SSAGA: SMs Synthesized for Asymmetric GPGPU Applications

SHAMIK SAHA, PRABAL BASU, CHIDHAMBARANATHAN RAJAMANIKKAM, AATREYI BAL, KOUSHIK CHAKRABORTY, and SANGHAMITRA ROY, USU BRIDGE Lab, Electrical and Computer Engineering, Utah State University

The emergence of GPGPU applications, bolstered by flexible GPU programming platforms, has created a tremendous challenge in maintaining high energy efficiency in modern GPUs. In this article, we demonstrate that customizing a Streaming Multiprocessor (SM) of a GPU at a lower frequency is significantly more energy efficient compared to employing DVFS on an SM designed for a high-frequency operation. Using a system-level CAD technique, we propose *SSAGA—Streaming Multiprocessors Synthesized for Asymmetric GPGPU Applications*—an energy-efficient GPU design paradigm. SSAGA creates architecturally identical SM cores, customized for different voltage-frequency domains. Our rigorous cross-layer methodology demonstrates an average of 20% improvement in energy efficiency over a spatially multitasking GPU across a range of GPGPU applications.

Categories and Subject Descriptors: C.1.4 [Processor Architectures]: Parallel Architectures; I.3.1 [Computer Graphics]: Hardware Architectures—Graphics processors

General Terms: Design, Performance, Efficiency

Additional Key Words and Phrases: GPU, DVFS, custom design, power gating, decreased power consumption, improved energy efficiency, decreased peak chip temperature

ACM Reference Format:

Shamik Saha, Prabal Basu, Chidhambaranathan Rajamanikkam, Aatreyi Bal, Koushik Chakraborty, and Sanghamitra Roy. 2017. SSAGA: SMs synthesized for asymmetric GPGPU applications. *ACM Trans. Des. Autom. Electron. Syst.* 22, 3, Article 49 (April 2017), 20 pages.
DOI: <http://dx.doi.org/10.1145/3014163>

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 [Lucas et al. 2013]. Consequently, GPUs are already close to the abominable *power wall* [Lee et al. 2011]. Considering the massive popularity of the mobile devices running GPGPU applications [Shebanow 2013; Akenine-Moller and Strom 2008], it is of utmost importance to develop energy-efficient systems that can

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

Authors' addresses: S. Saha, 112 Station Drive, Morrisville, NC, 27560; email: shamik@synopsys.com; P. Basu, C. Rajamanikkam, A. Bal, K. Chakraborty, and S. Roy, 310, Bridge Lab, David G. Sant Innovation Building, 4100 Old Main Hill, Logan, UT 84321; emails: prabalbasu1989@yahoo.com, chidham@aggiemail.usu.edu, bal.aatreyi@gmail.com, koushik.chakraborty@usu.edu, sanghamitra.roy@usu.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2017 ACM 1084-4309/2017/04-ART49 \$15.00

DOI: <http://dx.doi.org/10.1145/3014163>

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 coscheduling 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 [Adriaens et al. 2012; Aguilera et al. 2014a, 2014b]. 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 [Basoglu et al. 2010; Lee and Kim 2009; Kim et al. 2008; Garg et al. 2009] and is also being considered as a viable option for GPUs. Governed by the runtime 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 runtime 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 article: (1) the emerging GPGPU applications achieve their best energy efficiencies at diverse VF domains, and (2) 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 Synthesized 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 coscheduled 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. Contributions

We make the following key contributions in this article:

- We illustrate the performance asymmetry of various emerging GPGPU applications (Section 2.1).
- We demonstrate the energy benefit of a GPU customized for specific VF domains compared to a DVFS-scaled GPU designed for the nominal frequency (Section 2.2).
- We propose SSAGA, a novel energy-efficient GPU design paradigm, that amalgamates the advantages of architecturally identical SMs with the low power benefit of customized circuit design (Section 3).
- We build a robust *Simultaneous Perturbation Stochastic Approximation* framework for determining the optimal VF domains for our system (Section 3.3.5).
- We propose a couple of workload creation and scheduling algorithms to further boost the energy efficiency of the GPU by optimal assignment of an application to its VF domain (Section 3.4).
- Using a rigorous cross-layer methodology (Section 4), we demonstrate an average of 30% improvement in power consumption and 20% improvement in energy efficiency over a spatially multitasking GPU across a range of GPGPU applications (Section 5).

