5-2016

SSAGA: Streaming Multiprocessors (SMs) Sculpted for Asymmetric General Purpose Graphics Processing Unit (GPGPU) Applications

Shamik Saha
Utah State University

SSAGA: STREAMING MULTIPROCESSORS (SMS) SCULPTED FOR ASYMMETRIC GENERAL PURPOSE GRAPHICS PROCESSING UNIT (GPGPU) APPLICATIONS

by

Shamik Saha

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

of

MASTER OF SCIENCE

in

Computer Engineering

Approved:

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

UTAH STATE UNIVERSITY
Logan, Utah

2016

ABSTRACT

SSAGA: Streaming Multiprocessors (SMs) Sculpted for Asymmetric General Purpose Graphics Processing Unit (GPGPU) Applications

by

Shamik Saha, Master of Science

Utah State University, 2016

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

The evolution of the Graphics Processing Units (GPUs) over the last decade, has reinforced general purpose computing while sustaining a steady performance growth in graphics intensive applications. However, the immense performance improvement is generally associated with a steep rise in GPU power consumption. Consequently, GPUs are already close to the abominable *power wall*. With a massive popularity of the mobile devices running general-purpose GPU (GPGPU) applications, it is of utmost importance to ensure a high energy efficiency, while meeting the strict performance requirements.

In this work, we demonstrate that, customizing a Streaming Multiprocessor (SM) of a GPU, at a lower frequency, is significantly more energy efficient, compared to employing Dynamic Voltage and Frequency Scaling (DVFS) on an SM, designed for a high frequency operation. Using a system level Computer Aided Design (CAD) technique, we propose *SSAGA - Streaming Multiprocessors Sculpted for Asymmetric GPGPU Applications*, an energy efficient GPU design paradigm. SSAGA creates architecturally identical SM cores, customized for different voltage-frequency domains.

(46 pages)

PUBLIC ABSTRACT

SSAGA: Streaming Multiprocessors (SMs) Sculpted for Asymmetric General Purpose Graphics Processing Unit (GPGPU) Applications

Shamik Saha

The evolution of the Graphics Processing Units (GPUs) over the last decade, has reinforced general purpose computing while sustaining a steady performance growth in graphics intensive applications. However, the immense performance improvement GPUs are already close to the abominable *power wall*. With a massive popularity of the mobile devices running general-purpose GPU (GPGPU) applications, it is of utmost importance to ensure a high energy efficiency, while meeting the strict performance requirements. Hence, a flexible GPU design paradigm is the need of the hour.

In this work, using a system level Computer Aided Design (CAD) technique, we propose *SSAGA - Streaming Multiprocessors Sculpted for Asymmetric GPGPU Applications*, an energy efficient GPU design paradigm.

This thesis is lovingly dedicated to my parents, Susmita Saha and Subir Saha, and to my brother Suvro Saha. Their love, support and encouragement have sustained me throughout my life.

ACKNOWLEDGMENTS

I would like to express my gratitude to all the people who have helped me on my journey as a Masters student. This journey would have been impossible without the advice, guidance and financial support of my advisor, Dr. Koushik Chakraborty and my co-advisor Dr. Sanghamitra Roy. They have given me the opportunity to work on various research projects which has shaped the way I analyze situations both in my professional and personal life. I would also like to thank my committee member, Dr. Jacob Gunther, for his valuable comments and insights on my research.

I am grateful to all the members of the Bridge Lab for their support, guidance and friendship. I am extremely thankful to both Prabal and Chidham for being excellent roommates and friends. Prabal and Chidham ensured that I feel at home in the lab and always reviewed my work critically in order to make it better. To Atif, for coming up with the IoT project and critically reviewing the work before every submission. To Hu, for the excellent introduction to GPUs and the work we did on SwiftGPU. Rajesh, for helping me out in the first semester, critically reviewing my presentations and his wonderful insights on writing a paper. Aatreyi, for her efforts for SSAGA along with Prabal and Chidham. It was a wonderful experience to be a co-author in her work on choke points. Harshitha, for being my mentor in my first semester and helping me learn about the various tools that are used in the lab. I would also like to thank Asmita, for being my friend for the past eight years. I am extremely happy that she has decided to come to USU to pursue her Masters and for all the wonderful times and food we have shared in the last few months. I would also like to thank several students who have helped me one way or the other: Tarak, Manzi, Kurt, Brian, Pramesh, Andrew, Kenny, Kenyon, Trevor, Michael.

I am grateful to the ECE department and all of the staff members for making this journey enjoyable. I am thankful to Mary Lee and Tricia Brandenburg for their guidance in order to solve various administrative matters. I would also like to acknowledge the efforts of Trent Johnson and Scott Kimber for helping me out with various technical issues.

Last, but not the least, I would like to thank my family for their love and support. To Shrabanti, for being a constant source of inspiration and guidance. To Sadia, for being one of my closest friends and the numerous conversations that kept me going. And to Sritama, who always believed in me.

Shamik Saha

CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
DVFS	Dynamic Voltage and Frequency Scaling
MPSoC	Multi processor System on Chip
VF	Voltage Frequency
CAD	Computer Aided Design
SM	Streaming Multiprocessor
APP	Accelerated Parallel Processing
IPS	Instructions Per Second
PPW	Performance Per Watt
SPSA	Simultaneous Perturbation Stochastic Approximation
MILP	Mixed Integer Linear Programming
MIQP	Mixed Integer Quadratic Programming
QCQP	Quadratically Controlled Quadratic Program
SA	Simulated Annealing
ObPAS	Oblivious Workload Formation with Performance Aware Scheduling
EnPAS	Energy-efficient Workload Formation with Performance Aware Scheduling
SPAT-MT-ObPAS	Spatial Multitasking ObPAS
SPAT-MT-EnPAS	Spatial Multitasking EnPAS
SPAT-MT-DVFS-ObPAS	Spatial Multitasking with DVFS ObPAS
SPAT-MT-DVFS-EnPAS	Spatial Multitasking with DVFS EnPAS
PG	Power Gating

CHAPTER 1

Introduction

The evolution of GPUs over the last decade has reinforced general purpose computing while sustaining a steady performance growth in graphics intensive applications. However, the immense performance improvement is generally associated with a steep rise in the GPU power consumption [1]. Consequently, GPUs are already close to the abominable *power wall* [2]. Considering the massive popularity of the mobile devices running GPGPU applications [3,4], it is of utmost importance to develop energy efficient systems, which can meet the strict performance requirements. Hence, a flexible GPU design paradigm is the need of the hour.

Researchers have recently explored the power-performance benefits of co-scheduling multiple applications, sharing the GPU resources. This technique, referred to as *spatial multitasking*, vastly improves the aggregate throughput of a GPU, without compromising the performance demands of the running applications [5–7]. While technically intriguing, spatial multitasking brings in unique challenges and opportunities for circuit-architectural innovation, to further bolster the energy efficiency of the modern GPUs.

Dynamic Voltage and Frequency Scaling (DVFS) is one of such circuit level adaptations, that has played a major role in rapidly curtailing the overall chip power consumption in MPSoCs [8–11], and is also being considered as a viable option for GPUs. Governed by the run-time core utilization of the running applications, a DVFS engine adjusts the voltage and frequency (VF) levels of the on-chip cores, to save power. However, the efficacy of DVFS is gradually diminishing, as run-time adaptations of VF levels cannot modify the inherent properties (threshold voltage and gate size for example) of the transistor devices, designed at the nominal frequency. Moreover, DVFS is highly ineffective in reducing dynamic power consumption for lower technology nodes, as the cessation of Dennard scaling virtually obviates supply voltage reduction.

To tackle these challenges, we uncover a couple of intriguing circuit-architectural phenomena in this paper: (a) the emerging GPGPU applications achieve their best energy efficiencies at diverse VF domains, and (b) customizing a Streaming Multiprocessor (SM) at a lower frequency, is significantly more energy efficient, compared to employing DVFS on an SM, designed for a high frequency operation. Combining these insights with our rigorous cross-layer analysis, we propose a novel, energy efficient system level CAD approach, referred as *SSAGA—Streaming Multiprocessors Sculpted for Asymmetric GPGPU Applications*. A SSAGA style GPU comprises architecturally identical SMs, which are customized to be power-performance optimal at different VF domains. In order to exploit the energy benefit of SSAGA, we create a robust optimization framework and scheduling algorithms, that ascertain the optimal application to VF domain assignments, for a range of co-scheduled GPGPU applications. *To the best of our knowledge, our work is the first of its kind, to explore application specific customized SM design, for a spatially multitasking GPU.*

1.1 Organization

The rest of this thesis is organized as follows:

- **Motivation:** We illustrate the performance asymmetry of various emerging GPGPU applications. We demonstrate the energy efficiency and dynamic energy consumption benefit of a GPU, customized for specific VF domains, compared to a DVFS scaled GPU, designed for the nominal frequency. (Chapter 2).

