8-2019

Predicting Critical Warps in Near-Threshold GPGPU Applications Using a Dynamic Choke Point Analysis

Sourav Sanyal
Utah State University

PREDICTING CRITICAL WARPS IN NEAR-THRESHOLD GPGPU APPLICATIONS USING A DYNAMIC CHOKE POINT ANALYSIS

by

Sourav Sanyal

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

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

Sanghamitra Roy, Ph.D. Major Professor	Koushik Chakraborty, Ph.D. Committee Member
Reyhan Baktur, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2019

ABSTRACT

Predicting Critical Warps in Near-Threshold GPGPU Applications using a Dynamic Choke Point Analysis

by

Sourav Sanyal, Master of Science

Utah State University, 2019

Major Professor: Sanghamitra Roy, Ph.D.
Department: Electrical and Computer Engineering

General-purpose graphics processing units (GPGPU), owing to their enormous parallelism have found ubiquitous applications in parallel computing. Along with an unprecedented performance offered by every new generation of the GPGPUs, their peak power rating has also increased over the years. As an inevitable consequence, Near-Threshold Computing (NTC) has come to the rescue, offering a substantially lower demand on the power supply unit, while striving to achieve a super-threshold performance, by exploiting a higher parallelism. However, a severe device-level delay variability arising from process variation (PV), can significantly diminish the NTC system performance. In this work, choke points–a unique device-level characteristic of PV at NTC–that can exacerbate the delays of the GPGPU parallel warps have been explored. In order to improve the NTC GPU performance, a family of holistic circuit-architectural solutions, referred to as Choke Point Aware Warp Speculator (CPAWS) has been proposed. CPAWS identifies the choke point induced critical warps in GPGPU applications, and improves their execution latencies in their respective execution units. Compared to a state-of-the-art warp scheduling policy, the best scheme improves the performance and energy-efficiency of an NTC GPU by ∼39% and ∼31%, respectively. while incurring marginal hardware overheads.

(48 pages)

PUBLIC ABSTRACT

Predicting Critical Warps in Near-Threshold GPGPU Applications using a Dynamic Choke Point Analysis

Sourav Sanyal

General purpose graphics processing units (GP-GPU), owing to their enormous thread-level parallelism, can significantly improve the power consumption at the near-threshold (NTC) operating region, while offering close to a super-threshold performance. However, process variation (PV) can drastically reduce the GPU performance at NTC. In this work, choke points—a unique device-level characteristic of PV at NTC—that can exacerbate the warp criticality problem in GPUs have been explored. It is shown that the modern warp schedulers cannot tackle the choke point induced critical warps in an NTC GPU. Additionally, *Choke Point Aware Warp Speculator*, a circuit-architectural solution is proposed to dynamically predict the critical warps in GPUs, and accelerate them in their respective execution units. The best scheme achieves an average improvement of ∼39% in performance, and ∼31% in energy-efficiency, over one state-of-the-art warp scheduler, across 15 GPGPU applications, while incurring marginal hardware overheads.

And those who were seen dancing were thought to be insane by those who could not hear the music. To all those souls....

ACKNOWLEDGMENTS

First and foremost I would like to thank my father Dr. Dipankar Sanyal who has been my biggest motivation since childhood and who also happens to be my best critic. I would also like to thank my mother Sonali Sanyal who has raised me and groomed me in the most caring way possible. She has always been my biggest support system and has always encouraged me in almost all my endeavours. I am greatly indebted to my grandmother, Late Deepa Sanyal for being my childhood best friend and for imparting in me the much treasured life lessons.

I express my sincerest gratitude to my primary advisor Dr. Sanghamitra Roy for giving me the opportunity to work as a graduate student in USU BRIDGE LAB. She is one of the nicest persons I have ever encountered in my life. I would like to thank her for the weekly meetings that took place in Spring 2018, where we would discuss the progresses of my research work. She would almost always point out any mistakes that would be made by me and give exceptional insights. I am greatly thankful to Dr. Koushik Chakraborty. Coming from a different background, I got to learn so much about Computer Architecture by taking his courses. His interactive way of teaching is really stimulating for the mind. I also learned how to critique research papers and formulate research problems by staying under his tutelage. His reprimands and berating have always nurtured me in the best possible way to become a better version of myself. Outside work, both Dr Roy and Dr Chakraborty have also been extremely loving and caring guardian-like figures, inviting the lab members to their home and throwing parties with delicious foods and hearty conversations. I will never forget performing on Durga Puja and Diwali, where Dr Roy had gracefully sung melodious Indian songs, while I had accompanied her by strumming my guitar. I am eternally grateful to both my advisors because of whom my two years in Logan have been the perfect balance between work and play.

I would like to thank Prabal Basu, my dear friend and roommate who has also been my role model ever since I came to Logan. He has had a huge contribution in my research. His

apt questioning and incisive timely review have been the biggest catalysts for my reseach venture. The serious and sometimes lighthearted conversations with him over dinner every night has shaped my conscience in ways beyond expressing. I am thankful to Tarak Saha for him being the fatherly figure away from home, always helping me in daily chores and giving me the right advice whenever I needed. I am forever obligated to Pramesh Pandey for teaching me to drive in the summer of 2018 and also for being my closest confidante and brother in USU. I am also lucky to have Padma Ghimire as a friend who has never let me leave her home without feeding me at least one of the many brilliant dishes she can cook. I have been fortunate to experience the company of Chidhambaranathan Rajamanikkam, who has always been an immensely supportive friend and guide and for sharing the mutual interest to try out cooking various kinds of experimental food. I would like to thank Anindya Bagchi for taking me to various hikes and excursion activities in the scenic national parks of Utah and Colorado. I want to thank Asmita Pal for being the most helpful person ever. The friendly banters and humorous conversations with her will be forever treasured. I would also like to thank Aatreyi Bal for being the brilliant teacher she is and for being a great friend. I am also fortunate to experience the personality of Rajesh JS for being the chilled out and street-smart person that he is.

I am hugely obligated to my undergraduate college friends Prasid, Nirmal and Arpan and my childhood friend Dodo for being the best companions one can have. I would also like to thank Tricia Brandenburg for being the most prompt and responsible coordinator and for making all the formalities extremely smooth and fast.

Lastly, I would like to thank a very special person in my life, Disha Sinha. Her love and warmth has shaped my personality in the best ways possible. The long phone calls and endless threads of texts that I share with her are priceless. She has literally shown me the right directions at many phases of life and has always been at my side in my good times and bad times. She has played a tremendous role in my life for which I am forever grateful.

Sourav Sanyal

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

GPGPU	General-Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
NTC	Near-Threshold Computing
STC	Super-Threshold Computing
PV	Process Variation
SIMD	Single Instruction Multipe Data
CPAWS	Choke Point Aware Warp Speculator
PTX	Parallel Thread Execution
FU	Functional Unit
FinFET	Fin shaped Field Effect Transistor
NTV	Near-Threshold Voltage (related to VARIUS-NTV)
TC	Thin Channel (related to VARIUS-TC)
VF	Voltage-Frequency
SM	Streaming Multiprocessor (related to GPU Compute Unit)
TB	Thread Block (related to GPU Compute Unit)
STA	Statistical Timing Analysis
WLB	Warp Latency Booster (related to CPAWS components)
WEP	Warp Execution Prioritizer (related to CPAWS components)
OWD	Operand Width Difference
CUDA	Compute Unified Device Architecture (related to GPU Instruction)
APP SDK	Application Software Development Kit (related to AMD's APP SDK)
DLE	Dynamic Load Execution (related to Warp Profiling)
RR	Round-Robin
GTO	Greedy-Then-Oldest
CAWA	Criticality Aware Warp Scheduler
WLB-TB	Warp Latency Booster - Thread Block
WLB-SM	Warp Latency Booster - Streaming Multiprocessor

CHAPTER 1

INTRODUCTION

The growing adoption of general-purpose graphics processing units (GPGPU) over the last decade marks an important landmark in the era of parallel computing. As the industry has continued dedicating overwhelming Single Instruction Multiple Data (SIMD) resources to extract an ever-increasing performance, the power envelope of GPGPUs has undergone a steady rise [1]. With a view to constraining the power consumption, while sustaining the performance, researchers have recently explored the benefits of Near-Threshold Computing (NTC) in the realm of GPGPUs [2]. Despite its impressive energy-efficiency, the NTC design paradigm is afflicted with a severe process variation (PV) induced delay variability problem, throttling the overall system performance [3]. In this work, it is demonstrated how choke points–a unique device-level characteristic of PV at NTC–can redefine the critical warp(s) in GPGPU applications. In the NTC domain, *choke points*–a small set of gates with PV induced delay variations–embody a unique challenge in designing circuits/micro-architectures [4, 5], substantially different from a large body of existing works on general PV [3,6–8]. Performing a dynamic path sensitization analysis, it is observed that an irregular formation of choke points in GPU SIMD lanes can aggravate the delays of the critical warp within a thread block, from 25% to 66%, for a range of GPGPU applications, operating at the NTC condition (Section 3.4). Moreover, owing to a heightened sensitivity to within-die PV at NTC [2], the delay profiles of the warps belonging to the same instruction, can exhibit a significant diversity, across different streaming multiprocessors (SM) of a GPU. *Consequently, choke points at NTC often radically alter the architecturally induced critical warps, thus dwarfing the efficacy of architecturally tailored state-of-the-art warp schedulers* [9–11].

