8-2018

Split Latency Allocator: Process Variation-Aware Register Access Latency Boost in a Near-Threshold Graphics Processing Unit

Asmita Pal
Utah State University

SPLIT LATENCY ALLOCATOR: PROCESS VARIATION-AWARE REGISTER ACCESS LATENCY BOOST IN A NEAR-THRESHOLD GRAPHICS PROCESSING UNIT

by

Asmita Pal

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
Amanda Lee Hughes, 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

Split Latency Allocator: Process Variation-Aware Register Access Latency Boost in a Near-Threshold Graphics Processing Unit

by

Asmita Pal, Master of Science

Utah State University, 2018

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

Operating a Graphics Processing Unit (GPU) at Near Threshold Computing (NTC) domain comes with a significant performance variability as a consequence of process variation (PV). In the emerging era of General Purpose GPUs (GPGPUs), the existence of a large register file is inevitable. This work investigates the increased sensitivity of the GPGPU register file to PV and suggests a dynamic allocation of thread blocks based on register access latency. The variation in maximum operating frequencies of cores in a GPU, is further exploited to hide the excessive long access latencies. The proposed technique optimizes GPU energy consumption by $\sim$35% over an ideal PV-free GPU operating at Super Threshold regime. The area and power overhead for Split Latency Allocator is 1.1% and 1.9% respectively.

(38 pages)

PUBLIC ABSTRACT

Split Latency Allocator: Process Variation-Aware Register Access Latency Boost in a Near-Threshold Graphics Processing Unit

Asmita Pal

Over the last decade, Graphics Processing Units (GPUs) have been used extensively in gaming consoles, mobile phones, workstations and data centers, as they have exhibited immense performance improvement over CPUs, in graphics intensive applications. Due to their highly parallel architecture, general purpose GPUs (GPGPUs) have gained the foreground in applications where large data blocks can be processed in parallel. However, the performance improvement is constrained by a large power consumption. Likewise, Near Threshold Computing (NTC) has emerged as an energy-efficient design paradigm. Hence, operating GPUs at NTC seems like a plausible solution to counteract the high energy consumption. This work investigates the challenges associated with NTC operation of GPUs and proposes a low-power GPU design, *Split Latency Allocator*, to sustain the performance of GPGPU applications.

To the memory of Dadu, who always believed in me.

ACKNOWLEDGMENTS

The completion of this thesis could not have been possible without the assistance of many, who have been by my side over the past two years. First and foremost, I would like to express my sincere gratitude to my adviser, Dr. Koushik Chakraborty and my co-adviser, Dr. Sanghamitra Roy, for their insight, guidance and financial support, throughout my Masters program. I attribute my motivation for engaging in research to them. Also, I would like to thank my committee member, Dr. Amanda Lee Hughes, for her valuable comments on this research. I am also grateful to Dr. Todd Moon for introducing me to the wonderful domain of Neural Networks.

I would like to thank each member of the BRIDGE laboratory for making the journey of research challenging and enlightening. To Aatreyi, for being there for me right from my very first day in Logan, to co-authoring me in my first publication. To Prabal, for patiently listening to my plethora of questions related to research and beyond. I'm grateful to Pramesh, for inspiring me to question our findings at every juncture and supporting me in every project I undertook at USU. To Chidham, for the rigorous brainstorming sessions and critically reviewing my work. Rajesh, for his insights on my writing and for helping me with my presentation in the second semester. I would also thank Atif for helping me understand the work-flow of multi-threaded applications and the expanse of tools involved in a research project. To Sourav for the fun discussions and keeping the lab lively even during deadlines. Lastly, I would thank Shamik for not only introducing me to the world of GPUs but more importantly, being by my side since high school.

I would also express my appreciation for the ECE department and all of the staff members. I would like to acknowledge the efforts of Tricia Brandenburg and Mary Lee Anderson for her guidance with paperwork and all kinds of administrative matters. I'm extremely thankful to Faith Spraktes and Steve Meeker for supporting me through various technical issues.

Without the love and encouragement of my parents, Tapasi Pal and Asoke Kumar Pal, I would not have been able to sustain through my Master's program. In addition to this I would like to thank my friends for making this journey extremely memorable: Bhagyashree, Anusna, Vidita, Srijanee, Sohum, Bhumika, Kanak and Avik. And, thank you, Sayan, for encouraging me to take up new challenges in life and assuring that I will do good. Last but not the least, I would thank a very special person, Anish, for his unconditional support throughout my grad-school and for inspiring me to strive for the best in life.

Asmita Pal

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
CPU	Central Processing Unit
NTC	Near Threshold Computing
STC	Super Threshold Computing
PV	Process Variation
CU	Compute Unit
SLA	Split Latency Allocator
RTL	Register-Transfer Level
RF	Register File
RAL	Register Access Latency
SRAM	Static Random-Access Memory
AMD APP SDK	AMD Accelerated Parallel Processing Software Development Kit
SIMD	Single Instruction Multiple Data
VGPR	Vector General Purpose Registers
IPS	Instructions Per Second
WID	Within-Die
BIST	Built-In Self Test
PS-DWM	Pre-Shifting Domain Wall Memory
DSPL	Decoupled SIMD Pipeline

CHAPTER 1
INTRODUCTION

In the past decade, Graphics Processing Units (GPUs) have managed to grab the attention of the researchers, owing to their immense performance improvement over Central Processing Units (CPUs). The performance improvement in GPUs is induced by extensively exploiting thread level parallelism in highly parallel applications. However, this upswing is constrained by large power consumption in GPUs [1]. Contrarily, Near Threshold Computing (NTC) has emerged as the promising energy-efficient design point, as opposed to Super Threshold Computing (STC) [2]. At NTC, the supply voltage is marginally higher than the threshold voltage. The reduction in power consumption at NTC is a result of both reduced supply voltage and consequent reduction in operating frequency. Inevitably, GPUs operating at NTC have evolved as an intriguing research topic [3].

Despite the energy efficiency offered by NTC, there are several design challenges in this regime. The biggest bottleneck at NTC comprises of 10X performance degradation, accompanied with extensive process variation (PV) sensitivity [4]. Though performance degradation is tackled by the parallelism offered by GPUs, the spatial expanse of GPUs make them more vulnerable to PV. To support the massive thread level parallelism GPUs employ latency-hiding with the aid of fast context-switching between threads. A huge register file is required to maintain the latency-hiding ability. Therefore the register file becomes a potential soft target for PV [5]. The most prominent manifestation of PV in GPU register files is the variability in the access latency.

This work explores PV sensitivity of register access latency in GPUs. PV can affect the latency either way, exacerbating the access latency difference between the slowest and fastest accessible register file. Further, PV can affect the operating frequency of Compute Units (CUs) thereby inducing significant variability in their performance. Consequently, a PV-agnostic mapping of register files to CUs can remarkably degrade the processor performance.

It is therefore essential to dynamically assess the effects of PV on register access latency as well as CU frequency.

1.1 Organization

The rest of the thesis is organized as follows:

- **Background:** The contemporary research related to energy-efficient GPU design is explored, to validate why a different approach needs to be adopted for GPUs operating at NTC (Chapter 2).