- **SSAGA Design:** We provide a brief overview of our energy efficient design paradigm (Chapter 3).

- **Methodology:** We describe our planned simulation infrastructure and the tools that will be used to evaluate our proposed technique in Chapter 4.

- **Experimental Results:** We figure out the different VF levels for our design paradigm and analyze its efficacy in Chapter 5.

- **Literature Review:** We review contemporary research work to increase energy efficiency in GPUs to distinguish our approach of designing an energy efficient GPU (Chapter 6).

- **Conclusion:** We conclude in Chapter 7 by briefing about our goal.

CHAPTER 2

Motivation

In this section, we investigate the performance asymmetry among emerging GPGPU applications (Section 2.1). We carefully analyze the inefficacy of performing DVFS in a GPU (Section 2.2), and subsequently motivate SSAGA, our energy efficient GPU design paradigm (Section 2.3).

2.1 Performance Asymmetry of GPGPU Applications

A linear relation between frequency and execution time, incurs a commensurate performance degradation, when the operating frequency of a core is reduced. However, in reality, the emerging GPGPU applications exhibit a stark variety of frequency sensitivities. Figure 2.1 illustrates such variation in performance degradation, when various benchmarks from AMD's Accelerated Parallel Processing (APP) SDK suite [12], run at 525 MHz and 350 MHz. The values are measured with respect to the performance at 700 MHz. We notice that a 50% reduction in operating frequency, brings about less than 5% performance degradation in *Histogram* and *RadixSort*. On the other hand, *BitonicSort* and *Blackscholes* suffer ∼50% degradation, at the same reduced frequency. Such a dichotomy of frequency sensitivity stems from the intrinsic compute bound and memory bound nature of the GPGPU applications. Many techniques, particularly in CPU design, in the past have employed DVFS to exploit this dichotomy, which we now delve into.

2.2 DVFS: A Discreet Achilles Heel

While DVFS techniques have gained popularity in the past, a few recent works reveal the inefficacy of DVFS in delivering an optimal performance per watt, in multicore and network-on-chip domains [13,14]. Using a meticulous analysis, we demonstrate the limited potential of DVFS in improving the energy efficiency of a GPU. A DVFS engine sporadically

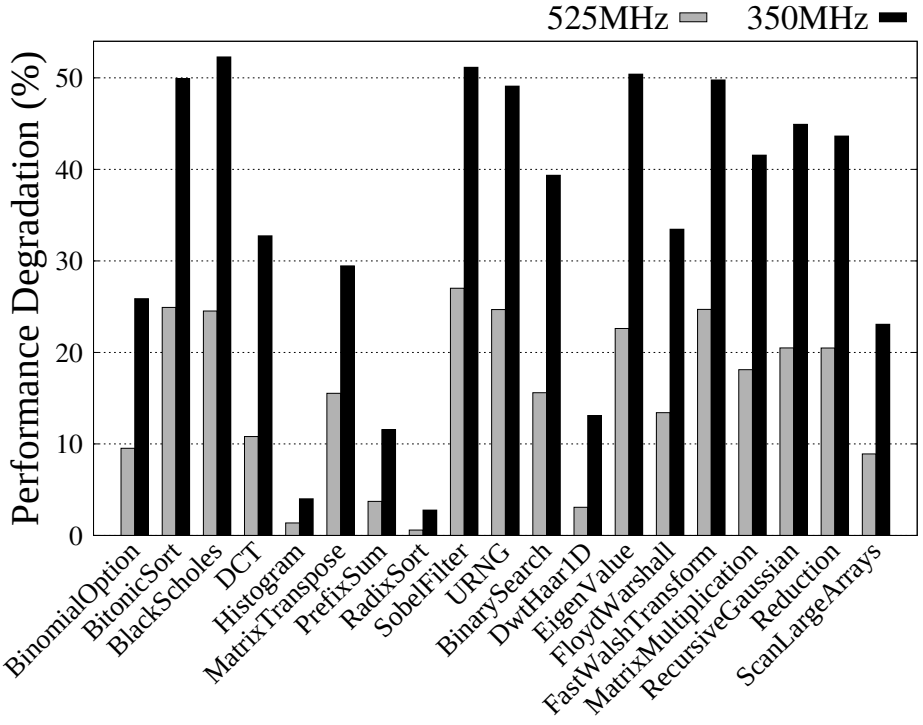


Fig. 2.1: Performance Degradation (lower is better).

runs an SM core away from its nominal frequency. Altering the frequency of operation from the nominal frequency, may negatively impact the energy efficiency of the SM, as run-time adaptations cannot alter the inherent device characteristics (threshold voltages and gate size distribution for example), which are designed for the nominal frequency. DVFS is also quite ineffective in reducing the dynamic power consumption at lower technology nodes, as the cessation of Dennard scaling virtually obviates supply voltage reduction. To empirically demonstrate the limitations of DVFS, we synthesize the GPU hardware components, detailed next.

Methodology: We use the Synopsys Design Compiler to synthesize a representative GPU RTL [15], using a 32 nm standard cell library by Synopsys, with Multi-VT optimization (i.e. using multiple threshold voltages) at the nominal frequency. We consider a customized design, and a DVFS scaled design, at various operating frequencies. To create a customized design, we synthesize the GPU RTL using Multi-VT optimization, at multiple operating voltages (1X to 0.9X of the nominal voltage), and choose the design with minimum power

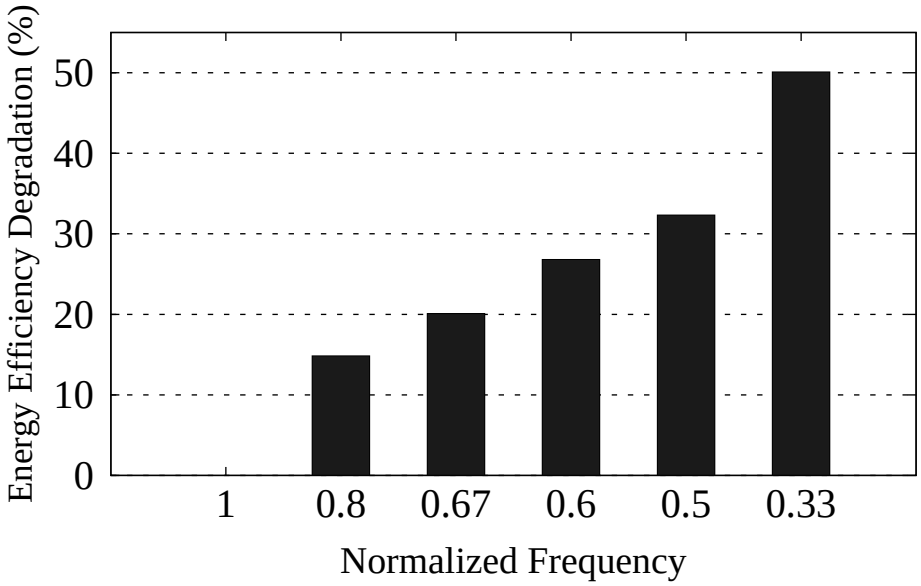


Fig. 2.2: Energy Efficiency Degradation with DVFS (lower is better).

consumption. On the other hand, we compute the downscaled power numbers at various frequencies, to emulate a DVFS on the nominal GPU.

Results: Figure 2.2 illustrates the energy benefits of the customized design paradigm, as opposed to a DVFS augmented design. Typically, a GPU is designed to be power-performance optimal at the nominal frequency (marked as 1 in the X-axis). We perform DVFS on the nominal GPU, to operate it at various lower frequencies. At each operating frequency, we report the respective degradation in energy efficiency of the DVFS scaled GPU, compared to the customized GPU, at that operating frequency. We notice a staggering ∼50% degradation in energy efficiency at 33% of the nominal frequency. A primary reason for a custom design GPU to be more energy efficient than a DVFS enabled GPU is the fact that it consumes less dynamic power. Figure 2.3 illustrates that the custom design paradigm consumes significantly less dynamic power than a DVFS augmented GPU. We notice that the custom design GPU consumes ∼32% less dynamic power than the DVFS scaled GPU at 33% of the nominal frequency. The custom design GPU is likely to consume even lesser dynamic power and be more energy efficient than a DVFS enabled GPU at lower technology nodes, due to cessation of Dennard scaling.