In order to moderate the choke point induced warp delays, as well as, to reduce the performance imbalance among the parallel warps, a family of novel adaptive techniques,

referred to as *Choke Point Aware Warp Speculator (CPAWS)* has been proposed. Using a low-complexity yet efficient profiling architecture, CPAWS first profiles the GPU warps and predicts the choke point induced critical warps in each barrier execution of a GPGPU kernel (Section 4.3). Two modes of design have been proposed in the subsequent steps. In the first mode (Section 4.4), the critical warps, upon detection, are boosted with accelerated execution latencies in their respective functional units for the rest of the barrier intervals. The second mode (4.5), on the other hand, gives more scheduling priority to the detected critical warps, so that their executions are endowed with more SIMD resource allocations, thereby scheduling them more frequently, throughout the barrier intervals.

To the best of my knowledge, this is the first work that uncovers the impact of choke points on the warp latency, and provides a holistic circuit-architectural solution to reclaim the power-performance benefits of GPGPUs at NTC.

The contributions in this work are discussed next:

1.1 Contributions

- The role of choke points is explored in radically transforming the architecturally induced critical warps in GPGPUs, operating at near-threshold condition (Section 3).
- It is demonstrated that existing warp scheduling policies cannot effectively alleviate the delays of choke point induced critical warps (Section 6).
- A family of low-overhead adaptive techniques called *Choke Point Aware Warp Speculator* has been proposed, which identifies circuit-level delay variabilities in the architecture layer, and improves the performance of the parallel warps in GPGPUs (Section 4).
- The best scheme achieves an average of ∼39% and ∼49% improvements in performance, and ∼31% and ∼29% improvements in energy-efficiency, respectively, over two state-of-the-art warp scheduling policies CAWA [10] and GTO [11], across 15 GPGPU applications, while incurring marginal hardware overheads (Section 6).

CHAPTER 2

RELATED WORKS

Earlier research pursuits pertinent to this work can be broadly classified into two categories, as discussed next.

Impact of PV in NTC systems:

Researchers have investigated the prospects of NTC systems for about a decade now. Dreslinski *et al.* explored the energy-efficiency benefits, as well as, the limitations of NTC systems [6]. Karpuzcu *et al.* unearthed the paramount impact of PV in limiting the potential of NTC systems [3]. PV sensitivity of STC GPUs and corresponding performance bottlenecks have gathered multiple research initiatives over time. For example, Aguilera *et al.* have delved into recovering performance loss in PV affected GPUs [12]. *However, PV affected NTC-GPUs have garnered minimal attention from the researchers.*

Basu *et al.* have proposed a run-time technique to adjust the degree of parallelization and speed of long latency datapaths to improve the GPU performance at NTC [2]. De *et al.* indicated that PV sensitivity in NTC systems can engender certain intriguing features, like choke points, that worsen both the reliability and the performance of a system [4]. Bal *et al.* have recently investigated choke points in a CPU and have proposed a dynamic approach to tackle them [5]. *Nevertheless, exploring the significance of choke points in the realm of NTC-GPUs remains an uncharted territory.*

Warp criticality analysis for GPGPU applications:

For many GPGPU applications, conventional round-robin schedulers offer sub-optimal

warp scheduling, leading to the warp criticality problem. Lee *et al.* proposed a critical warp characterization to develop a scheduler which does a better job of reducing the gaps in the warp execution times [9]. In addition, Lee *et al.* further proposed a coordinated warp acceleration technique by managing the compute and memory resources of the GPGPU workloads [10]. *However, none of these works inspect the influence of PV sensitivity in aggravating the warp criticality problem.*

Warps in flight can often lead to memory divergence where some warps execute quickly due to cache hits while others stall due to misses or cache queuing. Ausavarungnirun *et al.* , after characterizing the warp divergence, solved this problem by proposing a set of cache management policies [13]. Liu *et al.* formulated another set of warp scheduling policies based on priority and criticality [14]. Liu *et al.* further proposed a synchronization aware inter-scheduler coordination technique to align dependent warps issued by separate schedulers [15]. Yu *et al.* developed a warp scheduling policy by profiling based on pipeline stalling [16]. In another research conducted by Anantpur *et al.* , a warp scheduling algorithm was proposed which also reduced the execution time by prioritizing warps based on their progress [17].

To the best of my knowledge, this will be the first work to thoroughly analyze the impact of choke points on the warp criticality, and consequently on the performance and energy-efficiency of an NTC-GPU.

CHAPTER 3

MOTIVATION

In this chapter, it is demystified how the presence of choke points exacerbates the warp criticality problem.

The formation of critical warps is explained (Section 3.1) and impact of choke points in further complicating the warp criticality problem at NTC has been discussed (Section 3.2). Using a cross-layer experimental setup (Section 3.3), choke point induced performance imbalance in GPUs have been elaborated (Section 3.4). The chapter is concluded by establishing the need for a warp criticality prediction technique for NTC GPUs (Section 3.5).

3.1 What is Warp Criticality?

Existing warp schedulers maintain a very high performance by efficiently utilizing the SIMD resources among the available warps in the pool. In order to hide the latency due to various architectural hazards, a new warp can preempt the execution of a stalled warp, obviating several penalty cycles. However, a large disparity in the latency distribution of the warps, belonging to a thread block, can lead to additional performance penalties. For example, within a thread block, the slowest warp can keep the other warps waiting in the pipeline for a considerable time, even after their executions. Consequently, a new thread block cannot be swapped in, until the slowest warp in the current thread block finishes its execution. This phenomenon is known as the *warp criticality problem*, and the slowest warp in a thread block is called the *critical warp* [9].

The warp criticality problem prevents an optimal resource utilization within an SM. Previous works have explored several architectural factors (e.g., sub-optimal cache management, homogeneous timing resource allocation), that ensue a critical warp in a thread block [9, 10]. *However, for the first time, this work investigates how choke points aggravate*

the warp criticality problem at NTC, leading to a tremendous performance loss in GPUs.

3.2 Impact of Choke Points on Warp Criticality

In this section, the main focus is on the distribution of choke points–logic gates that alter non-critical paths into critical ones upon sensitization, across fabricated design instances [4, 5]. Being a manifestation of the fabrication process, choke point induced delay variations cannot be estimated by pre-silicon techniques like static timing analysis [5]. The distribution of choke points in a fabricated chip is heavily influenced by specific sensitized circuit topologies [18]. For example, it is observed that, on an average, the ratio of the percentage of total gates, acting as choke points in the data-paths of the GPU functional units (FU), and that in the control-paths of the same FUs, is $\sim 3\times$, for 15 GPGPU applications. *Hence, choke points only in the data-paths of an NTC GPU are considered in this work.*

A choke point in an FU can drastically increase the thread latency, which in turn, can degrade the corresponding warp execution time. In a thread block with an otherwise balanced workload across all its warps, a singular increase in the warp latency worsens the warp criticality problem. If multiple warps are subject to varying degrees of choke point induced delay deviations, that thread block can exhibit a high inter-warp performance imbalance, further degrading the GPU performance. As the delay sensitivity to PV increases at lower operating voltages, NTC GPUs are more prone to choke point induced performance degradations compared to their super-threshold (STC) counterparts. Next, a brief discussion is presented along with an overview of the experimental methodology to clearly understand the interplay between choke point induced performance imbalance and the warp criticality problem in NTC GPUs.

3.3 Methodology

A Southern Island AMD GPU has been modeled [19] and 15 GPGPU applications have been executed in order to obtain cycle-wise FU utilization metadata for several PTX instructions. For each GPGPU application, a representative loop-body has been selected as a thread block, scheduled using a conventional round-robin warp scheduler. To syn-

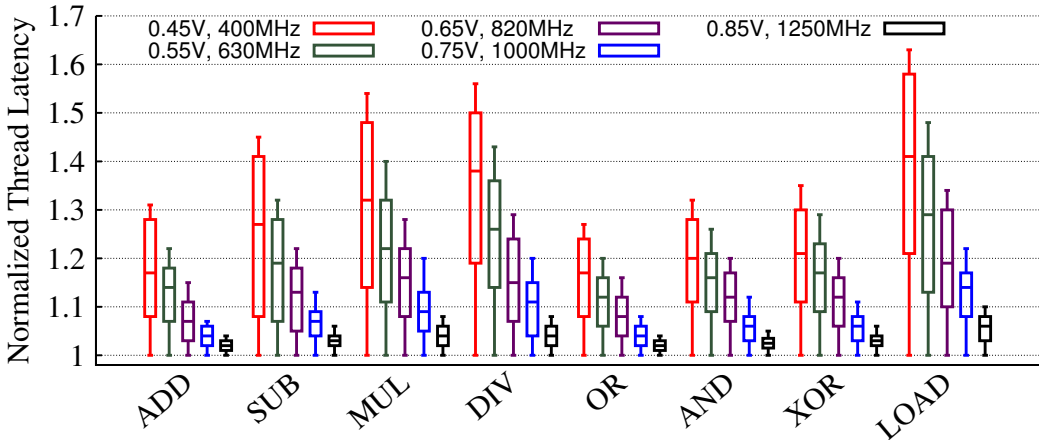

Fig. 3.1: Relative performance slowdown at different operating VF points. A higher value in Y axis indicates a greater slowdown.

thesize a GPU ALU [20], the 15-nm FinFET library from NanGate [21] has been used. To model PV for FinFET technology at STC and NTC operating conditions, the VARIUS [7], VARIUS-NTV [8] and VARIUS-TC [22] models have been incorporated. Feeding an in-house statistical timing analysis tool with the architectural metadata from Multi2Sim and synthesized ALU netlist, the sensitized delay characteristics of several FUs have been obtained.