Fig. 1. Performance degradation (lower is better).

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), which subsequently motivates 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 1 illustrates such variation in performance degradation when various benchmarks from AMD’s Accelerated Parallel Processing (APP) SDK suite [app 2016] run at 525MHz and 350MHz. The values are measured with respect to the performance at 700MHz. 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 [Chakraborty and Roy 2011; Bokhari et al. 2014]. Using a meticulous analysis, we demonstrate the limited potential of DVFS in improving the

Fig. 2. Energy efficiency degradation with DVFS (lower is better).

energy efficiency of a GPU. A DVFS engine sporadically 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 runtime adaptations cannot alter the inherent device characteristics (threshold voltages and gate size distribution, for example) that 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 [Valverde 2009] using a 32nm 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 ($1\times$ to $0.9\times$ of the nominal voltage) and choose the design with minimum power 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 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 on 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 $\sim 50\%$ degradation in energy efficiency at 33% of the nominal frequency. A primary reason for a custom-designed GPU to be more energy efficient than a DVFS-enabled GPU is the fact that it consumes less dynamic power. Figure 3 illustrates that the custom-designed paradigm consumes significantly less dynamic power than a DVFS-augmented GPU. We notice that the custom-designed GPU consumes $\sim 32\%$

Fig. 3. Dynamic power consumed by custom-designed GPU compared to DVFS-enabled GPU.

less dynamic power than the DVFS-scaled GPU at 33% of the nominal frequency. The custom-designed GPU is likely to consume even less dynamic power and be more energy efficient than a DVFS-enabled GPU at lower technology nodes, due to cessation of Dennard scaling.

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.

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 that 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 workloads¹ with varying degrees of memory/compute intensity. *To show the data for Sections 5.3, 5.4, and 5.5, we consider a subset of 10 workloads that represent the overall variation of the entire workload space.* Our system implements

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

Fig. 4. Improvement in energy efficiency versus operating frequency (negative values signify degradation).

memory isolation among the coexecuting applications to prevent malicious applications from accessing the memory space of other applications [Adriaens et al. 2012]. The SMs are distributed evenly among the applications [Adriaens et al. 2012]. 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.

3.2. SSAGA: An Optimization Problem

3.2.1. Energy Efficiency Heterogeneity. Figure 4 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 4 demonstrates some of the applications, representing three distinct trends. For example, Figure 3 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). A third category of applications (e.g., *PrefixSum*) in Figure 3 exhibit a nonmonotonic 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 ($k \leq n$) applications can be coscheduled 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, four coscheduled applications out of a total of 20, with four $\{V, F\}$ tuples, give rise to 29,767,680 possible solutions. Hence, finding the optimal solutions to maximize the energy efficiency is nontrivial 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 a *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 coscheduled, 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

$$D = \{(V_1, F_1), (V_2, F_2), \dots, (V_p, F_p)\}. \quad (1)$$

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

$$v = \{(V_i, F_i) : (V_i, F_i) \in D\}, 1 \leq i \leq k. \quad (2)$$

Hence, the objective is to

$$\text{Maximize} \quad \sum_{i=1}^k \Delta \left(\frac{Perf}{Power} \right) (v_i), \quad (3)$$

where k is the number of coscheduled applications. Both Equations (2) and (3) have the same upper limit, as the number of coscheduled 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 coscheduled applications to be greater than P_0 . Hence, the constraint equation is

$$Y = \frac{1}{k} \sum_{i=1}^k \left(\frac{Perf(F_i)}{Perf(F_n)} \right) \geq P_0. \quad (4)$$