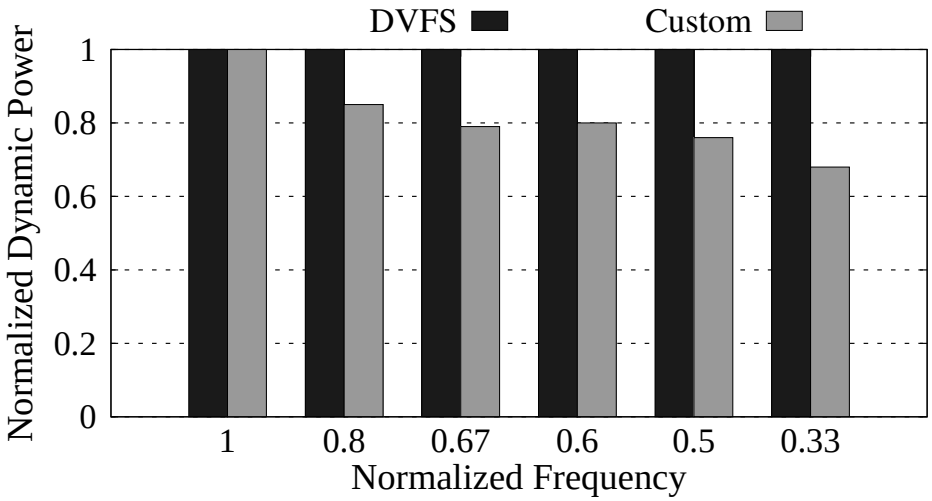


Fig. 2.3: Dynamic power consumed by custom design GPU compared to DVFS enabled GPU.

2.3 Significance

Our initial results have demonstrated the benefit of a customized circuit design, as opposed to a DVFS scaled system design (Section 2.2). Exploiting this novel technique, along with the inherent performance asymmetry of the emerging GPGPU applications (Section 2.1), we embark on exploring SSAGA, our novel GPU design paradigm.

CHAPTER 3

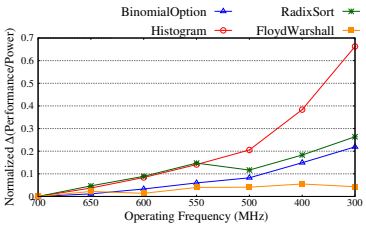
Energy Efficient GPU Design Paradigm

In this section, we propose SSAGA, our energy-efficient system level CAD paradigm. SSAGA aims to design the SMs of a GPU at specific VF levels, dictated by the performance asymmetry of the GPGPU applications. We discuss the system level overview (Section 3.1) and the need for an optimization framework (Section 3.2), before delving into the details of SSAGA (Section 3.3). We conclude with our workload creation and scheduling policies (Section 3.4).

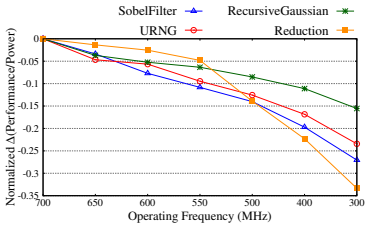
3.1 System Overview

We consider our spatially multitasking GPU to be composed of different VF domains. We select the VF domains for our GPU by subjecting our robust optimization framework (Section 3.3) to a plethora of workloads (Section 5.1). We assume that our custom designed GPU has an infinite pool of applications which can be scheduled onto specific VF domains. The infinite pool of applications helps us model a real world implementation, in which SMs can be dynamically allocated and reclaimed from completed GPGPU kernels. The infinite number of applications also rigorously exercises our design paradigm as it is subject to workloads [1] with varying degrees of memory/compute intensity. *To show the data for Section 5.3, 5.4 and 5.5, we consider a subset of 10 workloads which represent the overall variation of the entire workload space.* Our system implements memory isolation among the co-executing applications to prevent malicious applications from accessing the memory space of other applications [5]. The SMs are distributed evenly among the applications [5]. To ensure appropriate application to VF domain mapping, we have adopted various scheduling policies explained in Section 3.4. The implementation details of these policies are discussed in Section 3.5.

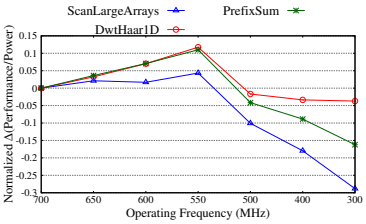
[1] We define a *workload* as the combination of applications, while an *application* is an individual GPGPU benchmark.



(a) Energy efficiency improves as frequency decreases.



(b) Energy efficiency degrades as frequency decreases.



(c) Energy efficiency changes non-monotonically as frequency decreases.

Fig. 3.1: Improvement in Energy Efficiency vs. Operating Frequency (Negative values signify degradation).

3.2 SSAGA: An Optimization Problem

3.2.1 Energy Efficiency Heterogeneity

Figure 3.1 illustrates why maximizing the energy efficiency of a spatially multitasking GPU is a combinatorial optimization problem. We characterize 17 GPGPU applications across a wide range of operating frequencies, and notice a remarkable diversity in their energy efficiency profiles. Figure 3.1 demonstrates some of the applications, representing three distinct trends. For example, Figure 3.1(a) shows that the performance per watt of *Histogram* and *BinomialOption*, monotonically increases at lower frequencies. In contrast, *SobelFilter* and *Reduction* display a steady reduction in energy efficiencies, as the frequency decreases (Figure 3.1(b)). A third category of applications (e.g. *PrefixSum*) in Figure 3.1(c), exhibit a non-monotonic progression of the efficiency, with declining operating frequency. As the applications have vastly disparate frequency domains for energy efficient executions, searching for an optimal VF domain configuration in the GPU involves a large solution space.

3.2.2 Solution Space of SSAGA

Finding the optimal application to VF domain assignment is computationally intensive. Considering a total of n applications out of which k ($k <= n$) applications can be co-scheduled on the GPU, and the number of available $\{V,F\}$ tuples to be p, the size of the solution space is ${}^{n}P_k * p^k$. As an example, 4 co-scheduled applications out of a total of 20, with 4 $\{V,F\}$ tuples, give rise to 29767680 possible solutions. Hence, finding the optimal solutions to maximize the energy efficiency is non-trivial and requires a robust optimization framework.

3.3 Optimization Framework

Pertinent to our optimization framework, we formulate an objective function (Section 3.3.1), determine the constraints (Section 3.3.2), and ascertain the bounds for the tunable variable (Section 3.3.3). We subsequently discuss the choice of our optimization algorithm (Section 3.3.4), and present *Simultaneous Perturbation Stochastic Approximation (SPSA)* based approach, to ascertain the optimal solutions (Section 3.3.5).

3.3.1 Objective Function

To optimize the energy efficiency when multiple applications are co-scheduled, we aim at maximizing the overall Performance Per Watt (PPW) of the GPU. We define performance in terms of Instructions Per Second (IPS), to make our objective function independent of the execution time of each application. To minimize the impact of the wide variation of power and performance values, across a diverse workload space, we normalize the PPW of each application, with respect to its PPW at the nominal frequency. We set an upper bound on the operating frequency range, and call it the nominal frequency (F_n). We denote the difference between the normalized PPWs at an operating frequency, and at F_n, as $\Delta(\frac{Perf}{Power})$. The set of permissible VF domains for the GPU is defined as

$$D = \{(V_1, F_1), (V_2, F_2), \ldots, (V_p, F_p)\} \tag{3.1}$$

Our objective is to maximize the total $\Delta(\frac{Perf}{Power})$, for all the applications running simultaneously at different VF domains on the GPU. In our system we implement memory isolation by segmenting memory and assigning each segment to an application. The set of VF domains selected for an application is given by

$$\nu = \{(V_i, F_i) : (V_i, F_i) \in D\}, 1 \leq i \leq k \tag{3.2}$$

Hence, the objective is to

$$Maximize \quad \sum_{i=1}^{k} \Delta(\frac{Perf}{Power})(\nu_i) \tag{3.3}$$

where k is the number of co-scheduled applications. Both Equations 3.2 and 3.3 have the same upper limit, as the number of co-scheduled applications is equal to the number of V-F domains in our work. The objective function is calculated by simulating the workload at a particular V-F domain.