3.4 Results

In this section, the slowdown of a warp's performance has been demonstrated due to critical thread(s)[1]. At a given operating voltage-frequency (VF) point, the performance slowdown distribution of different threads in a warp has been expressed as the latency distribution of the threads normalized to the latency of the fastest thread belonging to the same warp.

Figure 3.1 illustrates the performance slowdown distributions at various operating VF points for all the threads in a warp across different FUs. Only 2% of the total logic gates in an SM has been considered to be affected by PV, to show the remarkable potency of choke points [18] caused by limited faulty gates.

[1]The slowest CUDA thread in a warp is termed as the critical thread.

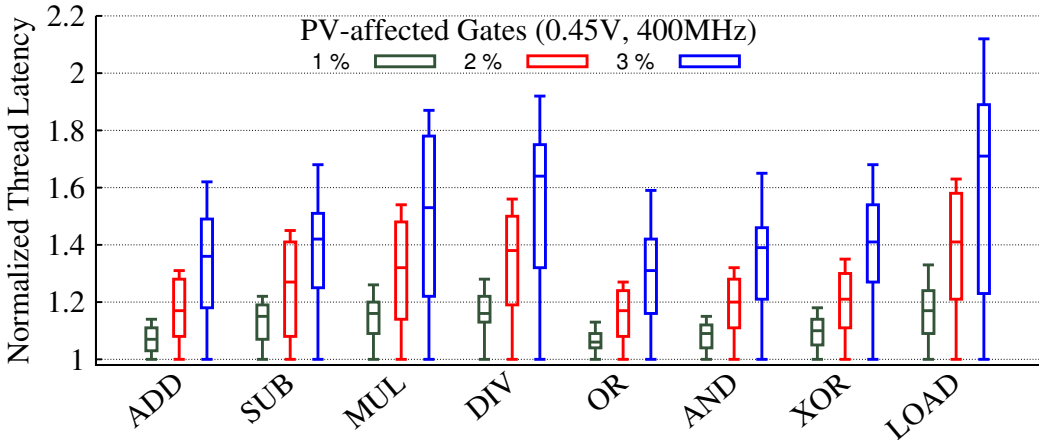

Fig. 3.2: Relative performance slowdown at different extents of PV. A higher value in Y axis indicates a greater slowdown

It is noticed that:

- For a given FU, a lower operating VF results in a greater intra-warp performance variation, indicating a greater slowdown. For example, the slowdown in the critical threads in MUL increases by ∼50% from (0.85V, 1.2GHz) to (0.45V, 400MHz). This is due to a significantly higher delay sensitivity to PV at lower operating voltages. Hence, the warp criticality problem is specifically detrimental to the performance of the NTC GPUs.

- At a given operating point, different FUs are differently affected by choke points. For example, at (0.45V, 400 MHz), the critical thread latency for DIV is ∼19% higher than that for ADD. This indicates the dependence of choke point induced delay degradation on the specific sensitized circuit paths.

Figure 3.2 displays the slowdown for different percentages of PV affected logic gates across various FUs at (0.45V, 400MHz). It is observed that the slowdowns increase with the fraction of the PV affected gates for all the FUs. For example, the critical thread slowdown for SUB when 3% gates are affected by PV, is ∼38% more than that when 1% gates are affected by PV. Circuits fabricated at lower technology nodes being more affected by PV, are also highly prone to the choke point induced warp criticality problem.

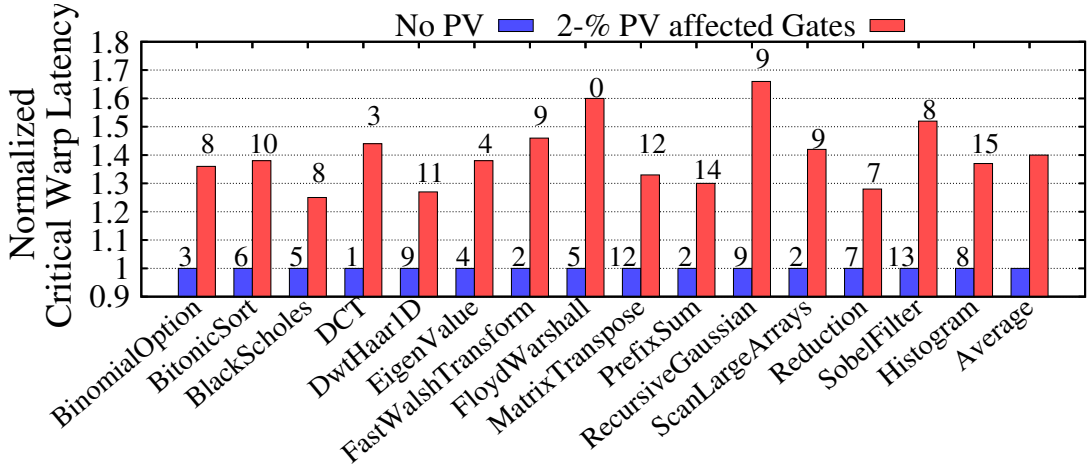

Fig. 3.3: Comparison of the choke point induced critical warp latencies (red bars) in 15 GPGPU applications at (0.45V, 400MHz), and considering 2% PV-affected gates. The values are normalized to the corresponding architecturally induced critical warp latencies (blue bars) found in an ideal PV-free GPU, operating at the same voltage-frequency.

Figure 3.3 illustrates the normalized latencies of the choke point induced (red bars), and the corresponding architecturally induced (blue bars) critical warps, within the representative thread blocks of 15 GPGPU applications, at the NTC operating condition.

It is observed that:

- the latencies of the choke point induced critical warps are significantly greater than the corresponding architecturally induced critical warps. The presence of choke points at NTC, exacerbates the critical warp latencies from 25% to 66%, with an average increase of 40%.

- The formation of choke points engenders new critical warps—different from the respective architecturally induced critical warps—in 11 out of 15 GPGPU applications. This fact is reflected by the warp IDs on top of each bar in Figure 3.3. For 4 applications (viz., *EigenValue*, *MatrixTranspose*, *Reduction* and *RecursiveGaussian*), choke points only increase the corresponding latencies of the architecturally induced critical warps, without ensuing a new critical warp in their representative thread blocks.

As architecturally tailored warp schedulers [9,10] are agnostic of circuit-level dynamic delay variabilities, they cannot efficiently tackle the choke point induced critical warps at NTC.

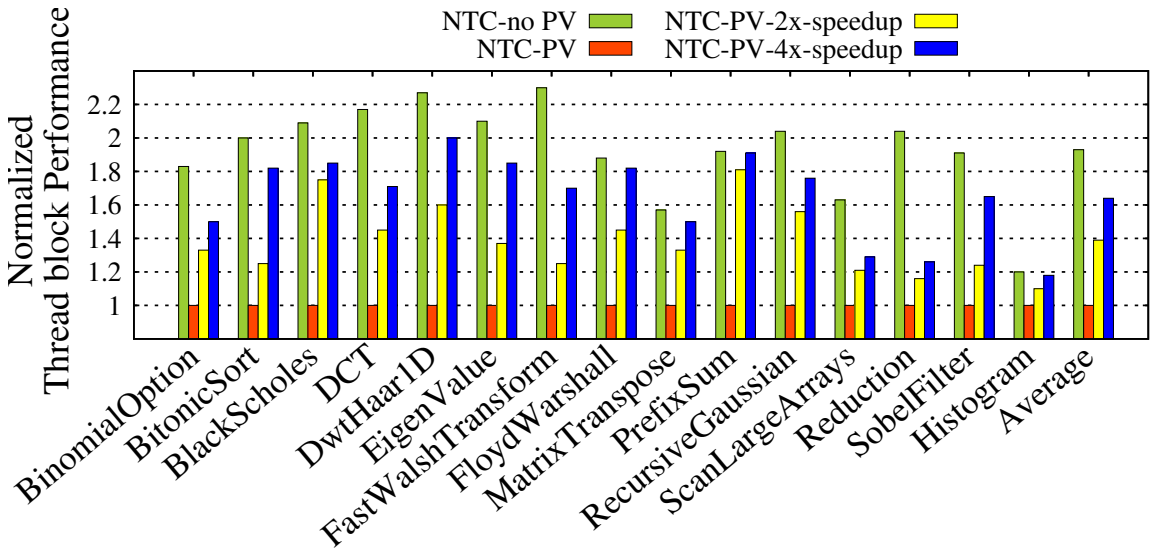

Fig. 3.4: Comparison of thread block performances of 15 GPGPU applications under different scenarios of NTC. The values are normalized to the corresponding PV affected NTC-GPU without any speedup, operating at the same voltage-frequency.

3.5 Insight to CPAWS

Figure 3.4 illustrates an empirical analysis where the critical warp latencies of the concerned thread block (Section 3.3) are statically sped up by factors of 2× and 4× for the 15 applications. The thread block performance is modeled as the inverse of the execution latency normalized to that of an NTC-GPU with 2% PV-affected gates. For the same thread block, the performance of a PV-free NTC-GPU is ∼93% more than the PV-affected NTC GPU. However, it is observed that even in presence of PV, with static speed-ups applied to the critical warp latencies, the average thread block performance improvements over the PV affected NTC-GPU is significant. With 2× and 4× speed-ups, performance improves by ∼39% and ∼64% respectively across all the applications. For 4× speed-up, the performances of *PrefixSum* and *FloydWarshall* are very close to a PV-free NTC GPU. A similar improvement is also noticed in the thread block latency when the critical warps are allocated more SIMD time slots. This result hints the benefit of improving the execution latencies of critical warps, which can reduce the idle time within a thread block, leading to better performance.