- **Motivation:** The factors plaguing the performance of modern GPGPU applications are investigated. Following which, it is observed that there is a the steady improvement in performance of GPGPU applications when the effect of those degrading factors is minimized (Chapter 3).

- **Split Latency Allocator Design:** Split Latency Allocator, a low overhead technique is proposed, to mitigate the exacerbating effect of PV in register files of a GPU. This technique leverages the latency detection in a wavefront to decide their mapping to CUs to improve the performance and power consumption of NTC GPU (Chapter 4).

- **Cross-layer Methodology:** In Chapter 5 the tools and simulation parameters involved in developing the proposed design across multiple layers is described to establish the feasibility of the scheduler design.

- **Experimental Results:** A comprehensive analysis of SLA, is discussed in Chapter 6. The proposed scheme achieves an improvement in GPU energy by 35.5% over an ideal PV-free GPU operating at STC. A baseline GPU RTL synthesis, augmented with SLA gives marginal area and power overheads of 1.1% and 1.9%, respectively.

- **Conclusion:** Chapter 7 highlights the contribution of this research towards the greater GPU research community.

CHAPTER 2
BACKGROUND

Previous works in the field of energy-efficient GPU design primarily focus on the STC regime. However, recent researches in this area, related to this work, can be broadly classified into two categories: (a) remodelling GPU register file architecture for energy-efficient GPGPUs, and (b) mitigating process variation impact in GPGPUs. In the first category, Jing et al. proposed an embedded-DRAM architecture to reduce the ever increasing cost and power overheads of SRAM based register files [6]. Majeed et al. exploited the inter-access latency of GPGPU register files to propose a tri-modal control unit [7]. Mao et al. proposed a nonvolatile racetrack memory to improve the scalability of GPGPUs [8]. Jeon et al. presented a register file virtualization technique to reduce the physical area as well as power consumption of architected register files [5]. In the second category, Liang et al. proposed variable latency register files to mitigate the effects of process variation in CPUs [9]. Basu et al. designed a dynamic parallelization scheme to address the performance variance introduced in GPGPUs due to process variation in the NTC regime [3]. For GPUs capable of incorporating spatial multitasking, Aguilera et al. proposed the assignment of variable resources based on the nature of applications [10]. The effects of PV on CPUs have been addressed in several works. Teodorescu et al. proposed a linear programming algorithm to tackle the differential static power due to within-die PV [11]. The power consumption of each cache was reduced by sourcing them at individual voltages, according to their vulnerability to PV, in the Variation Trained Drowsy Cache [12]. Seo et al. examined the effects of accentuated PV sensitivity in near-threshold operated SIMD architectures, where the number of critical paths increased manifold [13]. ReCycle applies time slack management in pipelines to overcome the effects of PV on pipeline delays [14]. Further, several researchers have proposed architectural models to counteract parametric variation at NTC [15–17]. GPUs and CPU-GPU heterogeneous systems exhibit a marked difference in architecture and operation. With an increased usage of these systems, it is crucial to check the portability of existing techniques and in fact, come up with newer designs, to curb the effects of variability and make them energy-efficient.

However, to the best of my knowledge, this is the first work to explore the effects of process variation at NTC on register access latency in GPGPUs.

CHAPTER 3
MOTIVATION

This chapter, shows that the register file is a potential soft target for process variation in a GPU at NTC (Section 3.1). Using a cross layer methodology (Section 3.2), it is observed that any change in access latency, due to PV, can affect the overall performance of a GPU (Section 3.3). Additionally, there is a steady improvement in performance with shorter register access latency.

3.1 NTC-GPU limitations

Individually, GPUs and NTC have emerged as promising design areas. However, their combined potential as a high performance and energy-efficient design paradigm has hardly been explored [3]. NTC operation reduces power consumption but hampers performance due to aggravated PV sensitivity. The following sections address the effects of this variability and its potential in evolving as a performance bottleneck in GPUs.

3.1.1 Process Variation in NTC circuits

The large spatial spread of GPUs makes them inherently vulnerable to PV. The degree and effects of PV on different components of a GPU can be largely different. However, the most prominent manifestation of PV at NTC is in the form of transistor delay variability. The progressive miniaturization of technology nodes further adds to this variability [18]. As a result, PV transpires into a significant performance bottleneck in multicore architectures like GPU.

3.1.2 Register File(RF) Latency in GPU

The size and expanse of RFs in modern GPUs make them prone to PV. The shallow logic depth and large number parallel critical paths of SRAM structures enhances their vulnerability [19]. The SRAM access, read and write times can display marked variation from cell to cell, owing to the effects of PV. The cumulative latency of SRAM read/write access is referred to as register access latency (RAL). This PV induced RAL variability at NTC makes the RFs an intriguing design optimization challenge in NTC-GPUs. Section 3.3 explores how speeding up the overall access latency preserves the performance of a GPU.

3.1.3 Criticality in GPGPU applications

GPUs exploit thread-level parallelism to mask the performance bottlenecks of a single thread. AMD defines a wavefront as the set of threads executing the same instruction with different data. For certain applications, there is an execution time disparity between wavefronts of the same work group[1] and the execution time is dominated by the slowest wavefront. All the wavefronts in a work group need to finish execution before the next work group can be scheduled and the slowest wavefront in the work group imposes a wait time on other wavefronts [20]. Therefore, it becomes an indispensible determinant of overall system performance and for the rest of this work, it is defined as the *critical wavefront*. However, these critical wavefronts do not arise due to lack of synchronization among CUs. The criticality factor primarily springs from varying latencies and pipeline hazards within a work group. Thread criticality is a similar approach for multi-threaded CPU applications to tackle load imbalance on constrained resources, which in turn determines the overall multi-core performance [21]. In the shift from STC to NTC, access latency variation is exacerbated by the increasing parallel bottlenecks at NTC. Hence, RAL would be a controlling factor in determining the critical wavefronts, for a certain timeline[2] in a GPU.

3.2 Methodology

The approach adopted for this work is briefly outlined here. The microarchitectural model of VARIUS-NTV is used for modelling PV in the NTC regime [4]. AMD's Southern

[1]Work Group refers to a group of threads/work-items. It is similar to a thread block defined by NVIDIA. Henceforth, these terms are used interchangeably.

[2]GPGPU applications use timelines instead of intervals, as they capture overlap and illustrate critical path.

Islands architecture based GPU - HD Radeon 7970 model is simulated on Multi2Sim 4.2 [22]. Pre-compiled benchmarks from AMD APP SDK [23] suite is run on this GPU model. The register access latency is 8 cycles for PV-free STC and NTC. Due to the effect of PV the frequency at NTC is reduced upto 75% [24]. Compute Units are scaled upto 4 times at NTC as compared to STC.