3.3.2 Constraint Equation

With a view to maximizing the objective function, we schedule the applications to different VF domains, dictated by the applications' performance sensitivities. Applications running at a frequency lower than the nominal frequency F_n, are susceptible to a significant performance degradation. To limit such performance degradation, we define a constraint equation for the objective function. We restrict the average relative performance of the co-scheduled applications, to be greater than P_0. Hence, the constraint equation is

$$Y = \frac{1}{k} \sum_{i=1}^{k} (\frac{Perf(F_i)}{Perf(F_n)}) \geq P_0 \tag{3.4}$$

From Equation 3.4, we can conclude that P_0 can only be a fractional value greater than 0 and less than equal to 1.

3.3.3 Bounds Definition

For both the objective function and the constraint equation, the operating frequency is the only tunable variable. Hence, selection of the range of operating frequencies is vital for the quick convergence of the objective function. The upper bound of the frequency range is given by the nominal frequency (F_n), as this frequency gives the best performance for all the applications. The lower bound of the frequency range (F_l), is given by the frequency, at which the average performance degradation of the applications is limited by P_0. Thus, the bounds for the frequency range are denoted as

$$F_l \leq F_i \leq F_n \tag{3.5}$$

Our optimization algorithm uses this framework to select the VF domains for the applications (Section 3.3.5). Therefore, the complete optimization framework is

$$\begin{aligned} Maximize \quad & \sum_{i=1}^{k} \Delta(\frac{Perf}{Power})(\nu_i) \\ such\ that \quad & \frac{1}{k}\sum_{i=1}^{k}(\frac{Perf(F_i)}{Perf(F_n)}) \geq P_0 \\ & F_l \leq F_i \leq F_n \end{aligned}$$

3.3.4 Choice of Optimization Algorithm

A few key factors drive the judicious selection of our optimization algorithm. For example, we abandon the popular *Mixed Integer Linear Programming (MILP)* or *Mixed Integer Quadratic Programming (MIQP)*, or *Quadratically Constrained Quadratic Program (QCQP)* based optimization, as our objective function is not strictly linear or quadratic in nature (Section 3.3.1). We may approximate Equation 3.3 and formulate the objective function as MILP problem. However, with the advent of intra-SM multitasking [16, 17], and an increase in the number of SMs for the future GPUs, the number of VF domains is likely to increase. In such a scenario, the approximate objective function may predict

inaccurate V-F domains. We also cannot use the traditional Knapsack approach, as its space complexity increases exponentially with increasing accuracy. Hence, we need to use a stochastic approximation algorithm (e.g. *Simulated Annealing (SA), Simultaneous Perturbation Stochastic Approximation (SPSA)*) that generates reasonably accurate solutions with polynomial space-time complexity. Now, both SPSA and SA achieve the same level of accuracy for a given number of iterations. However, SPSA will converge to the optimal solution within a given level of accuracy with fewer measurements of the objective function than SA [18]. Hence, considering our large set of GPGPU applications, we select SPSA over SA, due to SPSA's quick convergence time.

3.3.5 Simultaneous Perturbation Stochastic Approximation (SPSA) Framework

Algorithm 1 shows how SPSA can be used to identify the VF domains. The algorithm follows an iterative approach in order to maximize the objective function.

Algorithm 1 SPSA based VF domain selection

1: Initialize: $V \leftarrow \nu_{in}$
2: a, c, A, α, γ ▷ non-negative coefficients
3: Calculate $O \leftarrow \sum_{i=1}^{k} \Delta PPW(V)$
4: **for** $j = 1 \rightarrow N$ **do**
5: Calculate a_j, c_j ▷ gain sequences
6: Generate Δ_j ▷ perturbation vector
7: Evaluate Loss function, $L(V) = \sum_{i=1}^{k} \Delta PPW(V \pm \Delta_j)$
8: Generate gradient approximation,
$g_j(\hat{V}) = (L(V))(\Delta_j)$
9: $V_{new} = V - a_j.g_j(\hat{V})$
10: $O_{new} = \sum_{i=1}^{k} \Delta PPW(V_{new})$
11: Calculate $Y(V_{new})$
12: **if** $Y(V_{new})$ is satisfied **then**
13: **if** $(O_{new} > O)$ **then**
$O \leftarrow O_{new}$ $V \leftarrow V_{new}$
14: **end if**
15: **end if**
16: **end for**

Initialization: We begin the algorithm by creating our initial solution, $\nu_{in} = (V_i, F_i)$, where $(1 \leq i \leq k)$. This solution satisfies our constraint equation by assigning the highest

values to all the VF domains. The selection of the gain sequences is a prominent part of SPSA, as it determines the performance of the algorithm. The coefficients α and γ, decide the step-size of each iteration [18].We select α and γ to be 0.6 and 0.1, respectively. The coefficient c is selected as a small positive number to measure more accurate loss function. Similarly, the coefficients a and A, are selected to eliminate the instabilities during the earlier iterations, and improve the performance in later iterations [18].

Gradient Approximation: We perturb the current solution V, by generating the gradient approximation function $g_j(\hat{V})$. In order to determine $g_j(\hat{V})$, we generate a simultaneous perturbation vector (Δ_j), and a loss function $(L(V))$ [18]. The loss function is same as Equation 3.3 in our case. The perturbation vector, Δ_j, is created in such a way, that each of its components is independently generated using the zero mean probability distribution satisfying the conditions in [19]. A simple choice to generate each component of Δ_j is to use a Bernoulli ± 1 distribution with a probability of 0.5 for each outcome. Uniform and normal random variables are prohibited by the regularity conditions of SPSA to generate the elements of Δ_j (as they have infinite inverse moments) [18]. On the other hand, the loss function, $L(V)$, is evaluated using the perturbation vector, and the current solution, V. We update the set of $(V_i,\ F_i)$ tuples, by subtracting the scaled gradient approximation, from the previous solution. We then calculate the new objective function, O_{new} based on the updated current solution. The algorithm terminates if it reaches the maximum achievable value, or the maximum number of iterations. For this work, we set the maximum number of iterations to 100.

3.4 Workloads and Scheduling Policies

Our SPSA framework ascertains the optimal VF levels for the GPU. However, the optimal VF levels alone cannot guarantee an energy efficient operation. The policies adopted for workload creation and scheduling, play a critical role in determining the energy efficiency of the GPU. For example, agnostically scheduling a compute intensive application to a lower VF domain, may negatively impact the energy-efficiency of the GPU. Hence, we explore a couple of smart workload formation and scheduling algorithms next.

3.4.1 Oblivious Workload Formation with Performance Aware Scheduling (ObPAS)

We use this scheme to create random workloads, agnostic of the benchmark characteristics.

Workload Formation: This scheme randomly selects 4 applications from a set of 17 applications, to create a workload. Table 4.2 lists the 10 workloads we consider in this scheme. [2]

Application Scheduling: Each application within a single workload is scheduled to the appropriate VF domain. We assign a highly compute intensive application to a higher VF level, as its performance suffers significant degradation while operating at a lower VF level. On the other hand, we assign a highly memory intensive application to a lower VF level, as it is fairly insensitive to frequency variation.

3.4.2 Energy-Efficient Workload Formation with Performance Aware Scheduling (EnPAS)

An oblivious creation of the workloads may result in a sub-optimal energy efficiency of the GPU. Motivated by Figure 3.1, that depicts the disparate frequency domains of the applications for maximum energy efficiencies, we attempt to create new workloads with an ideal mix of compute and memory intensive applications. We propose Algorithm 2, to create and schedule the workloads in this scheme.

Application Characterization: In order to segregate compute and memory intensive applications, we run the GPU kernel of each application for one iteration, and determine the number of ALU instructions (AI), as well as, the number of memory accesses (MA). The ratio MA/AI, normalized to the total number of instructions, determines the compute/memory intensity of an application. For example, a higher value of the ratio signifies a memory intensive application, and vice-versa. Figure 3.2 depicts the compute/memory intensities of all the applications.

[2] We consider 10 representative workloads which exhibit all the characteristics of the entire workload space.

Line 2 of Algorithm 2 initializes a threshold value (I_th) of the ratio MA/AI (normalized). The applications whose MA/AI is less than I_th, are compute intensive, whereas the rest of the applications are memory intensive (Lines 3 and 4).

Workload Formation: With a view to improving the GPU energy efficiency, a highly compute intensive component (i.e. an application) of a workload is balanced by a highly memory intensive one, detailed next.

Application Scheduling: We initially try to assign the compute intensive applications to the highest VF slots of the workloads. Our algorithm assigns a more compute intensive application to a higher VF slot (Lines 10 to 20). Next, we assign the memory intensive applications to the vacant VF slots of the workloads. We try to assign a more memory intensive application to a lower VF slot (Lines 21 to 32). The new workloads (using Algorithm 2) are shown in Table 4.3.