Hence, it is imperative to dynamically predict the timing behaviors of the warps, and accordingly, speed up the execution of the critical warp(s) in a thread block. In the following section, a family of novel circuit-architectural techniques has been developed that mitigates the performance loss due to warp criticality, while incurring a marginal power overhead to sustain the energy-efficiency benefit of NTC.

CHAPTER 4

CHOKE-POINT AWARE WARP SPECULATOR

In this chapter, *Choke-Point Aware Warp Speculator*—a family of adaptive techniques that dynamically predicts and improves the execution latencies of the critical warps in presence of PV-induced choke points (Section 4.1) is presented. A brief system overview is given and the two variants of CPAWS (Section 4.1) are presented. The design challenges are outlined (Section 4.2) and different phases of CPAWS are discussed (Sections 4.3, 4.4 and 4.5).

4.1 System Overview

The first and foremost stage of CPAWS is to predict the critical warps that are present in the GPU pipeline. A low-overhead circuit-level *Warp Profiler* is formulated in this context, which profiles the scheduled warps and identifies the critical ones during the execution of GPGPU kernels. Once, the critical warps are identified, CPAWS activates the second stage wherein the identified critical warps are executed with reduced latencies along the GPU SIMD lanes. Two variants are proposed:

- ***Warp Latency Booster (WLB)*** – This consists of a *Boost controller* which launches the critical warp(s) with boosted voltage-frequencies along their respective FUs for the rest of their barrier executions (Section 4.4).
- ***Warp Execution Prioritizer (WEP)*** – This architectural policy uses a *Priority Tuner* which alters the execution priority of the scheduled warps by allocating more SIMD time resources to the identified critical warps during subsequent barrier execution (Section 4.5).

Figure 4.1 represents a conceptual block diagram of CPAWS where both WLB and WEP are illustrated. A baseline round-robin (RR) thread block scheduler (not shown in

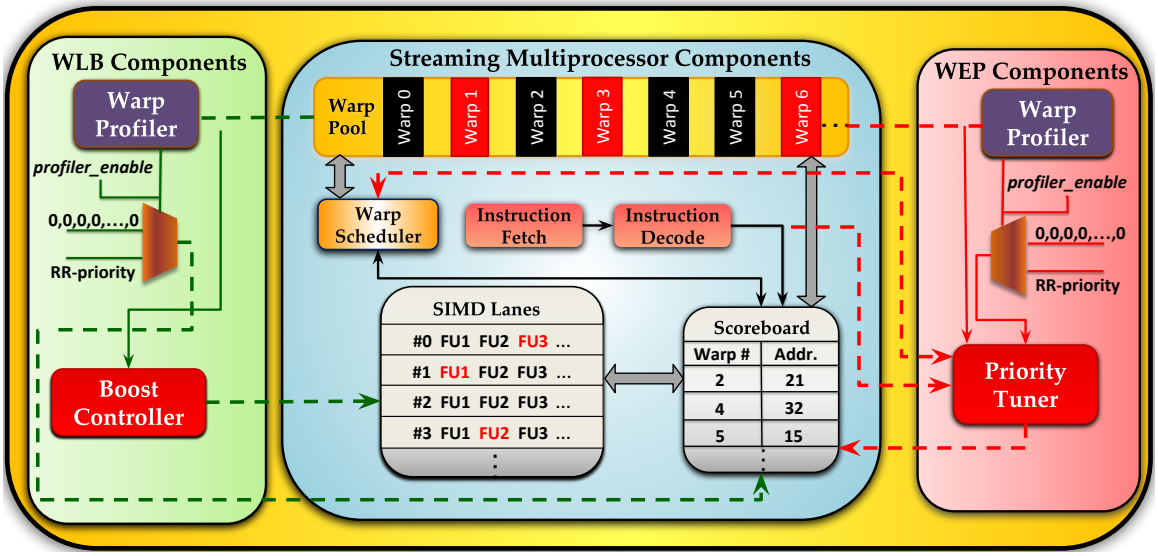

Fig. 4.1: The warp profiler predicts critical warps (Warp 1, Warp 3 and Warp 6) for both WLB and WEP. The boost controller executes them with increased latencies. The priority tuner updates the scheduling policies to allocate more timing slots. (FU3 in lane 0, FU1 in lane 1, and FU2 in lane 3) latencies are improved by employing WLB/WEP.

Figure 4.1) has been assumed to schedule thread blocks to available SMs [23]. Every SM maintains a scoreboard to manage the context of each warp in a thread block. An RR warp scheduler has been augmented to work in tandem with the proposed warp profiler. Given a barrier interval, the warp profiler ascertains the critical warp(s) in the profiling phase (Section 4.3). The baseline NTC GPU has been operated at a better-than-worst case VF point, and an existing error detection and correction mechanism has been employed to tackle intermittent timing violations [24].

4.2 Design Challenges

The design of CPAWS is associated with several crucial challenges, as listed below.

- **Choke Point Awareness:** It is imperative to correctly identify the warps that are most affected by choke point induced delay variabilities in the SIMD lanes. As the critical warps are selectively sped up, a misidentification can potentially lead to an even greater performance heterogeneity within an SM, severely degrading the power and performance of an NTC GPU.

- **Harmonious Execution of Profiler and Scheduler:** The baseline RR warp scheduler offers an impressive resource utilization by *fairly* scheduling the ready warps in the pool. It is critical to ensure that the warp profiler works in harmony with the RR warp scheduler, while analyzing the timing profiles of the warps.

- **Timing Overhead for Warp Profiling:** The warp profiler senses circuit level delays in order to identify critical warps during runtime. The timing overhead associated with such profiling should be kept to a minimum in order to achieve optimal performance.

- **Pipeline Synchronization:** It is essential to maintain synchronization among the GPU pipe stages when some warps are executed with boosted speeds. For example, any warp having an architectural dependency on a boosted warp needs to be issued earlier in order to prevent a queuing delay.

- **Maintaining High Energy-Efficiency:** It is important to judiciously trade off between the performance improvement of the warps and the associated power overheads. The warp-profiling should incur a minimal performance penalty, while the circuits of the profiler, boost controller and execution prioritizer should have marginal hardware footprints.

4.3 Profiling Phase

4.3.1 Principle of Warp Profiling

In view of choke point awareness, one of the most important questions in the design of CPAWS is: *how can it be guaranteed that a warp, identified as a critical one in the profiling phase, continues to be critical for the rest of the barrier interval?*

The answer to this question lies in the inherent nature of the warp execution. Say, n CUDA threads, comprising a warp, are scheduled to be executed on m SIMD lanes in an SM, where $n = k * m$ and k is a positive integer greater than 1. As each warp corresponds to only one instruction, each of the n CUDA threads utilizes the same FU. Assuming the FU latency to be 1 cycle, the total execution time of the warp is k cycles in the absence of

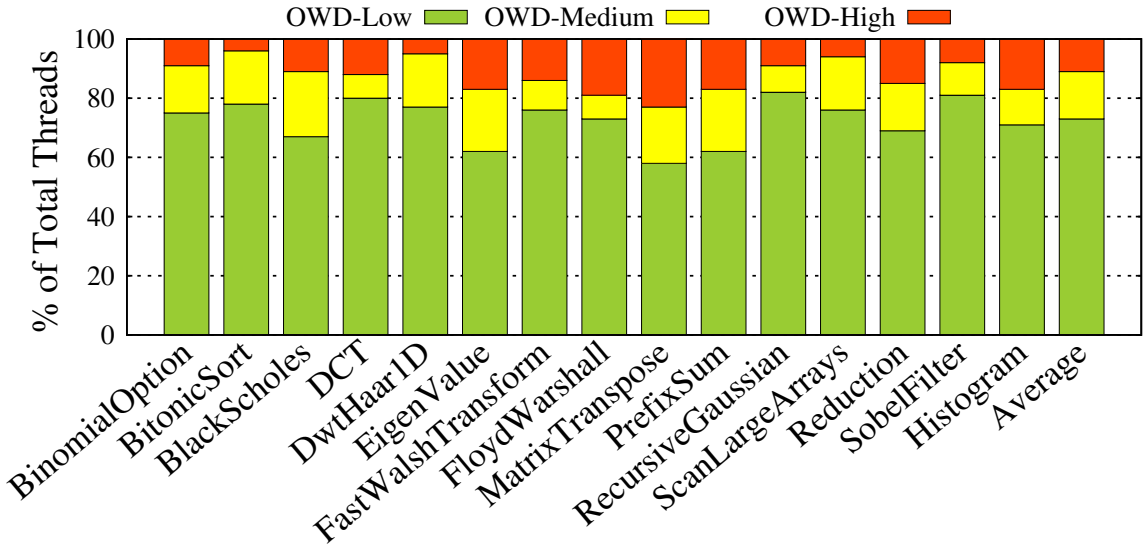

Fig. 4.2: Distribution of operand width difference (OWD) among the threads for 15 GPGPU applications at NTC.

pipeline stalls. As the warp utilizes the same FU in each of the k cycles, the choke point induced delays of the sensitized paths are triggered by the same instruction type across all the k cycles.