From Equation (4), we can conclude that P_0 can only be a fractional value greater than 0 and less than or 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

$$F_l \leq F_i \leq F_n. \quad (5)$$

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

$$\text{Maximize} \quad \sum_{i=1}^k \Delta \left(\frac{Perf}{Power} \right) (v_i)$$

$$\text{such that } \frac{1}{k} \sum_{i=1}^k \left(\frac{\text{Perf}(F_i)}{\text{Perf}(F_n)} \right) \geq P_0$$

$$F_l \leq F_i \leq F_n.$$

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)*, *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) and formulate the objective function as an MILP problem. However, with the advent of intra-SM multitasking [Wang et al. 2016; Xu et al. 2016] 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)*, *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 [Spall 1998]. 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 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:  $V \leftarrow v_{in}$ 
2:    $a, c, A, \alpha, \gamma$  ▷ nonnegative 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  $\Delta_j$  ▷ perturbation vector
7:   Evaluate Loss function,  $L(V) = \sum_{i=1}^k \Delta PPW(V \pm \Delta_j c_j)$ 
8:   Generate gradient approximation,
        $g_j(\hat{V}) = (L(V))(\Delta_j)$ 
9:    $V_{new} = V - a_j \cdot g_j(\hat{V})$ 
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
14:       $O \leftarrow O_{new}$   $V \leftarrow V_{new}$ 
15:    end if
16:  end if
17: end for

```

Initialization: We begin the algorithm by creating our initial solution, $v_{in} = (V_i, F_i)$, 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 [Spall 1998]. 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 [Spall 1998].

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 ($L(V)$) [Spall 1998]. The loss function is the same as Equation (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 Spall [1992]. 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 SPSSA to generate the elements of Δ_j (as they have infinite inverse moments) [Spall 1998]. 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 SPSSA 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 four applications from a set of 17 applications to create a workload. Table II 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 suboptimal energy efficiency of the GPU. Motivated by Figure 4, which 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

²We consider 10 representative workloads that exhibit all the characteristics of the entire workload space.

Fig. 5. 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.

and determine the number of ALU instructions (AI), as well as the number of memory accesses (MAs). 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 5 depicts the compute/memory intensities of all the applications.

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 10–20). 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 21–32). The new workloads (using Algorithm 2) are shown in Table III.

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.

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 [Ubal et al. 2012] to perform architectural simulation of AMD's Evergreen architecture GPU—Radeon R9. Table I lists the architectural parameters for our GPU. Figure 6 shows the VF domains we consider for this work. We create four VF domains (also referred to as SM clusters), each containing 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

Table I. AMD's Radeon R9 GPU Configuration

Parameters	Values
Frequency	700MHz
Number of SMs	80
Thread group size	64
Local memory	32KB, latency: 2 cycles
L2 cache	8x256KB, latency: 20ns
Device Memory	B/W: 264GB/s, latency: 100ns

ALGORITHM 2: Algorithm to Create and Schedule Workloads