3.5 Implementation

We use the system firmware to implement ObPAS and EnPAS (Section 3.4). The degree of memory/compute intensity of each application is stored in the form of LUTs. The algorithms for implementing ObPAS and EnPAS, have polynomial time complexity.

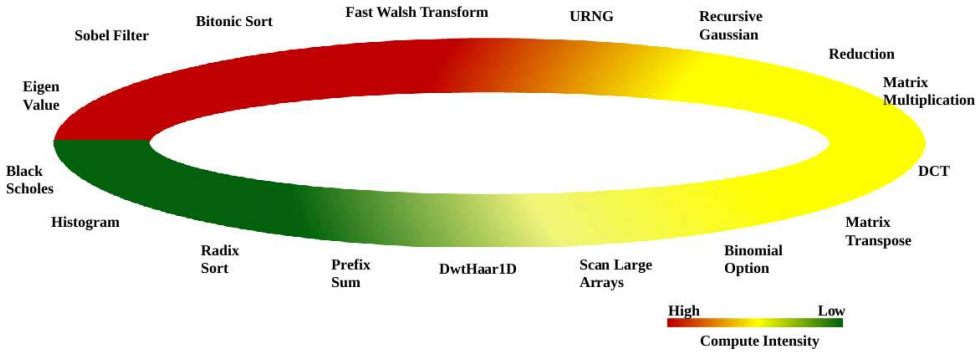


Fig. 3.2: The applications along the red and green arcs are highly compute and highly memory intensive, respectively. The ones along the yellow arc are moderately compute/memory intensive.

Algorithm 2 Algorithm to create and schedule workloads

STEP 1: To determine whether an application is compute or memory intensive.
Initialize: I_{th} ▷ threshold value of (memory-accesses)/(ALU-accesses).
array *compute* ▷ $setofapplicationswith(memory-accesses)/(ALU-accesses) < I\_th$ (ascending order).
array *memory* ▷ $setofapplicationswith(memory-accesses)/(ALU-accesses) > I\_th$ (ascending order).
$Workloads[id]$ ▷ user-defined datatype for workloads.
$voltage[level]$ ▷ Voltage levels for a particular workload.
boolean $occupied[level]$ ▷ Tracks if a voltage level is occupied.
STEP2: Rearrange Workloads.
Initialize: $id = level = 0$; ▷ Assume as *level* increases, voltage decreases
for $k = 0 \rightarrow (sizeof)arraycompute/(sizeof)compute[0]$ **do**
    if $Workloads[id].occupied[level]$ == false **then**
        Assign task to $Workloads[id].voltage[level]$
        $Workloads[id].occupied[level]$ = true
    end if
    Increment id to point to next workload.
    if id == Maximum number of workloads **then**
        Make id point to first workload.
        Make level point to next lower voltage.
    end if
end for
Initialize: $id = 0$, $level = 4$; ▷ Assume 4 VF levels.
for $k = (sizeof)arraymemory/(sizeof)memory \rightarrow 0$ **do**
    if $Workloads[id].occupied[level]$ == false **then**
        Assign task to $Workloads[id].voltage[level]$
        $Workloads[id].occupied[level]$ = true
    end if
    Increment id to point to next workload.
    if id == Maximum number of workloads **then**
        Make id point to first workload.
        Make level point to next higher voltage.
    end if
end for

CHAPTER 4

Methodology

Our cross-layer methodology amalgamates an architectural simulation framework with circuit level power and thermal estimation. Details of each component of the methodology are discussed next.

4.1 Architectural Simulation

We use Multi2Sim [20] to perform architectural simulation of AMD's Evergreen architecture GPU – Radeon R9. Table 4.1 lists the architectural parameters for our GPU. Figure 4.1 shows the VF domains, we consider for this work. We create 4 VF domains (also referred as SM clusters), each comprising 20 SMs. The domain configuration for SSAGA, is specified in Section 5.1. We consider a GPU with a reasonably large number of SMs as the GPUs from Nvidia and AMD are showing a steady increase in the number of SMs. We do not scale the voltage and frequency of individual SMs as it will have the following drawbacks: (a) modest area and power overheads due to voltage regulators and (b) challenges in physical design of the GPU. For each benchmark, we run the GPU kernel for 1 iteration, once for each of the multiple operating frequencies. The output statistics from Multi2Sim are used as inputs to our SPSA based optimization framework.

Table 4.1: AMD's Radeon R9 GPU configuration.

Parameters	**Values**
Frequency	700 MHz
Number of SMs	80
Thread Group Size	64
Local memory	32Kb, latency: 2-cycles
L2 cache	8x256Kb, latency: 20 ns
Device Memory	B/W: 264 GB/s, latency: 100 ns

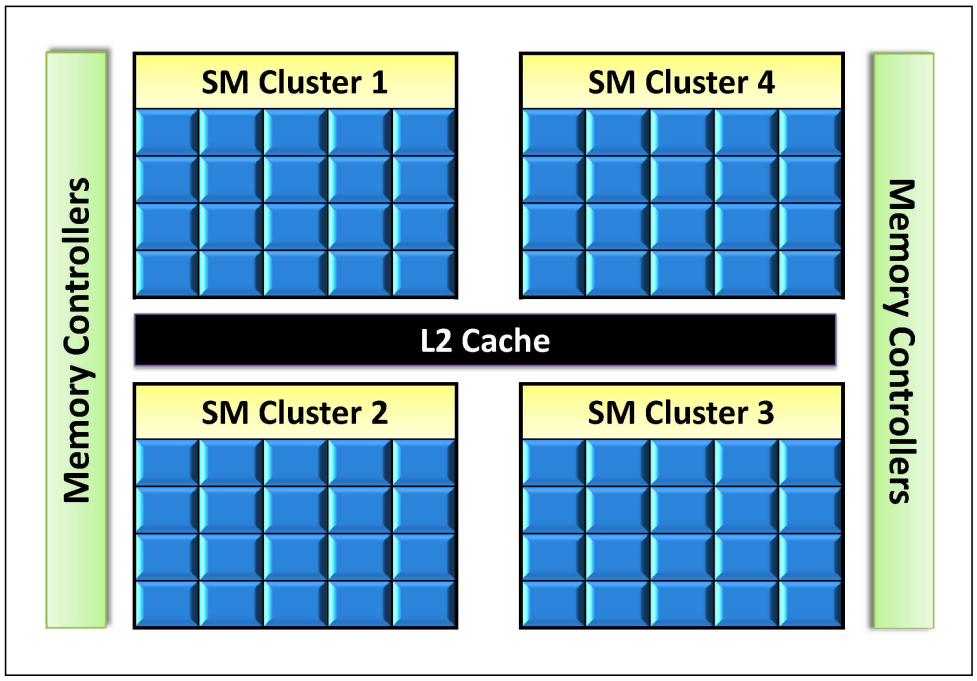


Fig. 4.1: SM clusters in GPU.

4.2 Workloads

We use several emerging GPGPU applications from AMD's Accelerated Parallel Processing (APP) SDK suite [12]. We create 10 workloads for our analysis, where each workload comprises 4 applications. We initialize I_th (in Algorithm 2) with an appropriate value and determine our set of compute and memory intensive benchmarks. Table 4.2 and 4.3 list the workloads based on ObPAS and EnPAS, respectively.

4.3 Power Estimation

To evaluate the dynamic and leakage power consumptions, we synthesize a representative GPU RTL [15] using Synopsys Design Compiler and a 32 nm standard cell library, at various target frequencies. We use multiple threshold voltages (e.g. regular V_t(RVT), low V_t(LVT) and high V_t(HVT)), and enable multi-VT optimization in synthesis. Although we have used an open source GPU RTL to evaluate the power consumption of the simulated AMD GPU, the trends exhibited in our results will be maintained for an AMD GPU as well. This happens because, irrespective of the power model we consider, an SM customized

Table 4.2: ObPAS based workloads. C and M in the parenthesis denote compute intensive and memory intensive application respectively.

Workloads	Constituents
$W1_O$	Binomial Option (M), DCT (M), Histogram (M), Matrix Transpose (M)
$W2_O$	Bitonic Sort (C), BlackScholes (C), EigenValue (C), Recursive Gaussian (C)
$W3_O$	URNG (C), SobelFilter (C), Radix Sort (M), Scan Large Arrays (M)
$W4_O$	FastWalshTransform(C), DwtHaar1D (M), Reduction(M), MatrixMultiplication (M)
$W5_O$	DwtHaar1D (M), PrefixSum (M), Histogram (M), Bitonic Sort (C)
$W6_O$	DwtHaar1D (M), Reduction (M), ScanLargeArrays (M), DCT (M)
$W7_O$	Radix Sort (M), BlackScholes (C), URNG (C), Binomial Option (M)
$W8_O$	SobelFilter (C), DCT (M), Reduction (M), MatrixMultiplication (M)
$W9_O$	Reduction (M), SobelFilter (C), PrefixSum (M), DwtHaar1D (M)
$W10_O$	Histogram (M), URNG (C), FastWalshTransform (C), Scan Large Arrays (M)

Table 4.3: EnPAS based workloads. C and M in the parenthesis denote compute intensive and memory intensive application respectively.