One thing to however note is that, the significant width of operands, i.e. the number of set operand bits also affect the formation of choke points. The sensitized path delays are hence potentially influenced by the changes in the significant operand widths in each iteration [5]. Therefore, a smaller difference in the significant operand widths in consecutive iterations of the same thread block leads to a better prediction of the warp criticality. In the subsequent iterations following the profiling phase, the metric Operand Width Difference (OWD) has been formulated to signify the percentage change in significant operand widths of the CUDA threads with respect to that of the profiled critical CUDA threads. The OWDs are classified as OWD-Low, OWD-Medium and OWD-High, when OWDs are less than 30%, between 30% to 60 % and more than 60% respectively.

Figure 4.2 shows the OWD distribution of the representative thread blocks for the 15 GPGPU applications. It is observed that, on an average, $\sim$73% of CUDA threads show less than 30% relative difference in significant operand width. It is inferred that for GPGPU

applications, the critical warp occurrences are dominated by the instruction type delays. Thus, the warp's timing behavior in the first $t_{profile}$ cycles ($t_{profile} < k$) of its execution, is a good proxy for its overall timing characteristic, where $t_{profile}$ is denoted as the *Profiling Time*. A certain number of cycles is also fixed as *Profiling Threshold* ($< k$), after the elapse of which the profiler updates the set of identified critical warps. Lowering the profiling threshold increases the prediction accuracy but at the cost of additional timing overhead.

4.3.2 Warp Profiler

Algorithm 1 shows the working principle of the profiler. To compare the timing profiles of the warps, the warp scheduler needs to issue all the warps simultaneously. This is a deviation from the default RR behavior, where only one warp is scheduled to execute in one cycle. Several strides have been made to ensure a harmonious execution of the warp profiler and the warp scheduler, which is detailed next.

Before the beginning of the profiling phase, the warp profiler communicates with the RR warp scheduler to assign equal priorities to all the warps in a thread block. Consequently, the warps get scheduled simultaneously. Once scheduled, the priorities of the warps are set to a high value (low priority) to obviate rescheduling of warps during the profiling phase.

For the purpose of profiling, each warp in the thread block needs to execute for at least one iteration. Hence, the duration of the profiling phase is set as the maximum of the execution times of all the instructions pertaining to the scheduled warps (line 3 in Algorithm 1). To account for PV, the instruction execution times are ascertained and stored for each SM, post fabrication. A low-complexity heuristic is adopted (line 6 to 13 in Algorithm 1) to ascertain the critical warp(s). The progress of a warp is tracked using the existing dynamic load execution (DLE) counter in modern GPUs [25]. The DLE count gets incremented only if the warp has finished executing a load. The critical warps, being tardy, have smaller DLE counts compared to the faster warps in the same thread block. When a thread block executes a load, the warp profiler logs the DLE counts for all the warps in that thread block. If the DLE count of a warp is found to be r ($r > 0$) standard deviations less than the mean DLE count of all the warps, that warp is inferred to be critical. The value of r

Algorithm 1 Warp Profiling

1: $w_i \epsilon W \quad W \leftarrow Available\ Warp\ Pool$
2: $w_i\_priority \leftarrow 0 \quad \forall w_i \epsilon W$
3: $t_{profile} = t_{clock} * Max(CPI_j) \quad \forall j \epsilon I_{w_i} \quad \forall w_i \epsilon W$
4: $Schedule\_Warps()$
5: $w_i\_priority \leftarrow \infty \quad \forall w_i \epsilon W$
6: $C_{w_\mu} \leftarrow 0$
7: **while** $t_{profile} > 0$ **do**
8: $\quad C_{w_\mu} \leftarrow C_{w_\mu} + C_{w_i} \quad \forall \quad C_{w_i} \leftarrow Warp's\ Execution\ Count$
9: **end while**
10: $C_{w_\mu} \leftarrow C_{w_\mu}/n \quad n \leftarrow Total\ No.\ of\ Warps$
11: **if** $C_{w_i} < C_{w_{\mu - r*\sigma}}$ **then**
12: $\quad CriticalWarp[\,] \leftarrow w_i \quad \forall w_i \epsilon W$
13: **end if**
14: $w_i\_priority \leftarrow RR - Priority \quad \forall w_i \epsilon W$

can be found empirically. Note that this mechanism can identify both choke point induced, as well as, architecturally induced critical warps. At the end of the profiling phase, all the warps are updated with the default RR priorities. As no additional delay sensing circuit is employed, the performance and hardware overheads of the profiler is marginal.

4.4 Warp Latency Booster (WLB)

The top few critical warps, obtained from the profiling phase, are executed with higher execution speeds in this scheme for the remaining barrier execution time. A low-overhead boost controller communicates with the profiler and manages the execution speeds of the warps. The speed enhancement is realized by dynamically boosting the latencies of the corresponding FUs in the SIMD lanes.

4.4.1 Circuit-Architectural Considerations for Boost

- In pursuit of sustaining an error-free and optimal pipeline activity, the dispatch logic is altered. By monitoring the progress of the boosted warps [25], subsequent dependent warps are accordingly dispatched. This, in turn, alleviates scoreboarding and results in a better thread block performance.

- In modern GPUs, many long-latency (i.e., greater than one clock cycle) FUs are pipelined to improve the warp performance. In the boosting phase, these pipelines are dynamically made transparent, so as to complete the FU executions in less times [2].

- To maintain low implementation overheads, two boost levels are considered, realized using additional voltage rails [2]. The runtime transition among different voltages is achieved using a setup similar to the one proposed in [26]. The timing and the hardware overheads from the additional voltage rails are considered in our evaluations (Section 6.5).

- In order to avoid complex synchronization circuitry, only integral boost levels are considered in this work, specifically, 2× and 3× the nominal FU latencies. Out of the n critical warps, the boost controller applies 3× boost to the top $n/2$ warps, and 2× boost, to the rest $n/2$ warps.

4.4.2 Improving Energy-Efficiency

- The latency improvement of a warp plateaus at very high boost levels (viz., 4× and more) when the FUs consume significantly high energy. Consequently, the maximum boost level is stipulated to be 3× the corresponding nominal execution speed of an FU.

- If a boosted warp is stalled due to any architectural hazard, it can incur a tremendous power overhead. To avoid this situation, a warp's execution time is tracked [25], and if the execution time crosses a threshold, the boost is disabled, i.e., the execution speed is reduced to the corresponding nominal value.

- When the performance heterogeneity among the parallel warps is insignificant, boosting the critical warp's execution can cause a non-critical warp to become critical. This phenomenon is known as *criticality inversion* [9] which reduces the benefit of WLB.

4.5 Warp Execution Prioritizer (WEP)

The main objective of WEP is to offer an architectural alternative to the voltage-frequency boosting solution of WLB. WEP aims to alleviate the inter warp performance imbalance by smartly reevaluating the warp scheduling priorities. The critical warps obtained from the profiling phase are given more priority in this mechanism by scheduling them more often along the SIMD lanes. Next, the implementation of this warp scheduling policy is explained in greater details.

Algorithm 2 Priority Tuner

```
1: if newBarrier() then
2:     create_PriorityVector(WarpProfiler, CriticalWarp[])
3:     updateIssueQueue()
4:     remWarps ← count(AvailableWarpPool)
5: end if
6: if CurrCycle − LastSortCycle > ProfilingThreshold then
7:     LastSortCycle ← CurrCycle
8:     Sort_Threadblocks(remWarps)
9:     for each Warp in Threadblock do
10:        WarpList ← sortWarps(PriorityVector)
11:    end for
12: end if
13: AddWarpstoIssueQueue(WarpList, ReadyWarps)
14: Schedule_Warps()
15: remWarps ← CalcremWarps(WarpList, IssuedWarps)
16: update_PriorityVector()
17: while !newBarrier() do
18:    ScheduleWarps(remWarps, PriorityVector)
19: end while
```

With the onset of each barrier, the priority tuner generates a new priority vector based on the warp criticality information and updates the issue queue (line 2-3 in algorithm 2). It also keeps track of the warps which have finished execution and gives more priority to those thread blocks which have more number of finished warps. This is because thread blocks with less number of active warps will tend to have lesser execution overlap, thereby exposing the SIMD latencies. Inside each thread block, the priority tuner updates the priority vectors in ascending order of warp progress. This allows the warp scheduler to

execute the critical warps faster as they are scheduled more frequently, thereby endowing them with more compute and time resources. Throughout the course of this operation, the number of active warps are constantly monitored and updated. The residing sibling warps in a thread block gradually reduce in number as their execution approaches termination. The scheduling priority is accordingly updated after every successive warp profiling phase (line 8-16 in algorithm 2).

Since the number of number of warps is not very large, the priority tuner takes only tens of cycles to modify the scheduling policy, resulting in marginal timing overhead. Although WEP does not involve any boosting, it can still cause criticality inversion when there is considerable difference in evaluated scheduling priority between critical and non-critical warps having small latency difference.

