A Survey of Methods for Analyzing and Improving GPU Energy Efficiency

SPARSH MITTAL, Iowa State University
JEFFREY S. VETTER, Oak Ridge National Laboratory and Georgia Tech

Recent years have witnessed phenomenal growth in the computational capabilities and applications of GPUs. However, this trend has also led to a dramatic increase in their power consumption. This article surveys research works on analyzing and improving energy efficiency of GPUs. It also provides a classification of these techniques on the basis of their main research idea. Further, it attempts to synthesize research works that compare the energy efficiency of GPUs with other computing systems (e.g., FPGAs and CPUs). The aim of this survey is to provide researchers with knowledge of the state of the art in GPU power management and motivate them to architect highly energy-efficient GPUs of tomorrow.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; I.3.1 [**Computer Graphics**]: Graphics Processor; H.3.4 [**Systems and Software**]: Performance Evaluation (Efficiency and Effectiveness); C.0 [**Computer Systems Organization**]: System Architectures

General Terms: Experimentation, Management, Measurement, Performance, Analysis

Additional Key Words and Phrases: GPU (graphics-processing unit), energy saving, power management, energy efficiency, architecture techniques, power model, green computing

ACM Reference Format:
Sparsh Mittal and Jeffrey S. Vetter. 2014. A survey of methods for analyzing and improving GPU energy efficiency. ACM Comput. Surv. 47, 2, Article 19 (July 2014), 23 pages.
DOI: http://dx.doi.org/10.1145/2636342

1. INTRODUCTION

As we enter into the post-petascale era, the requirements of data processing and computation are growing exponentially. To meet this requirement, researchers have moved from serial execution platforms to high-performance computing (HPC) platforms, such as multicore processors, FPGAs, and GPUs. GPUs, in particular, have been widely used for HPC applications due to their extremely high computational powers, and a large fraction of supercomputers in the Top500 list use GPUs to achieve unprecedented computational power [Top500 2013]. Thus, GPUs have become an integral part of today's mainstream computing systems.

The high-performance demands on GPUs, however, have influenced their design to be optimized for higher performance, even at the cost of large power consumption.

The work was performed when Sparsh Mittal was at Iowa State University. The article has been authored by Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract #DE-AC05-00OR22725 to the U.S. government. Accordingly, the U.S. government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. government purposes. This research is sponsored by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy.
Authors' address: S. Mittal and J. S. Vetter, 1 Bethel Valley Road, Future Technologies Group, Oak Ridge National Laboratory, Building 5100, MS-6173, Tennessee, USA 37830; email: {mittals, vetter}@ornl.gov.

DOI: http://dx.doi.org/10.1145/2636342

Hence, recent years have witnessed a marked increase in power consumption of GPUs. The elevated levels of power consumption of GPUs have a significant impact on their reliability, economic feasibility, architecture design, performance scaling, and deployment into a wide range of application domains. As a case in point, supercomputers built with CPU-GPUs consume a huge amount of power; for example, the Titan supercomputer consumes 8.2MW of power [Top500 2013]. Further, it has been estimated that an exascale machine, built with the technology used in today's supercomputers, will consume several gigawatts of power [Miller 2013]. To manage such high levels of power dissipation and continue to scale performance, power management techniques are essential for both CPUs and GPUs. While the area of power management in CPUs has been actively researched over the years, the area of power management in GPUs has yet to be fully explored. For these reasons, understanding the state of the art in GPU power management is extremely important for researchers to propose even more effective solutions to address the power challenges and design "green" GPUs of tomorrow.

In this article, we present a survey of research works aimed at analyzing and improving the energy efficiency of GPUs. We classify the techniques based on several parameters to provide insights into their important features. We also review the research works that compare the energy efficiency of GPUs with other computing systems such as CPUs, Cell processors, and FPGA. We believe that this will enable the readers to judge the energy efficiency of GPUs vis-à-vis alternate computing platforms and make important decisions.