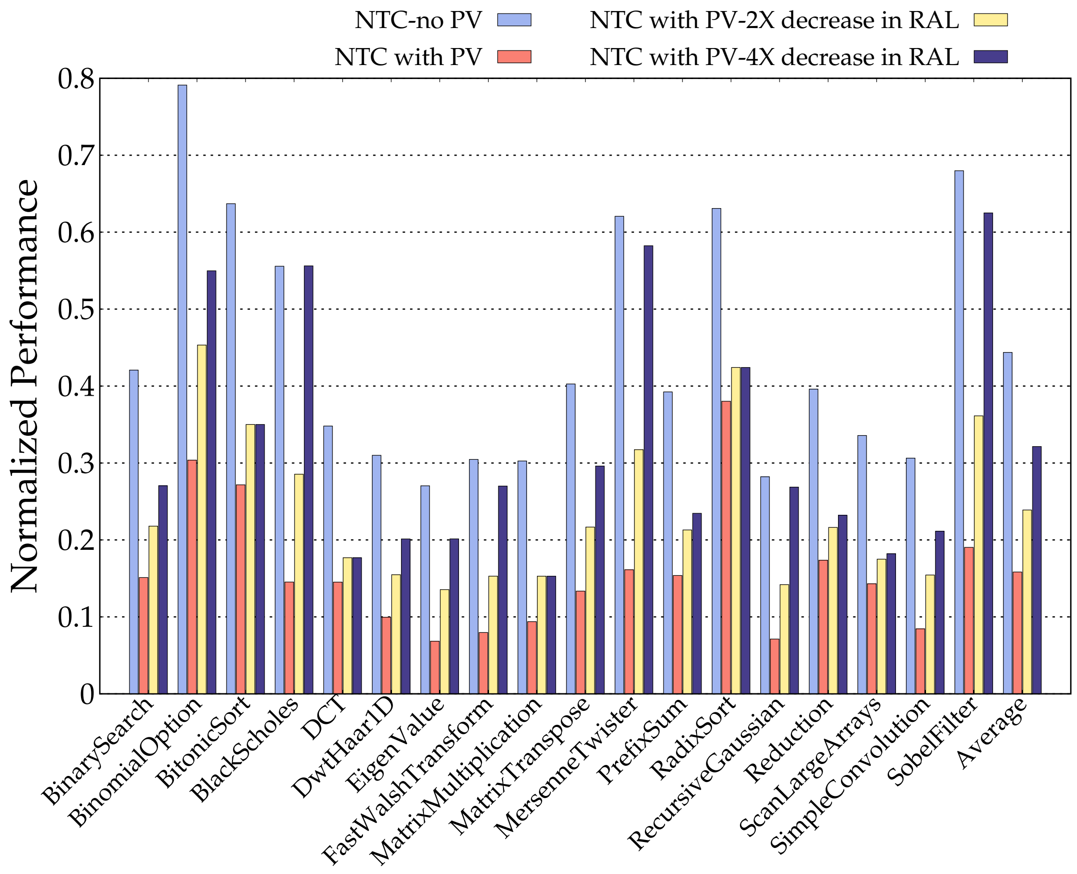

Fig. 3.1: NTC-GPU performance characteristics for : *NTC-no PV; NTC with PV; NTC with PV and 2X improvement in register access latency ; NTC with PV and 4X improvement in register access latency (Normalized to STC GPU).*

3.3 Performance Analysis

Figure 3.1 shows how register access latency can be exploited to improve the performance of the GPGPU applications. The GPGPU application performance is modelled as the inverse of the execution time. All performance values are normalized to a PV-free STC GPU. The applications display maximum variation in performance when run on an NTC GPU affected by PV. The cases of 2X and 4X improvement in RAL is considered, for the PV-affected GPU. As observed from figure 3.1, there is a pronounced performance uplift. As observed from figure 3.1, a pronounced performance uplift is observed. The average performance goes up by ∼33% when the register access latency is reduced to 25% of PV-induced value(16 cycles in this work). Reducing the access latency further, by a factor of 8X, shows no appreciable change in performance, as compared to its 4X counterpart. The minimum time taken for access by an SRAM limits this amelioration. With the RAL improvement, *BlackScholes, MersenneTwister* and *SobelFilter* achieve almost comparable performance to an NTC system without PV. PV being a greater concern in NTC than in STC [25], Figure 3.1 clearly shows the scope of register access latency improvement in high performance GPUs operating at NTC. To recuperate this performance loss, it is essential to devise a dynamic paradigm, which can overcome the aforementioned limitations. For further design, this work will be continued to contrive a technique which can give us an improved performance, without jeopardising the power efficiency offered by a GPU operating at NTC.

CHAPTER 4
SPLIT LATENCY ALLOCATOR DESIGN

4.1 Overview

This chapter presents a circuit-architectural technique, Split Latency Allocator (SLA), to adaptively overcome the limitations presented in Section 3.1. In Section 4.1, a brief overview of SLA is given, followed by Sections 4.2 and 4.3 which highlight the key aspects of the technique. It concludes with the allocator working principle in Section 4.4.

Figure 4.1 gives an overview of SLA. The proposed scheme, works in three stages. First, it categorizes registers according to their access latencies. Subsequently, it performs a dynamic assessment of the compute unit frequencies and accordingly assigns independent clocks to them. Finally, it maps the wavefronts, to the next available CU, according to the access latency of the register file associated with it. The SLA efficacy lies in its dynamic nature of monitoring the register file usage. In the following sections, each stage of the proposed technique is described in detail.

4.2 Access Latency Categorization

Post-fabrication, PV affects register files diversely. While some register files may experience increased access latency, others may speed up. This randomness of PV effects cannot be predicted at design time. To address this, a dynamic runtime categorization of register files based on their access latencies is proposed. For a register file with multiple read and write ports, the entry speed will deviate considerably. Using the RF port entry speed information reported by a BIST unit [26], it is observed that 80% RFs have fast entries and 20% have slow entries. The categorization of RFs allows for cognizant mapping of the RFs to CUs, such that there is maximum frequency benefits.