```

1: STEP 1: To determine whether an application is compute or memory intensive.
2: Initialize:  $I_{th}$  ▷ threshold value of (memory-accesses)/(ALU-accesses).
3: array compute ▷ setofapplicationswith(memory – accesses)/(ALU – accesses) <  $I_{th}$ 
   (ascending order).
4: array memory ▷ setofapplicationswith(memory – accesses)/(ALU – accesses) >  $I_{th}$ 
   (ascending order).
5: Workloads[id] ▷ user-defined datatype for workloads.
6: voltage[level] ▷ Voltage levels for a particular workload.
7: boolean occupied[level] ▷ Tracks if a voltage level is occupied.
8: STEP2: Rearrange Workloads.
9: Initialize:  $id = level = 0$ ; ▷ Assume as level increases, voltage decreases
10: for  $k = 0 \rightarrow (sizeof)arraycompute/(sizeof)compute[0]$  do
11:   if Workloads[id].occupied[level] == false then
12:     Assign task to Workloads[id].voltage[level]
13:     Workloads[id].occupied[level] = true
14:   end if
15:   Increment id to point to next workload.
16:   if id == Maximum number of workloads then
17:     Make id point to first workload.
18:     Make level point to next lower voltage.
19:   end if
20: end for
21: Initialize:  $id = 0, level = 4$ ; ▷ Assume 4 VF levels.
22: for  $k = (sizeof)arraymemory/(sizeof)memory \rightarrow 0$  do
23:   if Workloads[id].occupied[level] == false then
24:     Assign task to Workloads[id].voltage[level]
25:     Workloads[id].occupied[level] = true
26:   end if
27:   Increment id to point to next workload.
28:   if id == Maximum number of workloads then
29:     Make id point to first workload.
30:     Make level point to next higher voltage.
31:   end if
32: end for

```

SMs as it will have the following drawbacks: (1) modest area and power overheads due to voltage regulators and (2) challenges in physical design of the GPU. For each benchmark, we run the GPU kernel for one iteration, once for each of the multiple operating frequencies. The output statistics from Multi2Sim are used as inputs to our SPSA-based optimization framework.

4.2. Workloads

We use several emerging GPGPU applications from AMD's Accelerated Parallel Processing (APP) SDK suite [app 2016]. We create 10 workloads for our analysis, where

Fig. 6. SM clusters in GPU.

Table II. ObPAS-Based Workloads: *C* and *M* in the Parenthesis Denote Compute-Intensive and Memory-Intensive Applications, 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)

each workload comprises four applications. We initialize I_{th} (in Algorithm 2) with an appropriate value and determine our set of compute- and memory-intensive benchmarks. Tables II and III 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 [Valverde 2009] using the Synopsys Design Compiler and a 32nm 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

Table III. EnPAS-Based Workloads: *C* and *M* in the Parenthesis Denote Compute-Intensive and Memory-Intensive Applications, 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)

power model we consider, an SM customized 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 [Huang et al. 2004]. We model the steady-state temperature characteristics of the underlying GPU architecture and conduct HotSpot simulations at the grid level to generate 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 [Huang et al. 2007].

5. EXPERIMENTAL RESULTS

In this section, we figure out the different voltage frequency levels for our SSAGA design paradigm (Section 5.1) and 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 Hsu et al. [2003] and Macii et al. [2003].

5.1. V-F Domains

In order to determine the optimum voltage-frequency levels for SSAGA, we generate around 2,000 unique workloads that 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 IV. We have obtained the V-F domains in Table IV using our SPSA framework in Section 3.3.5. Clusters 2, 3, and 4 have the same voltage in order to ensure that the circuit satisfies timing.

Table IV. SSAGA Solution for VF Domain Configurations

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

5.2. Comparative Schemes

- **Spatial Multitasking ObPAS (SPAT-MT-ObPAS):** This scheme coschedules multiple applications on the GPU with an *even* distribution of the SMs [Adriaens et al. 2012]. All the coscheduled applications run at the nominal frequency (i.e., 700MHz) 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 the DVFS technique in Mei et al. [2013] 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 700MHz. 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 an 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 Lee et al. [2011]. 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 one 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*, and *MatrixTranspose* show less than 0.7% degradation in performance when the number of SMs is 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 [Wang et al. 2011].