Since it is infeasible to review all the research ideas proposed in the literature, we adopt the following approach to limit the scope of the article. We include only those studies that analyze GPU power consumption and the techniques that have been evaluated based on GPU energy efficiency. We do not include those studies that have been shown to improve only performance and not energy efficiency, even though the performance improvement is likely to translate to better energy efficiency. We include application-level and architectural-level techniques and not circuit-level techniques for improving energy efficiency. Further, since different techniques have been evaluated using different experimentation platform and methodologies, we only focus on their key ideas and generally do not present their quantitative results.

This article is organized as follows. Section 2 reviews the GPU terminology and also highlights the need for power management. Section 3 reviews the studies on comparing GPU energy efficiency with that of other computing systems. Section 4 discusses some power management techniques in detail. In both of these sections, we first provide an overview and classification of the methods and then discuss some of the techniques in detail. We finally provide concluding remarks and future research trends in Section 5.

2. BACKGROUND

2.1. GPU Terminology and Sources of Power Consumption

Recently, several researchers have proposed models and tools for measurement and estimation of GPU power consumption [Hong and Kim 2010; Ramani et al. 2007; Nagasaka et al. 2010; Sheaffer et al. 2005a; Zhang et al. 2011; Jiao et al. 2010; Zhang et al. 2011; Chen et al. 2011; Suda and Ren 2009; Enos et al. 2010; Wang and Ranganathan 2011; Ren 2011; Ren et al. 2012; Luo and Suda 2011; Pool et al. 2010; Stolz et al. 2010; Li et al. 2011; Wang and Chen 2012; Collange et al. 2009; Wang et al. 2010; Vialle et al. 2011; Kasichayanula et al. 2012]. These models provide insights into the working of GPUs and relative contribution of different components in the total power consumption. In what follows, we briefly review the GPU architecture,

terminology, and sources of power consumption as relevant for this article and refer the reader to the aforementioned works for more details.

A GPU has several streaming multiprocessors, each of which has multiple cores. For example, NVIDIA GeForce GTX 590 has dual GPUs, where each GPU has 16 streaming multiprocessors (SMs); each of these SMs has 32 cores, for a total of 512 cores in each GPU and 1,024 cores in the overall GTX 590 graphics card [GeForce GTX 590 2013]. The cores of a typical GPU are composed of ALUs, thread schedulers, load/store units, scratchpad memory, register file and caches, and so forth. A GPU is designed for stream or throughput computing, which has little data reuse, and hence, a GPU has a much smaller sized cache (e.g., 16KB L1 and 256KB L2 [Wong et al. 2010]) than a typical CPU. The GPU is used as a coprocessor with a CPU, and in such cases, the GPU is referred to as the "device" and the CPU as the "host." A GPU has its own device memory of a few GBs (gigabytes), and it is connected to the host through a PCI-Express (PCIe) bus. A GPU is programmed as a sequence of *kernels*. The code is executed in groups of 32 threads, called a *warp*. CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language) are widely used interfaces for programming GPUs.

The power consumption of GPUs can be divided into two parts, namely, leakage power and dynamic power. The dynamic power is a function of operating temperature and circuit technology. Leakage power is consumed when the GPU is powered, even if there are no runtime activities. The dynamic power arises from switching of transistors and is determined by the runtime activities. Different components such as SMs and memories (e.g., local, global, shared, etc.) contribute to this power consumption.

2.2. Need for Improving Energy Efficiency of GPUs

GPU power management is extremely important for the following reasons.

2.2.1. Addressing Inefficient Resource Usage. To meet the worst-case performance requirements, the chip designers need to overprovision the computing resources of GPUs; however, on average, the utilization of these resources remains low. Also, in several applications, memory bandwidth of GPUs acts as a performance bottleneck [Hong and Kim 2010; Daga et al. 2011; Cebrian et al. 2012; Spafford et al. 2012], due to which the cores are not fully utilized, which leads to energy inefficiency. Further, unlike massively parallel applications, regular parallel applications do not scale well beyond a certain number of cores, and hence, a large amount of power is wasted in idle cores or in synchronization. Finally, GPUs are increasingly being used in cloud infrastructure and data centers [Amazon EC2 2013], which experience highly varying usage patterns. Thus, dynamic power management techniques can offset these sources of inefficiencies by using runtime adaption.