In a GPU, each SIMD unit has Vector General Purpose Registers (VGPRs) containing

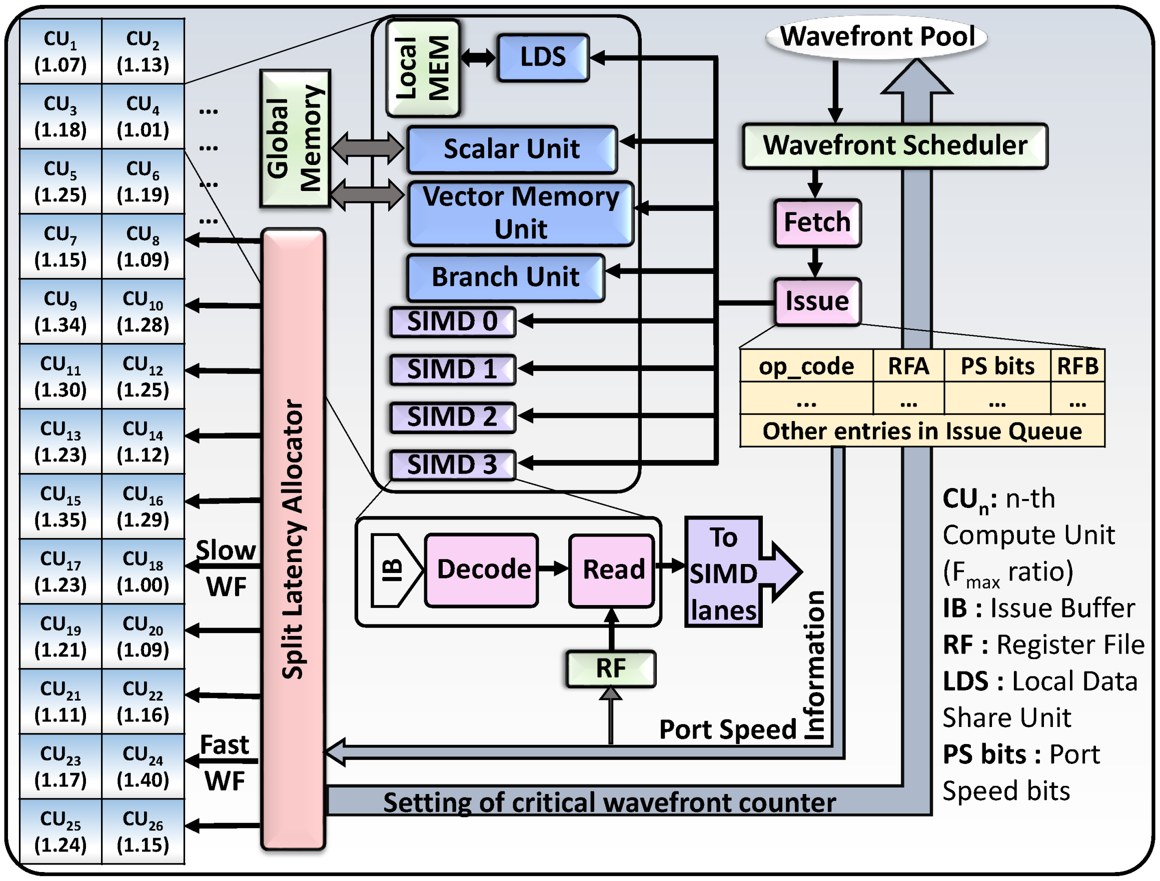

Fig. 4.1: Overview of Split Latency Allocator : *CUs are individually clocked with frequency ratio relative to that of the slowest CU. The wavefront categorization is done when an instruction is waiting in the Issue Queue, using the port speed bits. These categorized wavefronts are then mapped to CUs, where the CU to be chosen is decided by the range of operating frequency values from Equation 4.1 (Section 4.4.3).*

64 lanes that are upto 32-bit wide. When a wavefront is scheduled, a single operand read in an instruction reaches out to 64 registers, corresponding to each thread/work-item in the wavefront. The dominance of one slow register in the SIMD unit extends the register access latency of the entire CU. This addition of stalls incurs an average Instructions Per Second (IPS) loss of 14% (discussed in Section 6.1). To impede the IPS loss, the PV induced frequency variation in CUs needs to be taken into account. The upcoming section outlines a technique to address this issue.

4.3 Individual CU Clocks

Dynamically clocking individual CUs makes room for managing the performance variability among them. The effect of PV makes the frequency of the fastest CU upto 40% faster than the slowest CU [27]. The maximum operating frequency(F_{max}), for each CU is decided by the slowest SIMD unit in that CU. The F_{max} ratio for each CU is assigned relative to the slowest CU. This performance variability affects the power performance negatively. While faster CUs consume greater power owing to their frequency, slower CUs affect the overall execution time. Hence, a global frequency for all the CUs will dwindle the effective performance of a GPU, as it will operate at the frequency of the slowest Compute Unit. When a GPU is operated at NTC, this frequency variation in CUs becomes more prominent. Individual CU clocking allows maximum throughput even under RAL variations. Additionally, individual CU-clocking is a viable technique to bypass the effects of PV, as the thread blocks in each CU have rarefied interactions with other CUs, impairing the need for synchronization. Therefore, a faster CU can independently execute more thread blocks than a slower CU. The challenge of dynamically mapping of RFs to CUs with frequency variations is resolved by the adaptive nature of SLA.

4.4 Split Latency Allocator

Split Latency Allocator is faced with certain design challenges to eliminate the performance deficit caused by RAL variation and inefficient usage of CU resources in a given time line. The SLA operation is outlined next, using the strategies mentioned in Section 4.2 and Section 4.3.

4.4.1 Boost Dilemma

In Chapter 3 it was observed that a static increment or boost in the RAL can achieve high performance gain. However, uniform frequency modeling for all CUs dilutes this gain. This contradiction necessitates a PV-aware CU-allocation model, in addition, to RAL variation management, to ensure an overall performance benefit. SLA addresses this need. It resolves the boost dilemma by a critical-path-counter based scheduling. The detailed operation of SLA is discussed in Section 4.4.3.

4.4.2 Critical Wavefront Priority

The designation of priority counter to wavefronts essentially deals with the boost dilemma. As discussed in Section 4.3, the key to better performance is executing more thread blocks on a faster CU. The varying access latency of register files is leveraged to define slow and fast wavefronts. The slowest wavefronts in a thread block are designated as critical wavefronts. These critical wavefronts have longer execution time due to their longer RAL. SLA uses a counter based on critical wavefronts to schedule them in different CUs. This counter decides the priority of the wavefront to be scheduled and equalizes the execution time of all threads across a CU. A smaller value indicates that the wavefront needs more time resource. The counter has the same value as the ratio of slow and fast RFs used in Section 4.2.

4.4.3 SLA operation

When an instruction is issued, the port speed information is pulled up along with the register entry address. This allows us to classification as a fast or slow entry. SLA then decides the critical wavefronts from each CU and assigns a counter as discussed in Section 4.4.2. The use of individual CU clocks imparts different frequencies to each core, expressed by the F_{max} ratio stated in Section 4.3. With the knowledge of wavefront priority from the counter, the SLA maps the faster wavefronts to the cores with F_{max} ratio greater than a value decided by Equation (4.1)

$$ratio_{fast} = 1 + percentage\ of\ slow\ RFs \tag{4.1}$$

For example, for 80% fast and 20% slow entries, the faster wavefronts are mapped to CUs with F_{max} ratio greater than 1.2. Thus, the fast CU then becomes available for the next thread block in the queue and the critical wavefronts do not block the hardware resources. SLA assigns the critical wavefronts to the slower cores, which in turn, gives them more time resource to finish their execution. As per this technique, 80% of the thread blocks are executed faster than other thread blocks, owing to their operation in faster cores. The remaining 20% continue to run in the slower cores with F_{max} ratio less than 1.2. SLA is specifically effective when the thread blocks mapped to a certain CU have no critical wavefronts. Thus, the jeopardizing effects on performance due to critical path are successfully alleviated, by hiding the long latency dominated register accesses.

CHAPTER 5
METHODOLOGY

This chapter brings out the cross-layer methodology to validate the feasibility of the proposed design. Figure 5.1 depicts the multiple layers (device, architecture and circuit layer) which are described in detail next.