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

Fig. 7. Normalized power consumption (lower is better).

- **SSAGA-EnPAS:** This is also our proposed scheme that combines an 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 7 depicts the power consumptions of the comparative schemes when subjected to ObPAS and EnPAS policies. Figure 7(a) and Figure 7(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 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 7(a), SSAGA-ObPAS shows maximum energy saving among non-PG schemes for W10 and W50 as they are extremely memory intensive. On 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 reduce 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 8 shows the energy efficiencies of the comparative schemes when subjected to ObPAS and EnPAS policies. Figure 8(a) and Figure 8(b) are normalized to Spatial Multitasking ObPAS GPU (SPAT-MT-ObPAS) and 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 better performance per watt, compared to other schemes. Such an outcome stems from (1) optimal application to VF domain assignments and (2) reduced energy consumption of the customized SMs at the lower operating frequencies. However, for workload W20,

(a) Normalized Energy Efficiency (ObPAS policy).

(b) Normalized Energy Efficiency (EnPAS policy).

Fig. 8. Normalized energy efficiency (higher is better).

(a) Normalized Peak Temperature (ObPAS policy).

(b) Normalized Peak Temperature (EnPAS policy).

Fig. 9. Normalized peak temperature (lower is better).

SSAGA-ObPAS is slightly less energy efficient than SPAT-MT-ObPAS and SPAT-MT-DVFS-ObPAS. This is because W_{20} 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 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.³ 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.

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 9 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 7. SSAGA-ObPAS, SSAGA-ObPAS-PG, SSAGA-EnPAS, and SSAGA-EnPAS-PG show

³However, the energy efficiency of SPAT-MT-EnPAS is only 0.5% more than SPAT-MT-ObPAS.

improvements in the peak temperature from 2.3% to 4.7%, 3.3% to 8.7%, 3.6% to 8.1% and 4.7% to 10.7%, respectively.

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 ISAs (ASISAs) have the potential to outperform homogeneous processors in terms of performance/power or both. Shelepov et al. [2009] 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. Li et al. [2010] have considered heterogeneous cores along with nonuniform ISAs in their work. They propose several algorithms that have been implemented in an actual OS. Kumar et al. [2003] discuss the rise in processor power consumption and the potential of ASISA to reduce the processor power consumption. Kim et al. [2014] use ARM's big.LITTLE architecture and modify the Linux kernel to make it aware of the processor utilization.

Several existing works also aim at improving the energy efficiency of the GPUs. These works can be broadly classified into two categories: (1) the efficacy of DVFS in modern GPUs and (2) employing multitasking to increase resource utilization in GPUs. In the first category, Ge et al. [2013] have shown that DVFS, when employed in GPUs, exhibits greater savings in terms of power and energy compared to CPUs. Mei et al. [2013] propose to save energy by operating the GPU cores at their optimal frequency and scaling down the supply voltage. However, the efficacy of scaling core and memory frequencies depends on the application characteristics. Jiao et al. [2010] have shown that the performance and power consumption of GPU kernels are largely determined by two factors: (1) the issue rate of the instructions and (2) the ratio of global memory transactions to computation instructions. Nugteren et al. [2014] 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. Komoda et al. [2013] 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. You and Chung [2014] investigate the correlation between frames per second (FPS) and GPU utilization to design a Quality of Service-aware DVFS algorithm for embedded GPUs. Wang and Ranganathan [2014] propose a dynamic frequency scaling technique for GPU platforms based on a feedback controller and improve throughput.