2.2.2. Ensuring Reliability. Large power consumption has a significant effect on the reliability of the computing systems. A 15°C rise in temperature increases the component failure rates by up to a factor of two [Anderson et al. 2003]. The device failures may lead to system malfunction, and as GPUs become increasingly employed in supercomputers and business services, system malfunction may have a serious economic impact. For example, the service cost of merely 1 hour of downtime in brokerage operations and credit card authorization can be $6,450,000 and $2,600,000, respectively [Feng 2003]. Thus, since the performance requirements grow at a much faster pace than the effectiveness of cooling solutions, power management techniques are extremely important to ensure longevity and reliability.

2.2.3. Providing Economic Gains. For every watt of power dissipated in the computing equipment, an additional 0.5 to 1W of power is consumed by the cooling system itself [Patel et al. 2003], and with an increasing ratio of cooling power to computing

power, compaction of devices is inhibited, which results in increased operation costs. Due to these trends, in recent years, the energy cost of high-performance computing clusters has been estimated to contribute more than the hardware acquisition cost of IT equipment itself [Bianchini and Rajamony 2004; Mittal 2012].

2.2.4. Enabling Performance Scaling. The power challenges are expected to present the most severe obstacle to performance scaling, and it has been shown that thermal and leakage power constraints may disallow simultaneously using all the cores of a massively parallel processor [Esmaeilzadeh et al. 2013]. Large power consumption may necessitate complex cooling solutions (e.g., liquid cooling), which may increase chip complexity and offset the benefits of performance boost obtained by using GPUs.

2.2.5. Enabling Deployment in Wide Range of Applications. The energy efficiency of GPUs, relative to other alternatives (e.g., CPUs, FPGAs), will have a crucial role in deciding its adoption in various application domains. In recent years, ongoing technological innovations have greatly improved other computing systems. As we show in Section 3, for several applications, FPGAs have been found to have better performance and energy efficiency than GPUs. Moreover, while a few initial works have reported orders of magnitude difference in performance of GPUs and CPUs, other researchers who apply careful optimization on *both* CPUs and GPUs have reported much lower speedups of GPUs over CPUs, typically in the range of 0.7× to 15× [Lee et al. 2010; Zou et al. 2012; Chandramowlishwaran et al. 2010]. Thus, to maintain their competitiveness and justify their use in product design, GPUs must exhibit high energy efficiency.

2.2.6. Achieving the Goals of Sustainable Computing. It has been estimated that the carbon emission of ICT (information and communication technology) will triple from 2002 to 2020 [Smarr 2010], and hence, concerns for the environment will force policymakers and researchers to place higher emphasis on energy efficiency in the design of future computing systems. Thus, improving the energy efficiency of GPUs is also important for achieving the goals of sustainable computing.

3. RESEARCH WORKS ON ANALYZING GPU ENERGY EFFICIENCY

In this section, we review the research works that analyze energy efficiency of GPUs and compare it with that of other computing systems. We first present an overview and then discuss some of the research works in detail.

3.1. Overview

Modern GPUs consume a significant amount of power. The high-end GPUs, such as NVIDIA GeForce GTX 590 (40nm) and AMD Radeon HD 5970 (40nm), have a maximum power consumption of 365W [GeForce GTX 590 2013] and 294W [RADEON 2013], respectively. In contrast, Intel's Core i7-3770T (22nm) and Xeon E7-8870 (32nm) have a maximum power consumption of 45W and 150W, respectively [Intel Core i7 2013; Intel Xeon E7 2013]. Note, however, that for several applications, GPUs provide better performance than CPUs, which makes their energy efficiency better than those of CPUs.