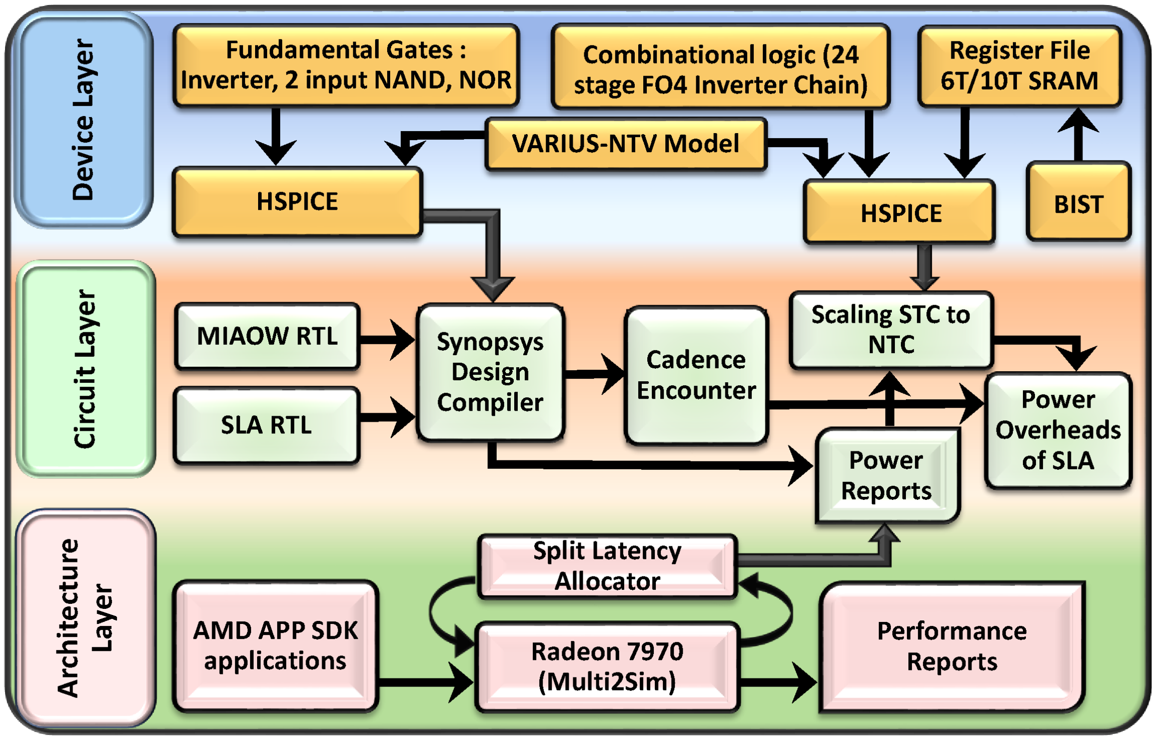

Fig. 5.1: Cross-layer Methodology.

5.1 Device Layer

In this study, VARIUS-NTV [4] is used to model a PV-affected GPGPU register file. The WID correlation distance coefficient ϕ is set to 0.5 and V_{th}'s σ^{sys} = 3.2% - 6.4% of the nominal V_{th} value. A 24-stage fan-out-of-4(FO4) inverter chain is used to model the combinational logic used in GPUs. The STC and NTC GPU register files are represented using a 6T-SRAM cell and 10T-SRAM cell [28], respectively, with provision for implementing BIST [26]. The RAL variation distribution so obtained is fed to the architectural simulator. The F_{max} of each CU is obtained by simulating with a 32nm technology node [29]. HSPICE simulations for the 32nm node, on fundamental gates and circuits, gives us the power consumption at NTC, with the aforementioned simulation parameters.

5.2 Architecture Layer

The architectural simulations, for this work, are performed on an augmented Multi2Sim [22] supporting independent operating frequency for each Compute Unit. The frequency variation caused by 80% variable latency technique is modeled into the simulator where the F_{max} of each CU is obtained using the *within-die*(WID) PV model outlined in Section 5.1. For each time line, based on the overlap and critical paths, the dynamic allocation of registers is done. The kernel is executed for 1 iteration for 21 benchmarks from the AMD APP SDK [23] suite, considering all 4 wavefront assignment strategies to CUs as discussed in Section 4.4. The simulations of each timeline gives us the performance accomplished by a certain application using SLA. Power consumption is calculated by combining the cycle level usage information from Multi2Sim with the dynamic and leakage power characteristics of the synthesized hardware. Table 5.1 enumerates the key hardware configuration parameters used for STC and NTC GPUs. At NTC, the frequency is scaled down by a factor of 4X and the number of available compute units is increased by the same factor. During migration from STC to NTC domain, the CU usage for different benchmarks reflect that the global memory configuration remains unaltered. It is acquired from state-of-the-art GPGPU architecture, Southern Islands, developed by AMD.

Parameter	**STC**	**NTC**
CU Frequency	1GHz	250MHz
No. of CUs	32	128
Register Access Latency	8 cycles	16/8/4 cycles
Supply Voltage	0.85V	0.45V
Process Node	32nm	32nm
L2 cache	8x768KB ; 16-way; latency:20 ns	8x768KB ;16-way; latency:20 ns
Global Memory	B/W: 264GB/s ; latency$\sim$300 ns	B/W: 264GB/s ; latency$\sim$300 ns
Wavefront Scheduling Policy	Round-Robin	Critical-Wavefront-Aware

Table 5.1: **GPU configuration.**

5.3 Circuit Layer

The power obtained from Section 5.2 is scaled to NTC range using the device-layer simulation results. A reference GPU RTL [30] is synthesized using Synopsys Design Compiler to estimate the power consumption of SLA and the initial GPU design. This GPU RTL has close resemblance to the baseline GPU used in this paper. It is augmented to implement the proposed technique and thereby evaluate the overheads of SLA.

CHAPTER 6
EXPERIMENTAL RESULTS

This chapter demonstrates a comprehensive analysis of SLA on the NTC GPU. The efficacy of SLA based on IPS for the various stages in the design is depicted in Section 6.1. For different degrees of PV affecting the RAL, the performance variation of the proposed technique is studied in Section 6.2. Various comparative schemes (Section 6.3), and their corresponding performance and energy consumption is analyzed (Section 6.4 and 6.5). Finally, the hardware implementation overheads of SLA are outlined in Section 6.6.

6.1 SLA Efficacy

Figure 6.1 shows the improvement in Instructions Per Second, due to SLA as opposed to a static boost in RAL. Also, when solely RAL categorization is implemented, IPS is lower than that of SLA. The IPS values for SLA reflect the runtime efficacy for this technique, primarily due to individual clocking of each CU compared to a global GPU clock. The IPS is normalized to that of an ideal PV-free STC GPU. This effective gain shows that SLA exploits the time resource for slow wavefronts and executes a greater number of fast wavefronts in a given timeline.

6.2 Sensitivity Analysis

As stated in Section 4.2, a single ratio is obtained for fast and slow entries of RFs. Different degrees of PV affecting the register access latency, will alter the ratio of fast and slow entries. Figure 6.2 presents the comparative performance characteristics of SLA for four different access latency categorizations. It is observed that SLA performance decreases for some benchmarks with the increase in the ratio of slow RFs. For benchmarks like *DCT* and *Reduction*, considerable performance variation is not exhibited. This anomaly can be leveraged on the varying number of critical wavefronts in a thread block.