In the second category, Adrians et al. [2012] have shown that *spatial multitasking* offers a better speedup compared to *cooperative multitasking*. Aguilera et al. [2014a] allocate resources to the workload (composed of several applications), depending on the workload characteristics and the operating frequency of the SMs. Reservation of cores [Kato et al. 2011; Joo and Shin 2014], hardware leaky-bucket-based thread-block interleaving [Liang et al. 2015], and resource sharing between applications [Aguilera et al. 2014b] are some of the techniques that are able to reduce power consumption and improve performance in GPUs. Jiao et al. [2015] have proposed a mix of concurrent execution and DVFS to achieve appreciable performance improvement. *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. [2016] and Xu et al. [2016] have demonstrated that intra-SM multitasking offers better performance and throughput over a spatially multitasking GPU. However, the authors did not confirm whether 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, the 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 onto 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.

7. CONCLUSION

In this article, we propose *SSAGA—Streaming Multiprocessors Synthesized 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

- Advanced Micro Devices (AMD). 2016. AMD Accelerated Parallel Processing (APP) Software Development Kit. <http://developer.amd.com/sdks/amdappsdk/>.
- J. Adriaens, K. Compton, N. S. Kim, and M. J. Schulte. 2012. The case for GPGPU spatial multitasking. In *Proceedings of High Performance Computer Architecture (HPCA'12)*. 79–90.
- P. Aguilera, J. Lee, A. F. Farahani, K. Morrow, M. J. Schulte, and N. S. Kim. 2014a. Process variation-aware workload partitioning algorithms for GPUs supporting spatial-multitasking. In *IEEE/ACM Design Automation & Test in Europe (DATE'14)*. 1–6.
- P. Aguilera, K. Morrow, and N. S. Kim. 2014b. QoS-aware dynamic resource allocation for spatial-multitasking GPUs. In *IEEE/ACM Asia and South Pacific-Design Automation Conference (ASP-DAC'14)*. 726–731.
- T. Akenine-Moller and J. Strom. 2008. Graphics processing units for handhelds. In *Proceedings of the IEEE*. 779–789.
- M. Basoglu, M. Orshansky, and M. Erez. 2010. NBTI-aware DVFS: A new approach to saving energy and increasing processor lifetime. In *ACM International Symposium on Low Power Electronic Devices (ISLPED'10)*. 253–258.
- H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran. 2014. darkNoC: Designing energy-efficient network-on-chip with multi-vt cells for dark silicon. In *IEEE/ACM Design Automation Conference (DAC'14)*. 161:1–161:6.
- K. Chakraborty and S. Roy. 2011. Topologically homogeneous power-performance heterogeneous multicore systems. In *IEEE/ACM Design Automation & Test in Europe (DATE'11)*. 1–6.
- S. Garg, D. Marculescu, R. Marculescu, and Ü. Y. Ogras. 2009. Technology-driven limits on DVFS controllability of multiple voltage-frequency island designs: A system-level perspective. In *IEEE/ACM Design Automation Conference (DAC'09)*. 818–821.
- R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong. 2013. Effects of dynamic voltage and frequency scaling on a K20 GPU. In *IEEE International Conference on Parallel Processing (ICPP'13)*. 826–833.
- S. Hsu, A. Alvandpour, S. Mathew, S.-L. Lu, R. Krishnamurthy, and S. Borkar. 2003. A 4.5-GHz 130-nm 32-kB L0 cache with a leakage-tolerant self reverse-bias bitline scheme. *IEEE J. Solid-State Circuits* 38, 5, 755–761.
- W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, and K. Skadron. 2007. An Improved Block-Based Thermal Model in Hotspot 4.0 with Granularity Considerations. In *Proceedings of the 6th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD'07)*. 1–10.

- W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy. 2004. Compact thermal modeling for temperature-aware design. In *IEEE/ACM Design Automation Conference (DAC'04)*. 878–883.
- Q. Jiao, M. Lu, H. P. Huynh, and T. Mitra. 2015. Improving GPGPU energy-efficiency through concurrent kernel execution and DVFS. In *Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO'15)*. 1–11.
- Y. Jiao, H. Lin, P. Balaji, and W. Feng. 2010. Power and performance characterization of computational kernels on the GPU. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing (GREENCOM-CPSCOM'10)*. 221–228.