CHAPTER 5

METHODOLOGY

In this chapter, an extensive cross-layer methodology, used to implement and evaluate the design of WLB and WEP components of CPAWS, is explained. Figure 5.1 depicts the three layers of the methodological setup

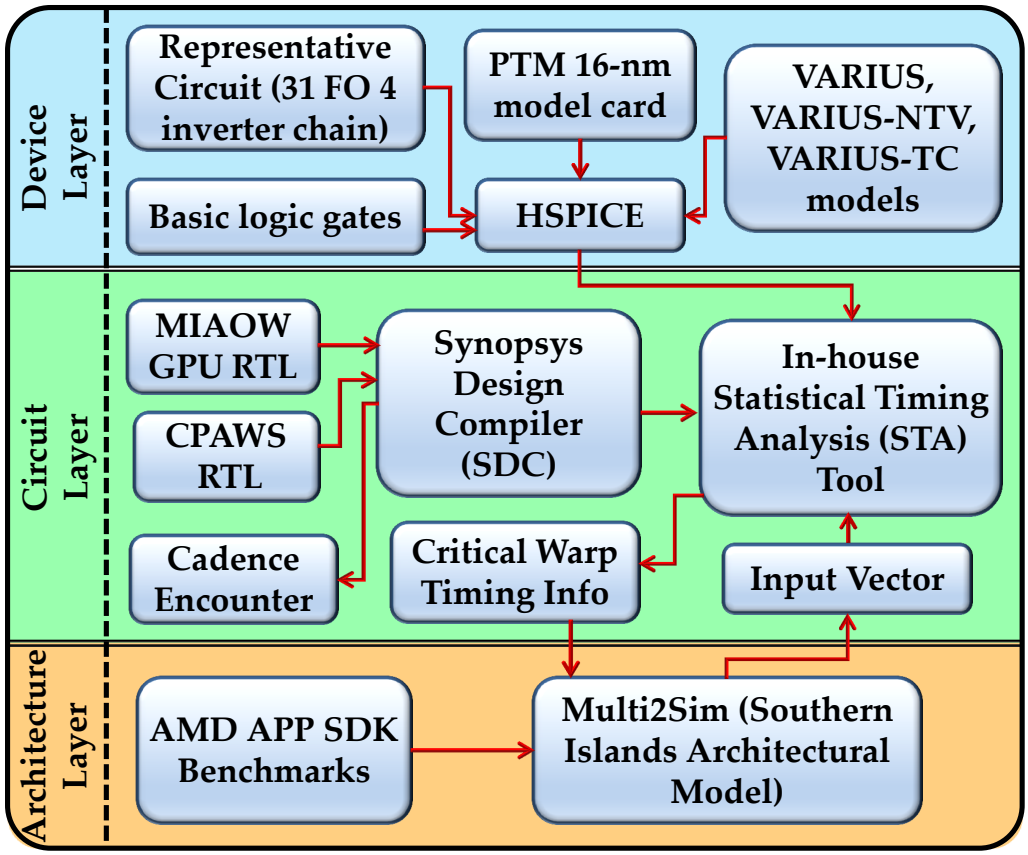

Fig. 5.1: Cross-layer methodology for CPAWS.

5.1 Device Layer

The NTC energy consumptions are estimated by performing HSPICE simulations on the basic logic gates (viz., NAND, NOR and Inverter). The 31-stage FO4 inverter-chain has been used as a representative of various combinational logics in a GPU [2]. The simulation parameters are obtained from the 16-nm Predictive Technology Model [27]. The interconnect power has been assumed to be ∼50% of the core dynamic power at NTC [2]. The delays of the basic gates are used in the circuit layer (Section 5.2) to ascertain the latencies of the warps sensitizing different FUs in the SIMD lanes. The impact of the within-die PV for STC and NTC operating conditions is incorporated by using the VARIUS [7] and VARIUS-NTV [8] models. The FinFET characteristics are obtained using the VARIUS-TC model [22].

5.2 Circuit Layer

The WLB and WEP components of CPAWS have been implemented by augmenting an open-source reference GPU RTL [20]. The reference and the augmented GPU RTLs have been synthesized using Synopsys Design Compiler, at the NTC operating condition. Place and route of the synthesized netlist have been performed using Cadence SoC Encounter to estimate the area, power, and wirelength overheads of CPAWS. The synthesized GPU netlist, along with an encoded PTX instruction vector obtained from the architecture layer (Section 5.3), have been used as inputs to the in-house statistical timing analysis (STA) tool. The warp timing information has been exported to the architecture layer to obtain the application performance.

5.3 Architecture Layer

The codebase of the Multi2Sim architectural simulator (version 5.0) [19] has been instrumented to implement the CPAWS. For evaluation, the AMD Southern Island GPU architecture have been emulated on Multi2Sim. Similar to [2], only the SIMD cores have

Parameters	Configurations
No. of SMs	128
Thread Block Size	16
FU Latency	4-64 / 2-32 / 1-21 cycles
Local Memory	32 KB, latency: 2 cycles
L2 Cache	8×768 KB, latency: 20 ns
Global Memory	B/W: 264 GB/s, latency: ∼300 ns

Table 5.1: NTC GPU architectural configurations.

been assumed to be operating at the NTC condition, while the memories operate at the STC region. The specific architectural parameters for evaluation are outlined in Table 5.1. 15 GPGPU applications from AMD's APP SDK suite [28] have been used to compare the power-performance benefits of the comparative schemes. To obtain the critical warps in different barrier intervals of an application, the executed instruction metadata has been collected from Multi2Sim, and is used as input vectors to the STA tool (Section 5.2). The boost phase of the WLB and priority tuning phase of WEP have been implemented by dynamically altering the FU latencies and by changing the scheduling priorities in Multi2Sim, respectively, for some warps based on the warp criticality information obtained from the STA tool.

CHAPTER 6

EXPERIMENTAL RESULTS

In this chapter, the different warp management policies which have been used as comparative schemes are elucidated (Section 6.1). A detailed discussion on the warp profiler prediction accuracy is given (Section 6.2) and an elaborate analysis of the comparative schemes is presented in terms of performance (Section 6.3) and energy-efficiency (Section 6.4). Additionally, the implementation overheads of CPAWS are discussed (Section 6.5).

6.1 Comparative Schemes

The following warp scheduling schemes are evaluated at the NTC operating condition (0.45V, 400MHz), considering 2% of randomly chosen logic gates, comprising all the SIMD FUs in an SM, being affected by PV.

- **Round-Robin (RR)**: This is the baseline scheme, where the scheduling is done by issuing a single warp in every clock cycle, after assigning equal priority to each warp.
- **Greedy-Then-Oldest (GTO)**: This scheme executes a single warp till it stalls and then selects the oldest ready warp for the execution [11].
- **Criticality Aware Warp Scheduler (CAWA)**: It consists of a a *warp scheduler* that allocates more SIMD resources to the predicted critical warps, and a *cache prioritizer* that favors the critical warps while accessing the cache [10].
- **WLB - Streaming Multiprocessor (WLB-SM)**: In this variant of WLB, implemented at the granularity of an SM, the warps across an entire SM are subjected to profiling in a given barrier execution.

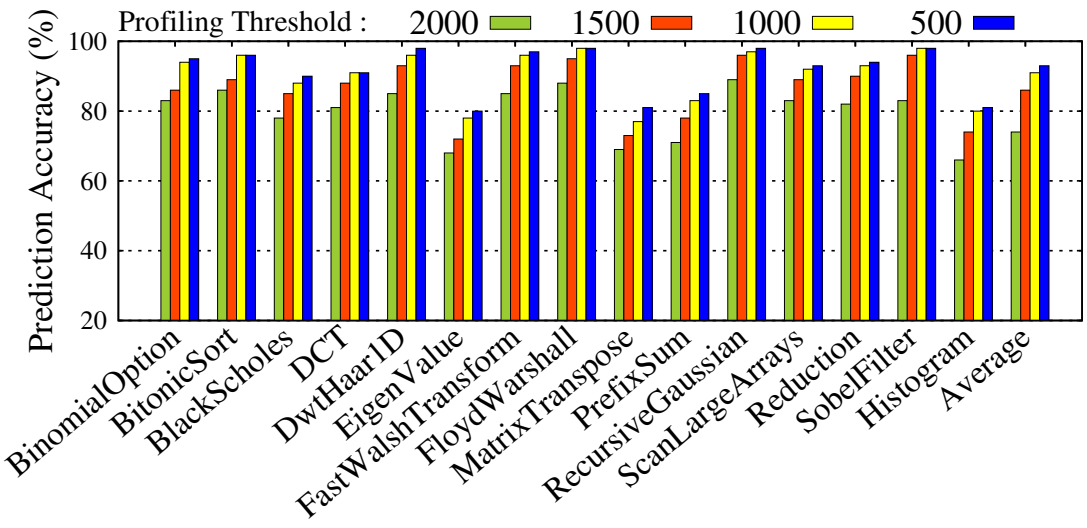

Fig. 6.1: Prediction accuracy comparison by tuning the profiling threshold of the CPAWS Warp Profiler.

- **WLB - Thread Block (WLB-TB)**: This is another variant of WLB, implemented at the granularity of a thread block. Unlike WLB-SM, the profiling for identifying choke point induced critical warps, is done for every thread block.

- **Warp Execution Prioritizer (WEP)**: This is the architectural policy which allocates more timing and compute resources to the profiled critical warps. WEP's granularity is a joint implementation at both thread block and SM levels. Though the profiler detects the critical warps for every thread block separately, WEP also sorts all the thread blocks in a descending order of execution in a given execution kernel.