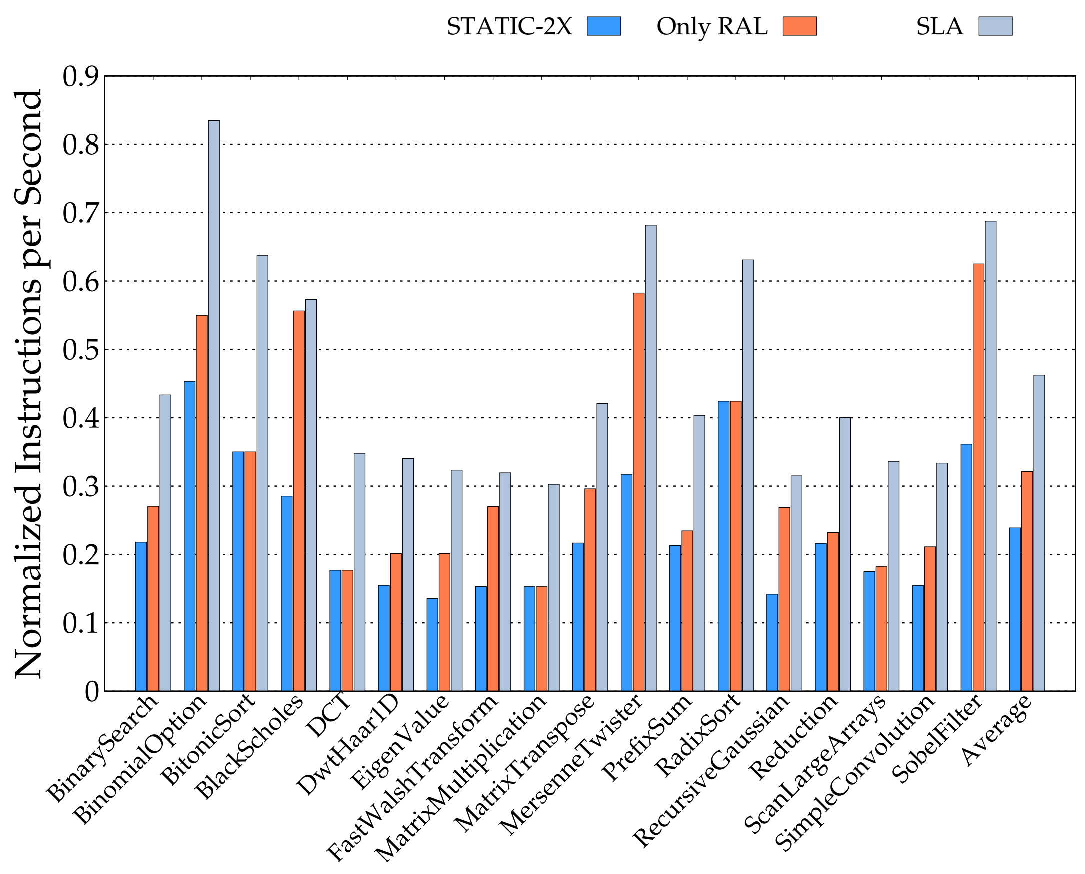

Fig. 6.1: IPS Comparison for Static Increment vs SLA-Normalized to STC GPU (Higher is better).

6.3 Comparative Schemes

- **Pre-Shifting DWM (PS-DWM):** This scheme averages the predictive shift policies, while using a Domain Wall Memory(DWM), by exploiting register locality across threads, as suggested by [31]. A history table keeps a record of registers accessed and shifts track heads, in the next occurence of same instruction.

- **Decoupled SIMD Pipeline (DSPL):** A decoupled SIMD pipeline is used for detecting timing violation error probabilities, independent of adjacent lanes. This technique relies on a stall based recovery for each lane similar to the approach proposed in [32].

- **SLA:** This is the proposed technique, comprising register file categorization based on access latency followed by dynamic mapping of wavefronts to individually clocked

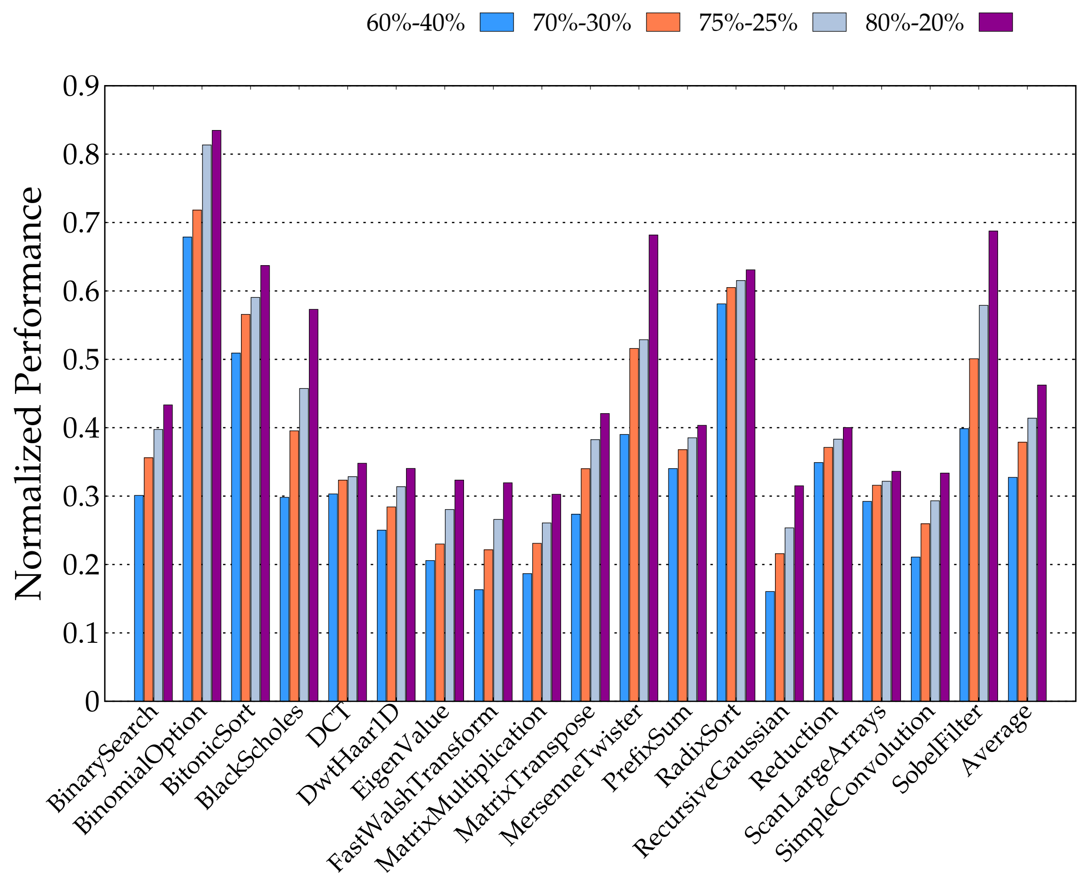

Fig. 6.2: Performance Variation with change in Fast vs Slow Port Ratio- Normalized to STC GPU(Higher is better).

CUs.