Workloads	Constituents
$W1_E$	BlackScholes (C), URNG (C), DwtHaar1D (M), RadixSort (M)
$W2_E$	BlackScholes (C), URNG (C), ScanLargeArrays (M), RadixSort (M)
$W3_E$	URNG (C), EigenValue (C), ScanLargeArrays (M), Histogram (M)
$W4_E$	SobelFilter (C), RecursiveGaussian (C), ScanLargeArrays (M), Histogram (M)
$W5_E$	SobelFilter (C), MatrixMultiplication (M), BinomialOption (M), Histogram (M)
$W6_E$	SobelFilter (C), MatrixMultiplication (M), BinomialOption (M), PrefixSum (M)
$W7_E$	BitonicSort (C), Reduction (M), Matrix Transpose (M), Prefix Sum (M)
$W8_E$	BitonicSort (C), Reduction (M), DCT (M), DwtHaar1D (M)
$W9_E$	FastWalshTransform (C), Reduction (M), DCT (M), DwtHaar1D (M)
$W10_E$	FastWalshTransform (C), Reduction (M), DCT (M), DwtHaar1D (M)

at a given frequency, will be more energy efficient than its DVFS scaled counterpart.

4.4 Thermal Estimation

We estimate the workload aware runtime power consumption, by combining the static and dynamic power of the synthesized hardware, with the runtime SM utilization data from Multi2Sim. We then feed the calculated power dissipation data to HotSpot 6.0, to generate the thermal characteristics [21]. We model the steady state temperature characteristics of the underlying GPU architecture, and conduct HotSpot simulations at the grid level, to generate an accurate thermal characteristics for the GPU. We use the grid model provided by HotSpot 6.0, as it is more accurate than the block model provided by the same. The grid

model divides the silicon die and the different components into regular grid cells. The grid model achieves more accuracy by modeling the lateral heat transfer in more detail than the block model [22].

CHAPTER 5

Experimental Results

In this section, we figure out the different voltage frequency levels for our SSAGA design paradigm (Section 5.1), analyze the efficacy of SSAGA, by presenting the power-performance benefits of various comparative schemes (Section 5.2). We discuss the power consumptions, energy efficiency and thermal benefit of the schemes in Section 5.3, 5.4, and 4.4, respectively.

It is important to mention that in all our experiments, we have considered the time and energy overhead produced by our schemes. Similarly, we have also taken into consideration the energy overhead due to the memories. This overhead has been calculated on the basis of the values given in [23] and [24].

5.1 V-F Domains

In order to determine the optimum voltage-frequency levels for SSAGA, we generate around 2000 unique workloads which have varying degrees of compute/memory intensity and subject it to our SPSA framework (Section 3.3.5). The workloads were created with the aim of covering the entire compute/memory intensity spectrum. The domain configuration for SSAGA according to our optimization framework, is specified in Table 5.1. Clusters 2,3 and 4 have the same voltage as many in order to ensure that the circuit meets timing.

Table 5.1: SSAGA Solution for VF Domain Configurations.

SM Cluster	**VF Configuration**
SM Cluster 1	1.05V, 700 MHz
SM Cluster 2	0.96V, 600 MHz
SM Cluster 3	0.96V, 500 MHz
SM Cluster 4	0.96V, 400 MHz

5.2 Comparative Schemes

- **Spatial Multitasking ObPAS (SPAT-MT-ObPAS)**: This scheme co-schedules multiple applications on the GPU, with an *even* distribution of the SMs [5]. All the co-scheduled applications run at the nominal frequency (i.e. 700 MHz) of the GPU. The workloads are created and scheduled using ObPAS policy.

- **Spatial Multitasking EnPAS (SPAT-MT-EnPAS)**: This scheme is similar to SPAT-MT-ObPAS except that the workloads are created and scheduled using the EnPAS policy.

- **Spatial Multitasking with DVFS ObPAS (SPAT-MT-DVFS-ObPAS)**: This scheme loosely models DVFS technique in [25], for a spatially multitasking GPU. For our work, we have considered two different core voltages: 0.96V and 1.05V. For each core voltage we change the frequency of the core from 400MHz to 700 MHz. We perform experiments to determine the ideal voltage frequency tuple of an application running on the GPU. For each application, we fix the voltage of the core and vary the frequency. The voltage frequency pair of an application is determined by its performance per watt at that voltage and frequency. We observe that memory intensive applications have a higher performance per watt at lower frequencies, whereas compute intensive applications have a higher performance per watt at higher frequencies. We also observe that a lower core voltage is significantly effective in saving energy for all applications. Again, we use the ObPAS policy to create and schedule the workloads. Each application of the workload runs at its optimal voltage and frequency pair as determined by our experiments.

- **Spatial Multitasking with DVFS EnPAS (SPAT-MT-DVFS-EnPAS)**: This scheme is similar to SPAT-MT-DVFS-ObPAS except that the workloads are created and scheduled using the EnPAS policy.

- **SSAGA-ObPAS**: This is our proposed scheme implemented with a SSAGA style GPU with optimized VF domains, and ObPAS scheduling.

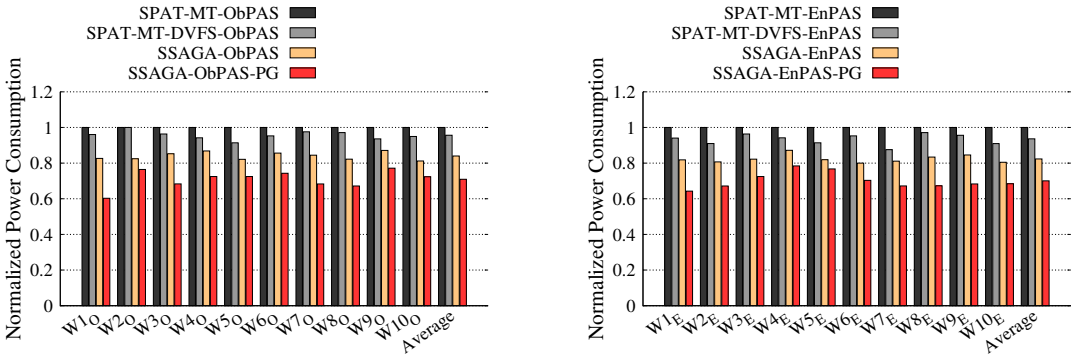
- **SSAGA-ObPAS-PG**: This scheme combines SSAGA-ObPAS with the core power gating technique, proposed in [2]. Based on the SM sensitivity, each application is squeezed to a lesser number of SMs, to reduce the idle time in each SM. The idle SMs are power gated. To study the sensitivity of an application to the number of SMs, for each application we vary the SMs from 1 to 20. We assume that an application of a workload will not run on more than 20 SMs, as in our work, each VF island is assigned 20 SMs. The performance of the application at the different number of SMs is noted. It is seen that certain applications such as *BinomialOption, BitonicSort Histogram, MatrixTranspose* show less than 0.7% degradation in performance when the number of SMs are reduced from 20 to 15. For our work, we continue to decrease the SMs for an application as long as the performance degradation is negligible and power gate the rest of the SMs. In our work, a non-power gated GPU uses all of its 80 SMs. However, for SSAGA-ObPAS-PG the number of active SMs depends on the SM sensitivity of the applications in the workload. We account for the wake-up latency overhead associated with power gating, in our architectural simulation [26].

- **SSAGA-EnPAS**: This is also our proposed scheme that combines a SSAGA style GPU with EnPAS scheduling.

- **SSAGA-EnPAS-PG**: This scheme is similar to SSAGA-ObPAS-PG except that the workloads are created and scheduled using the EnPAS policy.