6.2 Prediction Accuracy

Figure 6.1 shows the prediction accuracies of the warp profiler for different values of *Profiling Threshold*. Each benchmark has been simulated for 100 million instructions to note the profiled critical warps which has then been compared with oracle knowledge to evaluate the prediction accuracies. As the formation of critical warps is heavily dependent on the algorithmic artifact of the benchmarks, there is a notable variation in the prediction accuracy. It is seen that the improvement in accuracy is minimal as the profiling threshold

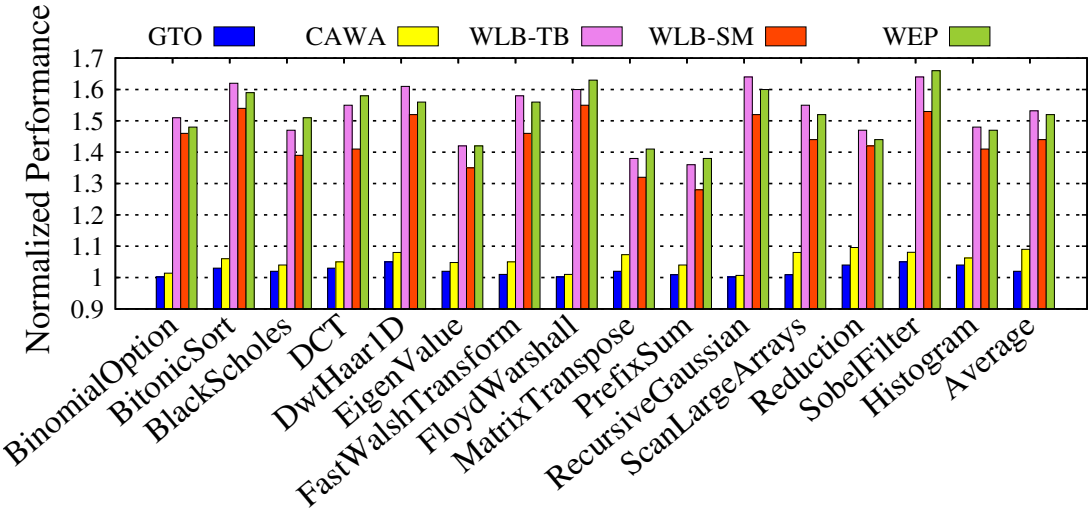

Fig. 6.2: Performance comparison (Higher is better).

is reduced from 1000 cycles to 500 cycles (∼91% to ∼93% on average). Hence, for a judicious trade-off between prediction accuracy and profiling overhead, 1000 cycles have been considered to be the profiling threshold for further analysis.

6.3 Performance Results

Figure 6.2 depicts the performances of 15 GPGPU applications under different comparative schemes (Section 6.1). The results are normalized to the respective performances under the baseline RR warp scheduler. On average, WLB-TB, WLB-SM and WEP outperform RR (by ∼53%, ∼44% and ∼52%), GTO (by ∼50%, ∼41% and ∼49%), and CAWA (by ∼41%, ∼32% and ∼39%), respectively. The warp criticality predictor in CAWA can identify only the architecturally induced critical warps [10]. As choke points often radically alter the architecturally induced critical warps (Section 3.4), CAWA frequently allocates sub-optimal SIMD resources for the choke point induced critical warps, leading to minor performance improvements over RR. On the other hand, as GTO executes a single warp till it stalls and then selects the oldest one, GTO often stalls repeatedly due to the choke point induced critical warps, incurring a severe performance penalty. For all the applications, WLB-TB offers the best performance. Next, the performance difference between WLB-TB, WLB-SM and WEP is elaborated.

CPAWS Variants	Criticality Inversion	Performance Improvements
WLB-SM	∼4.8%	∼44%
WLB-TB	∼2.3%	∼53%
WEP	∼2.7%	∼52%

Table 6.1: Criticality inversion and the performance improvements of the CPAWS variants over RR.

6.3.1 Criticality Inversion

Table 6.1 enumerates the frequency of the occurrences of the criticality inversion (Section 4.4.2) in the WLB variants and WEP, along with their respective average performance gains, compared to the baseline RR warp scheduler. A lower value of criticality inversion is associated with a higher performance improvement. WLB-SM is agnostic of the specific performance heterogeneity of the warps inside each thread block. Instead, it collectively profiles all the warps in an SM, to ascertain the critical warp(s) for that SM. Consequently, the inferred boost level(s) can potentially over-speed some warps in their respective thread blocks, resulting in several criticality inversions. WEP aims to schedule the critical warps in every thread block more frequently, but also in the order obtained after sorting the thread blocks on the basis of shortest remaining time. Consequently, WEP causes lower criticality inversion and hence better performance than WLB-SM. WLB-TB is explicitly tailored to reduce the performance imbalance at the granularity of a thread block by boosting the critical warps in every thread blocks separately. Hence it causes the least number of criticality inversions, and offers the best performance among all the schemes.

6.4 Energy-efficiency Results

Figure 6.3 shows the energy consumptions for all the comparative schemes, normalized to baseline. WLB-TB, WLB-SM consume ∼40% and ∼26% more energy, respectively, compared to the baseline. A large proportion of these energies comes from the power overheads from the boosting phase. As WLB-TB boosts more warps in a barrier interval, on an average, with respect to WLB-SM, the former consumes significantly more energy than the latter, for all the applications. The additional hardware for critical warp prediction

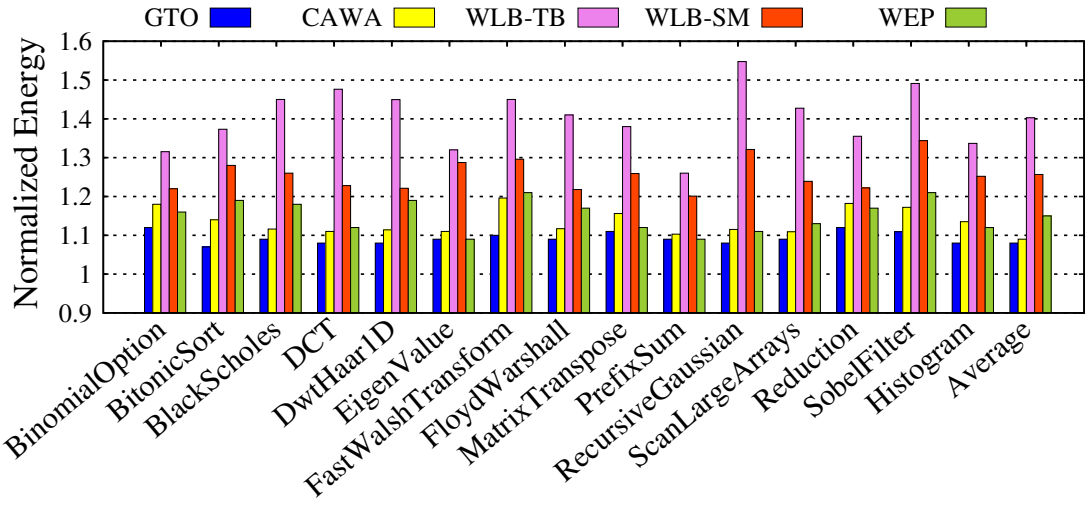

Fig. 6.3: Energy comparison (Lower is better).

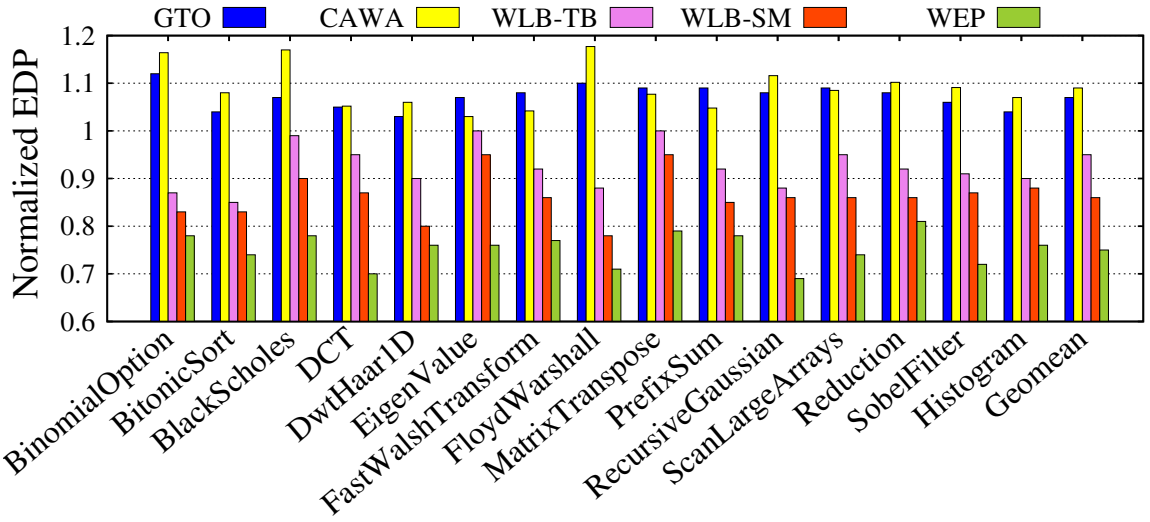

Fig. 6.4: EDP comparison (Lower is better).

in CAWA, and additional control logic in GTO, make them more energy-hungry than RR. WEP on the other hand consumes only ∼15% more energy compared to baseline. The energy consumption of WEP is required only for the warp profiling and scheduling modifications. Unlike the WLB twins, WEP does not involve any boosting mechanism, thereby making it the least energy-hungry scheme.