In recent years, several researchers have compared the power consumption of GPUs with that of other computing systems such as CPUs, Cell, or FPGA. For certain applications and platforms, GPUs have been found to be more energy efficient than CPUs [Zandevakili et al. 2012; Huang et al. 2009; Anzt et al. 2011; Baker et al. 2007; Thomas et al. 2009; Hamada et al. 2009; McIntosh-Smith et al. 2012; Lange et al. 2009; Ghosh et al. 2012; Udagawa and Sekijima 2011; Zou et al. 2012; Hussain et al. 2011; De Schryver et al. 2011; Van Essen et al. 2012; Betkaoui et al. 2010; Timm et al. 2010; Goddeke et al. 2008; Scogland et al. 2010; Danalis et al. 2010; Chung et al. 2010; Keckler

et al. 2011; Brodtkorb et al. 2010; Chau et al. 2013; López-Portugués et al. 2011; Cong et al. 2011; Pedram et al. 2012; Chow et al. 2012; Wang et al. 2012], while for other applications, CPUs have been found to be more energy efficient [Chandramowlishwaran et al. 2010; Kestur et al. 2010]. Some researchers also discuss the conditions under which CPUs or GPUs may be more efficient [Datta et al. 2008; Anzt et al. 2010; Calandrini et al. 2012; Fowers et al. 2013; Maghazeh et al. 2013]. For example, Datta et al. [2008] show that taking into account the overhead of data communication between CPUs and GPUs can significantly degrade GPU energy efficiency and can make them less energy efficient than CPUs.

Similarly, some authors have found FPGAs to be more energy efficient than GPUs [Kestur et al. 2010; Hefenbrock et al. 2010; Baker et al. 2007; Thomas et al. 2009; Pauwels et al. 2012; Birk et al. 2012; Hussain et al. 2011; Hamada et al. 2009; Gohringer et al. 2011; Zou et al. 2012; Benkrid et al. 2012; De Schryver et al. 2011; Lange et al. 2009; Williams et al. 2008; Richardson et al. 2010; Lee et al. 2010; Van Essen et al. 2012; Brodtkorb et al. 2010; Chau et al. 2013; Cong and Zou 2009; Llamocca et al. 2011; Cong et al. 2011; Waidyasooriya et al. 2012; Chow et al. 2012; Wang et al. 2012; Struyf et al. 2014], while others have found GPUs to be more energy efficient [Duan et al. 2011]. Similarly, some researchers observe other computing systems such as Cell, DSP (digital signal processor), or ASIC to be more energy efficient than GPUs [Chung et al. 2010; Baker et al. 2007; Benkrid et al. 2012; Mu et al. 2011; Pedram et al. 2012].

From these works, it is clear that although for the majority of works, FPGAs are more energy efficient than GPUs and GPUs, in turn, are more energy efficient than CPUs, a single platform cannot be accepted as the most energy efficient for all possible applications. The results crucially depend on the devices and evaluation methodology used in the experiments.

3.2. Discussion

Keckler et al. [2011] discuss the level of energy efficiency required for building future exascale machines. They show that building an exascale machine with a power budget of 20MW requires an energy efficiency of 20 picoJoules (pJ) per floating point operation. In contrast, state-of-the-art CPUs and GPUs incur 1700pJ and 225pJ, respectively, for each floating-point operation. This shows that although the GPUs are more energy efficient than CPUs, their efficiency needs to be improved further to fulfill exascale challenge.

Chandramowlishwaran et al. [2010] compare the performance and energy efficiency of a GPU with a multicore CPU for the fast multipole method. They have observed that on applying suitable optimization and parallelization, the CPU is nearly 1.7× faster than a single GPU and achieves 0.75× the performance of two GPUs. In terms of energy efficiency, the CPU is nearly 2.4× and 1.8× as energy efficient as the systems accelerated using one or two GPUs, respectively.

Datta et al. [2008] compare the performance and energy efficiency of a GPU with a CPU for stencil (nearest-neighbor) computations. They observe that while use of a large number of cores gives a significant performance and power advantage to the GPU over the CPU, when it is used as an accelerator offload engine for applications that primarily run on the host CPU, the performance and energy efficiency are severely degraded due to limited CPU-GPU bandwidth and low reuse within GPU device memory. Since the GPU can access CPU memory only through a PCI-express (PCIe) bus, for applications that require larger on-board memory than what is available on the GPU, the performance is significantly degraded.