5.3 Power Consumption

Figure 5.1 depicts the power consumptions of the comparative schemes when subjected to ObPAS and EnPAS policies. Figure 5.1(a) and Figure 5.1(b) are normalized to the baseline Spatial Multitasking ObPAS GPU (SPAT-MT-ObPAS) and Spatial Multitasking EnPAS GPU (SPAT-MT-EnPAS), respectively. SPAT-MT-DVFS-ObPAS and SPAT-MT-DVFS-EnPAS show marginal improvement when compared to their baseline. However, SSAGA-ObPAS, SSAGA-ObPAS-PG, SSAGA-EnPAS and SSAGA-EnPAS-PG significantly reduce the power consumption for all workloads, primarily due to the energy



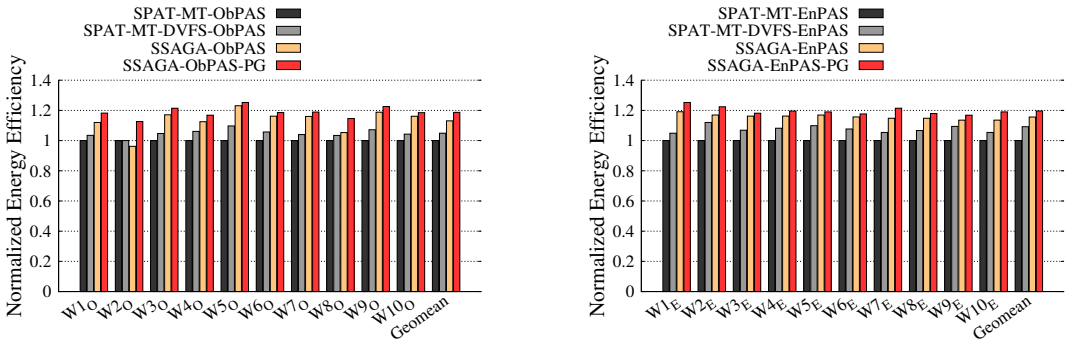
(a) Normalized Power Consumption (ObPAS policy). (b) Normalized Power Consumption (EnPAS policy).

Fig. 5.1: Normalized Power Consumption (lower is better).

benefit of the customized SM design, at lower frequencies. In general, a compute intensive task consumes more power than a memory intensive task, while operating at the same VF level. In Figure 5.1(a), SSAGA-ObPAS shows maximum energy saving among non-PG schemes for $W1_O$ and $W5_O$ as they are extremely memory intensive. On an average, SSAGA-ObPAS reduces the power consumption by 16% compared to its baseline (SPAT-MT-ObPAS), whereas SSAGA-EnPAS reduces the power consumption by 16.5% compared to its baseline (SPAT-MT-EnPAS). SSAGA-ObPAS-PG and SSAGA-EnPAS-PG further reduces power consumption by power gating the idle cores, hence consuming significantly less leakage power. We notice 29% and 30% average power savings with SSAGA-ObPAS-PG and SSAGA-EnPAS-PG, respectively, across all workloads compared to their respective baselines. From our experiments, we also see that SPAT-MT-ObPAS, SPAT-MT-DVFS-ObPAS, SSAGA-ObPAS and SSAGA-ObPAS-PG consume almost the same amount of power as their EnPAS counterparts (the differences are less than 1.1% for all schemes).

5.4 Energy Efficiency Results

Figure 5.2 shows the energy efficiencies of the comparative schemes, when subjected to ObPAS and EnPAS policies. Figure 5.2(a) and Figure 5.2(b) are normalized to Spatial Multitasking ObPAS GPU (SPAT-MT-ObPAS) and the Spatial Multitasking EnPAS GPU (SPAT-MT-EnPAS), respectively. We measure the energy efficiency in terms of IPS/Watt. SSAGA-ObPAS, SSAGA-ObPAS-PG, SSAGA-EnPAS and SSAGA-EnPAS-PG deliver a

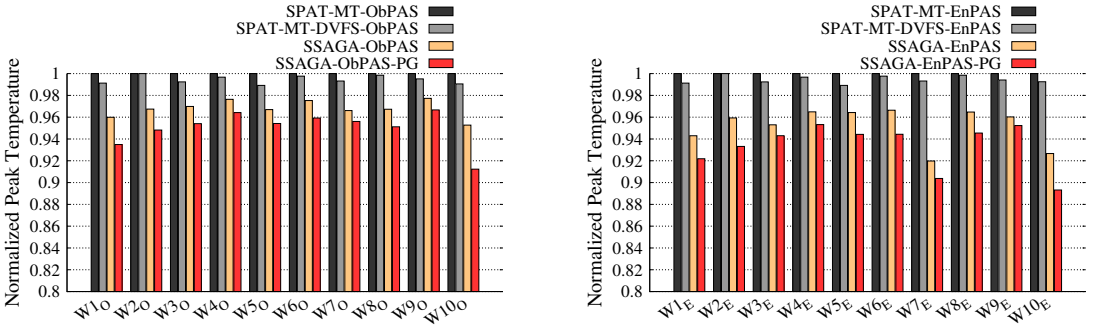


(a) Normalized Energy Efficiency (ObPAS policy). (b) Normalized Energy Efficiency (EnPAS policy).

Fig. 5.2: Normalized Energy Efficiency (higher is better).

better performance per watt, compared to other schemes. Such an outcome stems from (a) optimal application to VF domain assignments, and (b) reduced energy consumption of the customized SMs, at the lower operating frequencies. However, for workload $W2_O$, SSAGA-ObPAS is slightly less energy efficient than SPAT-MT-ObPAS and SPAT-MT-DVFS-ObPAS. This is because, $W2_O$ comprises four compute intensive applications. As a result, even a modest saving in energy consumption is unable to recoup the severe performance loss at the lower frequencies, for SSAGA-ObPAS. SSAGA-EnPAS performs uniformly better than the EnPAS counterparts of other schemes, as the workloads are judiciously created by balancing both compute intensive and memory intensive applications. On an average, SSAGA-ObPAS and SSAGA-EnPAS improve the energy efficiencies by 13% and 15.62% over their baselines, SPAT-MT-ObPAS and SPAT-MT-EnPAS, respectively. We also observe that the EnPAS counterparts of the various schemes are more energy efficient than their ObPAS counterparts. [1] SSAGA-ObPAS-PG and SSAGA-EnPAS-PG further improve the energy efficiency by squeezing the threads to a smaller number of SMs, and power gating the idle cores, at the cost of an acceptable performance degradation. Overall, the average energy efficiencies of SSAGA-ObPAS-PG and SSAGA-EnPAS-PG are 18% and 20% higher than their respective baselines.

[1] However, the energy efficiency of SPAT-MT-EnPAS is only 0.5% more than SPAT-MT-ObPAS.



(a) Normalized Peak Temperature (ObPAS policy). (b) Normalized Peak Temperature (EnPAS policy).

Fig. 5.3: Normalized Peak Temperature (lower is better).

5.5 Thermal Improvements

A reduction in the power consumption, translates to an improvement in the thermal profile of the chip. We present the thermal benefits of the comparative schemes in Figure 5.3, in terms of the peak die temperature of the GPU. The variation in the peak temperature, across all the workloads, resembles the power saving trend in Figure 5.1. SSAGA-ObPAS, SSAGA-ObPAS-PG, SSAGA-EnPAS and SSAGA-EnPAS-PG show improvements in the peak temperature from 2.3-4.7%, 3.3-8.7%, 3.6-8.1% and 4.7-10.7%, respectively.

CHAPTER 6

Related Work

Before GPUs became an integral part of general purpose computing several works explored the advantages of heterogeneous, single-ISA, multicore processors. These *asymmetric* single-ISA (ASISA) have the potential to outperform homogeneous processors in terms of performance/power or both. Shelepov et al. propose that in order to fully tap the potential of ASISA processors the OS needs to be heterogeneity aware, so that it can match jobs to appropriate cores. They propose HASS, which does the matching using per thread architectural signatures [27]. Li et al. have considered heterogeneous cores along with non-uniform ISAs in their work. They propose several algorithms which have been implemented in an actual OS [28]. Kumar et al. discuss the rise in processor power consumption and the potential of ASISA to reduce the processor power consumption [29]. Kim et al. uses ARM's big.LITTLE architecture and modifies the Linux kernel to make it aware of the processor utilization [30].