Figure 6.4 presents the energy efficiency benefits for all the comparative schemes in terms of energy-delay product (EDP), normalized to baseline. The proposed schemes (WLB-

SM, WLB-TB and WEP) are more energy-efficient than RR, GTO and CAWA. This is because the large performance improvements of WLB twins schemes grossly amortize their relatively high energy footprints. As WEP does not require boosting and yet improves performance significantly, it is the most energy-efficient scheme, surpassing RR, GTO and CAWA by ∼25%, ∼29%, and ∼31%, respectively, on an average. Energy efficiency of GTO is inferior to the proposed techniques, as GTO achieves only a minor performance improvement over RR, at the expense of a significantly high energy consumption. CAWA is the least energy-efficient scheme across all applications, on average, as its high energy footprint eclipses its meager performance benefits.

6.5 Implementation Overheads

The implementation hardware overheads are calculated with respect to the SM of an NTC GPU with RR warp scheduling. The hardware cost of WLB comes from the warp profiler, the boost controller, and the additional voltage rails. The implementation cost of the performance counters are excluded, as they are present in commercial off-the-shelf GPGPUs. Due to identical implementations of the warp profiler and the voltage rails, and only a minor difference in the boost control logic, both WLB-TB and WLB-SM incur almost identical overheads. The area, wire-length and power overheads for WLB are 0.71%, 2.4%, and 4.1%, respectively. As WEP does not include the boost controller, its area, wire-length and power overheads are 0.42%, 1.7%, and 2.9% respectively.

CHAPTER 7

CONCLUSION

Preserving the energy efficiency benefit of an NTC-GPU necessitates addressing choke point induced delay variabilities among parallel warps. In this work, it is demonstrated that cutting-edge GPU warp scheduling policies are oblivious to circuit-level performance variations, thus worsening the so-called warp criticality problem at the NTC regime. Deploying an efficient warp profiling framework in the architecture layer, choke point induced critical warps are predicted in a GPGPU kernel. To improve the inter-warp performance, the execution latencies of the critical warps are improved in their respective FUs, with minimal modifications to a baseline GPU, in different ways, using disparate circuit and architectural schemes. The cross-layer simulations on a range of GPGPU benchmarks, reveal that the proposed technique can gracefully tackle choke points in NTC-GPUs, leading to performance and energy-efficiency improvements, over state-of-the-art warp schedulers.

REFERENCES

[1] 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.

[2] P. Basu, H. Chen, S. Saha, K. Chakraborty, and S. Roy, "Swiftgpu: Fostering energy efficiency in a near-threshold gpu through a tactical performance boost," in *53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016.

[3] U. R. Karpuzcu, N. S. Kim, and J. Torrellas, "Coping with parametric variation at near-threshold voltages," in *IEEE Micro 33rd Volume*, 2013, pp. 6–14.

[4] V. De, "Fine-grain power management in manycore processor and system-on-chip (soc) designs," in *IEEE/AC, International Conference on Computer Aided Design*, 2015, pp. 159–164.

[5] A. Bal, S. Saha, S. Roy, and K. Chakraborty, "Revamping timing error resilience to tackle choke points at ntc systems," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, 2017, pp. 1020–1025.

[6] R. Dreslinski, M. Wieckowski, D. Blaaw, and D. Sylvester, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," in *Proceedings of the IEEE*, 2010, pp. 253 – 266.

[7] 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.

[8] 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," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012, pp. 1–11.

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

[10] S. Lee, A. Arunkumar, and C. Wu, "CAWA: coordinated warp scheduling and cache prioritization for critical warp acceleration of GPGPU workloads," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, 2015, pp. 515–527.

[11] T. G. Rogers, M. O'Connor, and T. M. Aamodt, "Cache-conscious wavefront scheduling," in *45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2012, Vancouver, BC, Canada, December 1-5, 2012*, 2012, pp. 72–83.

[12] 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," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, 2014, pp. 1–6.

[13] R. Ausavarungnirun, S. Ghose, O. Kayiran, G. H. Loh, C. R. Das, M. T. Kandemir, and O. Mutlu, "Exploiting inter-warp heterogeneity to improve GPGPU performance," in *2015 International Conference on Parallel Architecture and Compilation, PACT 2015, San Francisco, CA, USA, October 18-21, 2015*, 2015, pp. 25–38.

[14] Y. Liu, Z. Yu, L. Eeckhout, V. J. Reddi, Y. Luo, X. Wang, Z. Wang, and C. Xu, "Barrier-aware warp scheduling for throughput processors," in *Proceedings of the 2016 International Conference on Supercomputing, ICS 2016, Istanbul, Turkey, June 1-3, 2016*, 2016, pp. 42:1–42:12.

[15] J. Liu, J. Yang, and R. G. Melhem, "SAWS: synchronization aware GPGPU warp scheduling for multiple independent warp schedulers," in *Proceedings of the 48th International Symposium on Microarchitecture, MICRO 2015, Waikiki, HI, USA, December 5-9, 2015*, 2015, pp. 383–394.

[16] Y. Yu, W. Xiao, X. He, H. Guo, Y. Wang, and X. Chen, "A stall-aware warp scheduling for dynamically optimizing thread-level parallelism in gpgpus," in *Proceedings of the 29th ACM on International Conference on Supercomputing, ICS'15, Newport Beach/Irvine, CA, USA, June 08 - 11, 2015*, 2015, pp. 15–24.

[17] J. Anantpur and R. Govindarajan, "PRO: progress aware GPU warp scheduling algorithm," in *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, 2015, pp. 979–988.

[18] T. Shabanian, A. Bal, P. Basu, K. Chakraborty, and S. Roy, "Ace-gpu: Tackling choke point induced performance bottlenecks in a near-threshold computing gpu," in *ACM,/IEEE International Symposium on Low Power Electronic Devices*, 2018.

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

[20] *MIAOW GPU*, http://miaowgpu.org.

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

[22] S. K. Khatamifard, M. Resch, N. S. Kim, and U. R. Karpuzcu, "Varius-tc: A modular architecture-level model of parametric variation for thin-channel switches," 2016, pp. 654–661.

[23] M. Lee, S. Song, J. Moon, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Improving gpgpu resource utilization through alternative thread block scheduling," in *20th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2014, pp. 260–271.

[24] E. Krimer, P. Chiang, and M. Erez, "Lane decoupling for improving the timing-error resiliency of wide-simd architectures," in *39th International Symposium on Computer Architecture (ISCA)*, 2012, pp. 237–248.

[25] Y. Oh, M. K. Yoon, J. H. Park, Y. Park, and W. W. Ro, "Wasp: Selective data prefetching with monitoring runtime warp progress on gpus," in *IEEE Transactions on Computers Volume 67 Issue 9*, 2018, pp. 1366–1373.

[26] S. P. Mohanty and D. K. Pradhan, "ULS: A dual-V_{th}/high-kappa nano-cmos universal level shifter for system-level power management," in *ACMJournal on Emerging Technologies in Computing Systems (JETC) Volume 6 Issue 2*, 2010.

[27] Y. Oh, M. K. Yoon, J. H. Park, Y. Park, and W. W. Ro, "New generation of predictive technology model for sub-45nm early design exploration," in *IEEE Transactions on Electron Devices (TED) Volume 53 Issue 11*, 2006, pp. 2816–2823.

[28] *AMD Accelerated Parallel Processing (APP) Software Development Kit*, http://developer.amd.com/sdks/amdappsdk/.

CURRICULUM VITAE

Sourav Sanyal

Submitted Journal Papers

- **Sourav Sanyal**, Prabal Basu, Aatreyi Bal, Sanghamitra Roy, Koushik Chakraborty, **Exploring Warp Criticality in Near Threshold GPGPU Applications using a Dynamic Choke Point Analysis** , submitted to *IEEE Transactions on Very Large Scale Integration (TVLSI).*

Published Conference Papers

- **Sourav Sanyal**, Prabal Basu, Aatreyi Bal, Sanghamitra Roy, Koushik Chakraborty, **Predicting Critical Warps in Near Threshold GPGPU Applications using a Dynamic Choke Point Analysis** , accepted for publication in *IEEE/ACM Design Automation and Test in Europe (DATE) 2019.*

- Krishnendu Sanyal, **Sourav Sanyal**, Amitava Chatterjee, **Multi Objective Cost Sensitive Subspace Reduction Technique for Access Control System**, published in *IEEE International Conference on Wireless Communications, Signal Processing and Networking (WispNET) 2016.*

- Sibsankar Dasmahapatra, Rana Saha, Dipankar Sanyal, Agnimitra Sengupta, Utsav Bhattacharya, **Sourav Sanyal**, **Designing Low-Chattering Sliding Mode Controller for an Electrohydraulic System** , published in *IEEE International Conference on Control, Measurement and Instrumentation (CMI) 2016.*

- Sibsankar Dasmahapatra, Rana Saha, Dipankar Sanyal, Agnimitra Sengupta, **Sourav Sanyal** , **Designing Sliding Mode with Integral Control for Angular Rotation of a Link by Linear Electrohydraulic Actuation**, published in *IEEE India Council International Conference (INDICON) 2015.*

- **Sourav Sanyal**, Krishnendu Sanyal, Amitava Chatterjee, **Cost Sensitive Discriminant Analysis using Swarm Intelligence for Access Control System**, published in *IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN) 2015.*