Huang et al. [2009] evaluate the energy efficiency and performance of a GPU for a scientific computing benchmark, namely, GEM software, which is used to compute the electrostatic potential map of macromolecules in a water solution. The

CPU code is parallelized using Pthread (POSIX threads). They observe that although the GPU consumes significantly higher power than the CPU, the execution time of the GPU version of code is much smaller, and hence, the EDP (energy-delay product) of the GPU implementation is orders of magnitude better than that of both the serial and parallel versions of CPU implementation. Moreover, using a single-precision code improves the energy efficiency of GPU even more.

McIntosh-Smith et al. [2012] compare the energy efficiency of a GPU with that of a multicore CPU for a molecular mechanics problem. They observe that of the different GPU implementations tested, the best implementation outperforms all CPU implementations in both performance and energy efficiency. Moreover, for the real-world case where the dataset become larger, the benefits of GPU become even larger.

Kestur et al. [2010] compare the energy efficiency and performance of a GPU with that of a multicore CPU and an FPGA, for double-precision floating-point programs from the Basic Linear Algebra Subroutines (BLAS) library. They have shown that the FPGA offers comparable performance to the GPU while providing significantly better energy efficiency. Moreover, the multicore CPU also provides better performance and energy efficiency than the GPU.

Llamocca et al. [2011] compare a GPU and an FPGA for a 2D FIR (finite-impulse response) filter program that has application in video processing. They observe that due to its higher frequency and ability to exploit massive parallelization present in the algorithm, the GPU provides better performance than the FPGA. However, the FPGA consumes up to an order of magnitude less energy than the GPU.

Baker et al. [2007] compare the energy efficiency and performance of matched filter on an FPGA, an IBM Cell, a GPU, and a CPU. Matched filter is a signal-processing kernel that is used for extracting useful data from hyperspectral imagery. Relative to the CPU, the speedup of other computing systems is calculated and then a comparison is made on the metric of speedup and speedup per kilowatt values. The authors observe that both the Cell and FPGA outperform the GPU in performance and energy efficiency. Further, the GPU provides better performance and energy efficiency than the CPU.

Hefenbrock et al. [2010] implement the Viola-Jones face detection algorithm using multi-GPU and compare its performance and power consumption with that of the fastest known FPGA implementation of the same algorithm. They observe that using 4-GPUs provides comparable performance with the design using a single FPGA, while the energy efficiency of the FPGA design was orders of magnitude better than that of the 4-GPUs-based design.

Lange et al. [2009] compare the performance and energy efficiency of a GPU with an FPGA and a multicore CPU for geometric algebra computations. They observe that the GPU is less energy efficient than the FPGA, but more efficient than the CPU. They also note that taking data transfer overhead into account degrades the energy efficiency of the GPU.

Hussain et al. [2011] compare the energy efficiency of a GPU with that of an FPGA and a CPU for the k-means clustering algorithm, which is used in data mining. They observe that the FPGA provides better performance and energy efficiency than the GPU. Also, the GPU shows much better energy efficiency than the CPU.

De Schryver et al. [2011] compare the energy efficiency of a GPU and a multicore CPU with that of a hybrid FPGA–CPU implementation for Monte Carlo option pricing with the Heston model. This program finds applications in financial domains. The hybrid FPGA–CPU implementation divides the work between FPGA and CPU, such that computation-intensive kernels are executed on FPGAs. They observe that compared to the GPU implementation, the hybrid FPGA–CPU implementation provides less performance but higher energy efficiency. Moreover, the GPU implementation excels CPU in both performance and energy efficiency.

Thomas et al. [2009] compare energy efficiency of a GPU with an FPGA and a multicore CPU for random number generation. The authors experiment with different random number generation programs and compute the geometric mean of energy efficiency (number of samples generated per joule of energy). They observe that FPGAs provide an order of magnitude better energy efficiency than the GPU. Moreover, the GPU is found to be an order of magnitude more energy efficient than the CPU.

