energy efficiency through software-hardware co-design," 2013, pp. 93–102.

[2] P. Aguilera, K. Morrow, and N. S. Kim, "Qos-aware dynamic resource allocation for spatial-multitasking gpus," 2014, pp. 726–731.

[3] P. Aguilera, J. Lee, A. F. Farahani, K. Morrow, M. J. Schulte, and N. S. Kim, "Process variation-aware workload partitioning algorithms for gpus supporting spatial-multitasking," 2014, pp. 1–6.

[4] S. Saha, P. Basu, C. Rajamanikkam, A. Bal, K. Chakraborty, and S. Roy, "SSAGA: sms synthesized for asymmetric GPGPU applications," *ACM Trans. Design Autom. Electr. Syst.*, vol. 22, no. 3, pp. 49:1–49:20, 2017.

[5] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a single chip causes massive power bills gpusimpow: A gpgpu power simulator," April 2013.

[6] P. Basu, H. Chen, S. Saha, K. Chakraborty, and S. Roy, "Swiftgpu: Fostering energy efficiency in a near-threshold gpu through tactical performance boost," 2016.

[7] A. Pal, A. Bal, K. Chakraborty, and S. Roy, "Split latency allocator: Process variation-aware register access latency boost in a near-threshold graphics processing unit," *J. Low Power Electronics*, vol. 13, no. 3, pp. 419–427, 2017.

[8] S. Seo, R. G. Dreslinski, M. Woh, Y. Park, C. Chakrabarti, S. A. Mahlke, D. Blaauw, and T. N. Mudge, "Process variation in near-threshold wide simd architectures," 2012, pp. 980–987.

[9] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas, "Varius-ntv: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages," 2012, pp. 1–11.

[10] V. De, "Fine-grain power management in manycore processor and system-on-chip (soc) designs," 2015, pp. 159–164.

[11] A. Bal, S. Saha, S. Roy, and K. Chakraborty, "Revamping timing error resilience to tackle choke points at ntc systems," 2017, pp. 1020–1025.

[12] N. R. Pinckney, K. Sewell, R. G. Dreslinski, D. Fick, T. N. Mudge, D. Sylvester, and D. Blaauw, "Assessing the performance limits of parallelized near-threshold computing," in *DAC*, 2012, pp. 1147–1152.

[13] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. N. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proc. of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[14] X. Zhang, T. Tong, S. Kanev, S. K. Lee, G. Wei, and D. M. Brooks, "Characterizing and evaluating voltage noise in multi-core near-threshold processors," 2013, pp. 82–87.

[15] T. Gemmeke, M. M. Sabry, J. Stuijt, P. Raghavan, F. Catthoor, and D. Atienza, "Resolving the memory bottleneck for single supply near-threshold computing," 2014, pp. 1–6.

[16] U. R. Karpuzcu, N. S. Kim, and J. Torrellas, "Coping with parametric variation at near-threshold voltages," vol. 33, no. 4, pp. 6–14, 2013.

[17] T. N. Miller, X. Pan, R. Thomas, N. Sedaghati, and R. Teodorescu, "Booster: Reactive core acceleration for mitigating the effects of process variation and application imbalance in low-voltage chips," in *HPCA*, 2012, pp. 1–12.

[18] U. R. Karpuzcu, A. A. Sinkar, N. S. Kim, and J. Torrellas, "Energysmart: Toward energy-efficient manycores for near-threshold computing," 2013, pp. 542–553.

[19] S. Maiti, N. Kapadia, and S. Pasricha, "Process variation aware dynamic power management in multicore systems with extended range voltage/frequency scaling," in *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2015, pp. 1–4.

[20] S. Roy and K. Chakraborty, "Predicting timing violations through instruction level path sensitization analysis," 2012, pp. 1074–1081.

[21] A. Rahimi, L. Benini, and R. K. Gupta, "Hierarchically focused guardbanding: an adaptive approach to mitigate PVT variations and aging," 2013, pp. 1695–1700.

[22] D. Ernst, N. S. Kim, S. Das, S. Pant, R. R. Rao, T. Pham, C. H. Ziesler, D. Blaauw, T. M. Austin, K. Flautner, and T. N. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," 2003, pp. 7–18.

[23] R. Ye, F. Yuan, and Q. Xu, "Online clock skew tuning for timing speculation," 2011, pp. 442–447.

[24] L. Chang, D. J. Frank, R. K. Montoye, S. J. Koester, B. L. Ji, P. W. Coteus, R. H. Dennard, and W. Haensch, "Practical strategies for power-efficient computing technologies," vol. 98, no. 2, pp. 215–236, 2010.

[25] E. Krimer, P. Chiang, and M. Erez, "Lane decoupling for improving the timing-error resiliency of wide-simd architectures," 2012, pp. 237–248.

[26] *MIAOW GPU* - An open source RTL implementation of a GPGPU, 2015. [Online]. Available: http://miaowgpu.org.

[27] NanGate, http://www.nangate.com/?page_id=2328.

[28] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Varius:a model of process variation and resulting timing errors for microarchitects," vol. 21, pp. 3 –13, 2008.

[29] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, " Multi2Sim: A Simulation Framework for CPU-GPU Computing ," Sep. 2012.

[30] S. Lee and C. Wu, "Caws: criticality-aware warp scheduling for gpgpu workloads," 2014, pp. 175–186.

[31] D. Compiler, R. User, and M. Guide, "Synopsys," *Inc., see http://www. synopsys. com*, 2001.

[32] " AMD Accelerated Parallel Processing (APP) Software Development Kit ," 2016. [Online]. Available: http://developer.amd.com/sdks/amdappsdk/

CURRICULUM VITAE

Tahmoures Shabanian

Published Conference Papers

- ACE-GPU: Tackling Choke Point Induced Performance Bottlenecks in a Near-Threshold Computing GPU, Tahmoures Shabanian, Aatreyi Bal, Prabal Basu, Koushik Chakraborty, Sanghamitra Roy, in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2018.