- J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim. 2011. Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling. In *Parallel Architectures and Compilation Techniques (PACT'11)*. 111–120.
- W. Joo and D. Shin. 2014. Resource-constrained spatial multi-tasking for embedded GPU. In *ICCE*. 339–340.
- S. Kato, K. Lakshmanan, R. R. Rajkumar, and Y. Ishikawa. 2011. TimeGraph: GPU scheduling for real-time multi-tasking environments. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference (USENIXATC'11)*. 17–30.
- M. Kim, K. Kim, J. Geraci, and S. Hong. 2014. Utilization-aware load balancing for the energy efficient operation of the big.LITTLE processor. In *IEEE/ACM Design Automation & Test in Europe (DATE'14)*. 223:1–223:4.
- W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. 2008. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of High Performance Computer Architecture (HPCA'08)*. 123–134.
- T. Komoda, S. Hayashi, T. Nakada, and S. Miwa. 2013. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *IEEE International Conference on Computer Design (ICCD'13)*. 349–356.
- R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen. 2003. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Microarchitecture, 2003 (MICRO-36)*. 36, 81–91.
- J. Lee and N. S. Kim. 2009. Optimizing throughput of power and thermal-constrained multicore processors using DVFS and per-core power-gating. In *IEEE/ACM Design Automation Conference (DAC'09)*. 47–50.
- T. Li, P. Brett, R. Knauerhase, D. Koufaty, D. Reddy, and S. Hahn. 2010. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *Proceedings of High Performance Computer Architecture (HPCA'10)*. 9–14.
- Y. Liang, H. P. Huynh, K. Rupnow, R. S. M. Goh, and D. Chen. 2015. Efficient GPU spatial-temporal multi-tasking. *IEEE Trans. Parallel Distrib. Syst.* 26, 3, 748–760.
- J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink. 2013. How a single chip causes massive power bills GPU-SimPow: A GPGPU power simulator. In *International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*.
- A. Macii, E. Macii, and M. Poncino. 2003. Improving the efficiency of memory partitioning by address clustering. In *IEEE/ACM Design Automation and Test in Europe (DATE'03)*. 18–23.
- X. Mei, L. S. Yung, K. Zhao, and X. Chu. 2013. A measurement study of GPU DVFS on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower'13)*. 10:1–10:5.
- C. Nugteren, G.-J. van den Braak, and H. Corporaal. 2014. Roofline-aware DVFS for GPUs. In *Proceedings of International Workshop on Adaptive Self-Tuning Computing Systems*. 8–10.
- M. Shebanow. 2013. An evolution of mobile graphics. *Keynote Talk at High Performance Graphics*.
- D. Shelepov, J. C. S. Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar. 2009. HASS: A scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev.* 43, 2, 66–75.
- J. Spall. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Trans. Automatic Control* 37, 3, 332–341.
- J. C. Spall. 1998. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins Apl. Technical Digest* 19, 4 (1998), 482–492.
- R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli. 2012. Multi2Sim: A simulation framework for CPU-GPU computing. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. 335–344.
- Universidad de Costa Rica. 2009. *Theia GPU*. http://opencores.org/project,theia_gpu.
- P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng. 2011. Power gating strategies on GPUs. *ACM Trans. Archit. Code Optim.* 8, 3, 13:1–13:25.

- Y. Wang and N. Ranganathan. 2014. A feedback, runtime technique for scaling the frequency in GPU architectures. In *Proceedings of the 2014 IEEE Computer Society Annual Symposium on VLSI (ISVLSI'14)*. 430–435.
- Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo. 2016. Simultaneous multikernel GPU: Multitasking throughput processors via fine-grained sharing. In *Proceedings of High Performance Computer Architecture (HPCA'16)*. 358–369.
- Q. Xu, H. Jeon, K. Kim, W. Roo, and M. Annavaram. 2016. Warped-slicer: Efficient intra-SM slicing through dynamic resource partitioning for GPU multiprogramming. In *ISCA*. 1–13.
- D. You and K.-S. Chung. 2014. Quality of service-aware dynamic voltage and frequency scaling for embedded GPUs. In *IEEE Computer Architecture Letters*. 66–69.

Received May 2016; revised September 2016; accepted October 2016