Several existing works also aim at improving the energy efficiency of the GPUs. These works can be broadly classified into two categories: (a) the efficacy of DVFS in modern GPUs, and (b) employing multi-tasking to increase resource utilization in GPUs. In the first category, Ge et al. have shown that DVFS, when employed in GPUs exhibit greater savings in terms of power and energy compared to CPUs [31]. Mei et al. propose to save energy by operating the GPU cores at their optimal frequency, and scaling down the supply voltage [25]. However, the efficacy of scaling core and memory frequencies depend on the application characteristics. Jiao et al. have shown that the performance and power consumption of GPU kernels are largely determined by two factors: (a) the issue rate of the instructions; and (b) the ratio of global memory transactions to computation instructions [32]. Nugteren et al. propose to dynamically change the *compute - memory* ratio by scaling the voltage and frequency of memory for compute intensive workloads,

and processing elements, for memory intensive workloads [33]. Komoda et al. address the problem of frequency scaling in heterogeneous CPU-GPU systems, by developing empirical models of the system to guide DVFS and task mapping to avoid load imbalance and power violations [34]. You et al. investigate the correlation between frames per second (FPS) and GPU utilization, to design a Quality of Service aware DVFS algorithm for embedded GPUs [35]. Wang et al. propose a dynamic frequency scaling technique for GPU platforms based on a feedback controller and improve throughput [36]. Price et al. have studied the effects of temperature, supply voltage and frequency on GPU performance. They have shown that lowering the GPU supply voltage and increasing the clock frequency while maintaining a low die temperature can increase the energy efficiency of a NVIDIA K20 GPU [37].

In the second category, Adriens et al. have shown that *spatial multitasking* offers a better speedup compared to *cooperative multitasking* [5]. Aguilera et al. allocate resources to the workload (composed of several applications), depending on the workload characteristics and the operating frequency of the SMs [7]. Reservation of cores [38, 39], hardware leaky-bucket based thread-block interleaving [40], and resource sharing between applications [6], are some of the techniques that are able to reduce power consumption and improve performance in GPUs. Jiao et al. have proposed a mix of concurrent execution and DVFS to achieve appreciable performance improvement [41]. *However, to the best of our knowledge, ours is the first work, that proposes a novel system level CAD approach to design the GPU SMs, where individual SM is customized to operate at different voltage-frequency levels.*

Recently, Wang et al. and Xu et al. have demonstrated that intra SM multi-tasking offer better performance and throughput, over a spatially multitasking GPU ([16] and [17]). However, the authors did not confirm, if their proposed techniques are more energy efficient than spatial multitasking, especially in the presence of multiple VF domains, like SSAGA. Depending on the type of usage, an end user might demand a guaranteed throughput (a mobile phone game, for example), or an energy efficient execution (GPU based data center, for example) of the GPUs. Our proposed methodology is a suitable candidate for the second

type of usage. However, SSAGA design style can be augmented with intra-SM multitasking. Our proposed ObPAS and EnPAS schedulers need to be modified (with negligible or no additional overhead), to assign multiple applications on to a single VF domain. The applications assigned to a given VF domain are homogeneous (i.e. either compute intensive or memory intensive) in nature. Such applications can preemptively execute in the assigned VF domain, while conforming to the regulations of intra-SM multitasking.

CHAPTER 7

Conclusion

In this paper, we propose *SSAGA–Streaming Multiprocessors Sculpted for Asymmetric GPGPU Applications*. Utilizing the performance heterogeneity of the GPGPU applications, and low energy benefit of customized circuit design, SSAGA creates architecturally identical SM cores, customized for different voltage-frequency domains. Our CAD based cross-layer methodology indicates an average of 20% improvement in energy efficiency, over a spatially multitasking GPU, across a range of GPGPU applications.

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] J.Lee, V.Sathisha, M.Schulte, K.Compton, and N.S.Kim, "Improving throughput of power-constrained gpus using dynamic voltage/frequency and core scaling," Oct. 2011.

[3] M. Shebanow, "An evolution of mobile graphics," *Keynote talk at High Performance Graphics*, 2013.

[4] T. Akenine-Moller and J. Strom, "Graphics processing units for handhelds," 2008, pp. 779–789.

[5] J. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte, "The case for GPGPU spatial multitasking," 2012, pp. 79–90.

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

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

[8] M. Basoglu, M. Orshansky, and M. Erez, "Nbti-aware dvfs: A new approach to saving energy and increasing processor lifetime," aug. 2010, pp. 253 –258.

[9] J. Lee and N. S. Kim, "Optimizing throughput of power and thermal-constrained multicore processors using dvfs and per-core power-gating," 2009, pp. 47–50.

[10] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," 2008, pp. 123–134.

[11] S. Garg, D. Marculescu, R. Marculescu, and Ü. Y. Ogras, "Technology-driven limits on dvfs controllability of multiple voltage-frequency island designs: a system-level perspective," 2009, pp. 818–821.

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

[13] K. Chakraborty and S. Roy, "Topologically homogeneous power-performance heterogeneous multicore systems," Mar. 2011, pp. 1–6.

[14] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran, "darknoc: Designing energy-efficient network-on-chip with multi-vt cells for dark silicon," 2014, pp. 161:1–161:6.

[15] *Theia GPU*, 2009, http://opencores.org/project,theia_gpu.

[16] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous multikernel gpu: Multi-tasking throughput processors via fine-grained sharing," 2016, pp. 358–369.

[17] Q. Xu, H. Jeon, K. Kim, W. Roo, and M. Annavaram, "Warped-slicer: Efficient intra-sm slicing through dynamic resource partitioning for gpu multiprogramming," 2016, pp. 1–13.

[18] J. C. Spall, "An overview of the simultaneous perturbation method for efficient optimization," *Johns Hopkins apl technical digest*, vol. 19, no. 4, pp. 482–492, 1998.

[19] J. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation." IEEE, 1992, pp. 332–341.

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

[21] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy, "Compact thermal modeling for temperature-aware design," 2004, pp. 878–883.

[22] W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, and K. Skadron, "An improved block-based thermal model in hotspot 4.0 with granularity considerations," *University of Virginia*, 2007.

[23] S. Hsu, A. Alvandpour, S. Mathew, S.-L. Lu, R. Krishnamurthy, and S. Borkar, "A 4.5-ghz 130-nm 32-kb l0 cache with a leakage-tolerant self reverse-bias bitline scheme," in *IEEE JOURNAL OF SOLID-STATE CIRCUITS*, 2003, pp. 755–761.

[24] A. Macii, E. Macii, and M. Poncino, "Improving the efficiency of memory partitioning by address clustering," in *Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 18–23.

[25] X. Mei, L. S. Yung, K. Zhao, and X. Chu, "A measurement study of GPU DVFS on energy conservation," in *HotPower*, 2013, pp. 10:1–10:5.

[26] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng, "Power gating strategies on gpus," in *ACM Transactions on Architecture and Code Optimization*, 2011, pp. 13:1–13:25.

[27] D. Shelepov, J. C. S. Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "Hass: A scheduler for heterogeneous multicore systems," 2009, pp. 66–75.

[28] T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, and S. Hahn, "Operating system support for overlapping-isa heterogeneous multi-core architectures," 2010, pp. 9–14.

[29] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," 2003, pp. 81–92.

[30] M. Kim, K. Kim, J. Geraci, and S. Hong, "Utilization-aware load balancing for the energy efficient operation of the big.little processor," 2014, pp. 223:1–223:4.

[31] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong, "Effects of dynamic voltage and frequency scaling on a K20 GPU," 2013, pp. 826–833.

[32] Y. Jiao, H. Lin, P. Balaji, and W. Feng, "Power and performance characterization of computational kernels on the GPU," 2010, pp. 221–228.

[33] C. Nugteren, G.-J. van den Braak, and H. Corporaal, "Roofline-aware dvfs for gpus," 2014, pp. 8–10.

[34] T. Komoda, S. Hayashi, T. Nakada, and S. Miwa, "Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping," 2013, pp. 349–356.

[35] D. You and K.-S. Chung, "Quality of service-aware dynamic voltage and frequency scaling for embedded gpus," 2014, pp. 66–69.

[36] Y. Wang and N. Ranganathan, "A feedback, runtime technique for scaling the frequency in gpu architectures," 2014, pp. 430–435.

[37] D.C.Price, M. Clark, B. Barsdell, R.Babich, and L. Greenhill, "Optimizing performance-per-watt on gpus in high performance computing," 2015, pp. 1–9.

[38] S. Kato, K. Lakshmanan, R. R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," 2011, pp. 17–30.

[39] W. Joo and D. Shin, "Resource-constrained spatial multi-tasking for embedded gpu," 2014, pp. 339–340.

[40] Y. Liang, H. P. Huynh, K. Rupnow, R. S. M. Goh, and D. Chen, "Efficient GPU spatial-temporal multitasking," vol. 26, no. 3, pp. 748–760, 2015.

[41] Q. Jiao, M. Lu, H. P. Huynh, and T. Mitra, "Improving GPGPU energy-efficiency through concurrent kernel execution and DVFS," 2015, pp. 1–11.