6.4 Performance Reports

Figure 6.3 exhibits the performance variance of the different comparative schemes discussed in Section 6.3. The performance of all the schemes are normalized with respect to a PV-free STC GPU. SLA performs better than PS-DWM and DSPL for most of the GPGPU applications. SLA efficacy can be attributed to the efficient utilization of hardware resources, as a result of the critical wavefront mapping strategy. Due to the track head shift operation overhead, SLA delivers an average performance improvement of 16% over PS-DWM. Even the pipeline recovery mechanism of DSPL, gives us 15% lower performance on an average, for the NTC GPU. Due to larger number of cores at NTC, and the

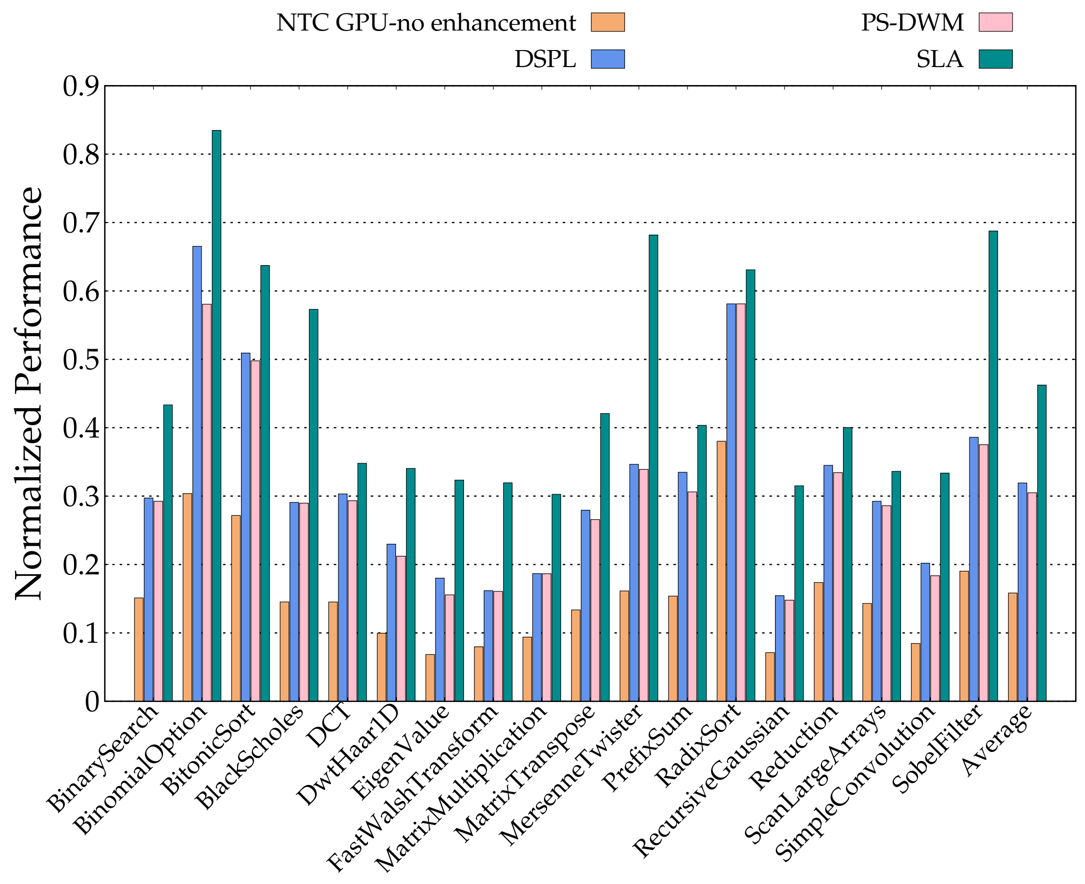

Fig. 6.3: Comparative Performance Analysis (Higher is better).

increasing SIMD units the decoupling technique has degraded performance than SLA. All the GPGPU applications perform better in an SLA-enhanced NTC GPU. On an average, the performance improvement is lower than that of a STC GPU, but there is a noticeable improvement of 31% over that of a NTC GPU affected by PV and no enhancement. This performance loss is compensated for an energy consumption improvement, which is discussed next.

6.5 Energy-Consumption Reports

Figure 6.4 depicts the energy consumption comparison of the schemes discussed in Section 6.3. Both DSPL and PS-DWM consume more energy than SLA, owing to the static power constraints prevalent in NTC operation. The distribution of wavefronts by

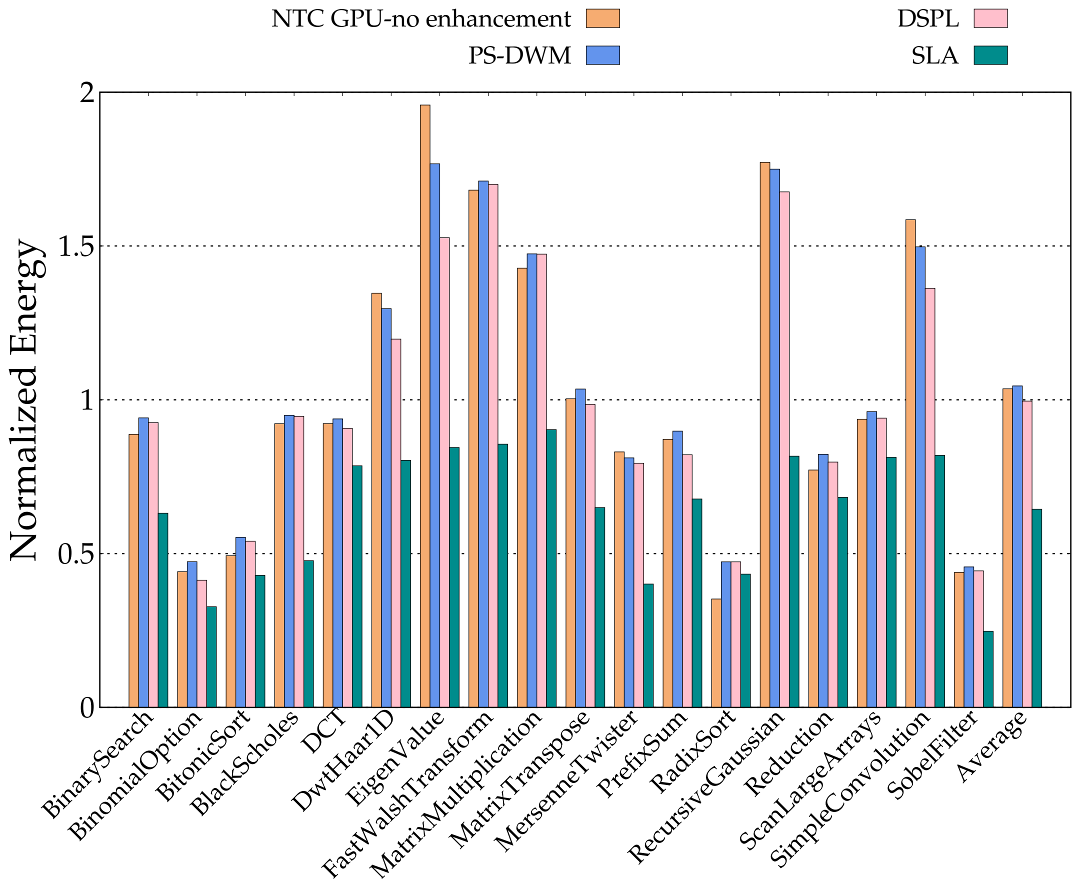

Fig. 6.4: Energy Consumption Analysis-Normalized to STC GPU (Lower is better).

SLA alleviates this problem, leading to overall reduction in energy consumption. For some benchmarks such as, *RadixSort, Reduction* and *ScanLargeArrays*, SLA does not reflect much energy gain over the other techniques, when compared to the NTC GPU-with no enhancement. This aberration rises because the number of critical wavefronts is not very high and SLA behaves like a default scheduler. The slow register file usage is fairly minimal in these cases. SLA shows about 35.5% improvement in average energy consumption over the baseline STC GPU.

6.6 Hardware Overheads

The hardware overhead comes from the latency adaptive infrastructure of SLA. Behavioral change of the wavefront schedular has minimal area overhead, owing to the existing issue priority mechanism of the scheduler. SLA shows an energy improvement of 35% over DSPL and 38% over PS-DWM. Using synthesized hardware (see Section 5) the area and power overheads for SLA implementation are estimated as 1.1% and 1.9% respectively, as compared to the baseline GPU.

CHAPTER 7
CONCLUSION

The various performance limitations of a GPU operating at NTC are analyzed in this thesis and a cross-layer approach is improvised to mitigate their effects. The proposition of SLA is based on the high degree of access latency variation exhibited by a PV-induced NTC GPU. SLA relies on efficient allocation of registers based on their access latency and exploits the existing GPU principles to ensure better performance in comparison to other contemporary schemes. When compared to other contemporary techniques, SLA performs 32% better, on an average for a PV-affected NTC GPU. SLA thus, emerges as a promising design with a staggering 35% improvement in energy consumption as compared to an ideal PV-free, globally clocked STC GPU.

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,” in *ISPASS*, April 2013.

[2] R.G.Dreslinski, M.Wieckowski, D. Blaauw, D.Sylvester, and T.Mudge, “Near-threshold computing: Reclaiming moores law through energy efficient integrated circuits,” in *Proc. IEEE*, Feb. 2010.

[3] P. Basu, H. Chen, S. Saha, K. Chakraborty, and S. Roy, “Swiftgpu: Fostering energy efficiency in a near-threshold gpu through tactical performance boost,” in *Proc. of DAC*, 2016.

[4] 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 *DSN*, 2012, pp. 1–11.

[5] H. Jeon, G. S. Ravi, N. S. Kim, and M. Annavaram, “Gpu register file virtualization,” in *Proc. of MICRO*, 2015, pp. 420–432.

[6] N. Jing, Y. Shen, Y. Lu, S. Ganapathy, Z. Mao, M. Guo, R. Canal, and X. Liang, “An energy-efficient and scalable edram-based register file architecture for gpgpu,” in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, 2013, pp. 344–355.

[7] M. Abdel-Majeed and M. Annavaram, “Warped register file: A power efficient register file for gpgpus,” in *HPCA*, 2013, pp. 412–423.

[8] M. Mao, W. Wen, Y. Zhang, Y. Chen, and H. H. Li, “Exploration of gpgpu register file architecture using domain-wall-shift-write based racetrack memory,” in *Proc. of DAC*, 2014, pp. 1–6.

[9] X. Liang and D. Brooks, “Mitigating the impact of process variations on processor register files and execution units,” in *Proc. of MICRO*, 2006, pp. 504–514.

[10] 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 *Proc. of DATE*, 2014, pp. 1–6.

[11] R. Teodorescu and J. Torrellas, “Variation-aware application scheduling and power management for chip multiprocessors,” in *Proc. of ISCA*, 2008, pp. 363–374.

[12] A. Sasan, K. Amiri, H. Homayoun, A. Eltawil, and F. Kurdahi, “Variation trained drowsy cache (vtd-cache): A history trained variation aware drowsy cache for fine grain voltage scaling,” *TVLSI*, vol. 20, pp. 630–642, 2012. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5712204

[13] 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,” in *Proc. of DAC*, 2012, pp. 980–987.

[14] A. Tiwari, S. R. Sarangi, and J. Torrellas, “Recycle: pipeline adaptation to tolerate process variation,” in *Proc. of ISCA*, 2007, pp. 323–334.

[15] U. Karpuzcu, N. S. Kim, and J. Torrellas, “Coping with parametric variation at near-threshold voltages,” *IEEE Micro*, vol. 33, no. 4, pp. 6–14, July 2013.

[16] J. Torrellas, N. S. Kim, and R. Teodorescu, “Parameter variation at near threshold voltage: The power efficiency versus resilience tradeoff,” University of Illinois, Tech. Rep., 2012.

[17] X. Liang, G.-Y. Wei, and D. Brooks, “Revival: A variation-tolerant architecture using voltage interpolation and variable latency,” in *Proc. of ISCA*, 2008, pp. 191–202.

[18] 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,” *Proc. IEEE*, vol. 98, no. 2, pp. 215–236, 2010.

[19] K. Bowman, S. Duvall, and J. Meindl, “Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration,” *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 2, pp. 183 –190, feb 2002.

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

[21] K. D. Bois, S. Eyerman, J. B. Sartor, and L. Eeckhout, “Criticality stacks: identifying critical threads in parallel programs using synchronization behavior,” in *Proc. of ISCA*, 2013, pp. 511–522.

[22] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, “ Multi2Sim: A Simulation Framework for CPU-GPU Computing ,” in *PACT*, Sep. 2012.

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

[24] C. Silvano, G. Palermo, S. Xydis, and I. S. Stamelakos, “Voltage island management in near threshold manycore architectures to mitigate dark silicon,” in *DATE, Dresden, Germany, March 24-28, 2014*, 2014, pp. 1–6.

[25] M. Seok, G. Chen, S. Hanson, M. Wieckowski, D. Blaaw, and D. Sylvester, “Cas-fest 2010: Mitigating variability in near-threshold computing,” in *J. Emerg Selec. Topics Cir. Sys*, vol. 1, no. 1, 2011, pp. 42–49.

[26] M. H. Tehranipour, Z. Navabi, and S. M. Fakhraie, “An efficient BIST method for testing of embedded srams,” in *Proc. of ISCAS*, 2001, pp. 73–76.

[27] J. Lee, P. P. Ajgaonkar, and N. S. Kim, “Analyzing throughput of gpgpus exploiting within-die core-to-core frequency variation,” in *ISPASS*, 2011, pp. 237–246.

[28] B. Calhoun and A. Chandrakasan, “A 256-kb 65-nm sub-threshold sram design for ultra-low-voltage operation,” in *J. of Solid-State Circ.*, vol. 42, no. 3, March 2007, pp. 680–688.

[29] W. Zhao and Y. Cao, “Predictive technology model,” June 2012. [Online]. Available: http://ptm.asu.edu/

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

[31] E. Atoofian, “Reducing shift penalty in domain wall memory through register locality,” in *Proc. of CASES*, 2015, pp. 177–186.

[32] E. Krimer, P. Chiang, and M. Erez, “Lane decoupling for improving the timing-error resiliency of wide-simd architectures,” in *Proc. of ISCA*, 2012, pp. 237–248.