Van Essen et al. [2012] implement the random forest classification problem used in machine learning on a GPU, an FPGA, and a multicore CPU. They observe that the FPGA provides the highest performance but requires a multiboard system even for modest size problems, which increases its cost. Further, on the performance-per-watt metric, the FPGA implementation is an order of magnitude better than the GPU implementation, which, in turn, is better than the CPU implementation.

Duan et al. [2011] compare a GPU with an FPGA and a multicore CPU on floating-point FFT implementation. For GPU and CPU implementation, they use standard libraries, and for FPGA, they develop their own implementation. They observe that the GPU is more energy efficient than the FPGA and the CPU for radix-2 FFT. They, however, observe a degradation in performance of the GPU for mixed-radix FFT.

Hamada et al. [2009] make a comparative study of a GPU, an FPGA, an ASIC, and a CPU for gravitational force calculation in N-body simulation in the context of astrophysics. They observe that the GPU outperforms the ASIC and the CPU in energy efficiency (performance per watt); however, its energy efficiency is an order of magnitude less than that of the FPGA.

Birk et al. [2012] compare the performance and energy efficiency of a GPU and an FPGA for 3D ultrasound computer tomography, which is used for medical imaging. They observe that the performance of the GPU is comparable with that of the FPGA; however, the FPGA offer much better energy efficiency.

Betkaoui et al. [2010] compare the energy efficiency of a GPU with an FPGA and a single and a multicore CPU for three throughput computing applications, viz. FFT, general (dense) matrix multiplication (GEMM), and Monte Carlo method (MCM). Of these, GEMM is limited by computations, FFT by memory latency, and MCM is embarrassingly parallel and hence is limited only by available parallelism. They use standard libraries for implementing these applications. They observe that for all three applications, the GPU outperforms the CPU on energy efficiency. Further, for GEMM, the GPU is more energy efficient than the FPGA, while for FFT and MCM, the FPGA is more energy efficient than the GPU. They note that the FPGA provides an advantage over the GPU for applications that exhibit poor data locality and a low memory bandwidth requirement.

Zou et al. [2012] compare a GPU with a CPU and an FPGA for the Smith-Waterman (S-W) algorithm. The S-W algorithm is used for performing pair-wise local sequence alignment in the field of bioinformatics. They highlight the need of making suitable optimizations on *all* three platforms for making meaningful comparisons. They observe that on the metric of performance per unit power, the FPGA is more energy efficient than the GPU, which in turn is more energy efficient than the CPU, although the advantage of the GPU over the CPU is small. The FPGA also provides higher performance than both the GPU and CPU.

Benkrid et al. [2012] compare a GPU with a CPU, an FPGA, and Cell BE (broadband engine) for the Smith-Waterman algorithm. They observe that on the energy-efficiency (performance-per-watt) metric, the FPGA and Cell BE perform better than the GPU, while the GPU performs better than the CPU. They further note that results also depend on the devices used and performance optimizations performed on each platform.

Pauwels et al. [2012] compare a GPU with an FPGA for the computation of phase-based optical flow, stereo, and local image features that are used in computer vision.

They observe that while the GPU offers better performance and accuracy than the FPGA, the FPGA is more energy efficient than the GPU.

Fowers et al. [2013] compare the energy efficiency of a GPU with that of an FPGA and a multicore CPU for the convolution problem, which has applications in digital signal processing. They observe that for very small signal sizes, the CPU is most energy efficient. However, as the signal size increases, the energy efficiency of the GPU and the FPGA increase, and for very large signal sizes, the FPGA outperforms the GPU in energy efficiency.

Mu et al. [2011] implement the high-performance embedded computing (HPEC) benchmark suite on a GPU and compare the performance and energy efficiency of the GPU with that of a DSP for this benchmark suite. This benchmark includes a broad range of signal-processing applications. They have observed that while the GPU provides at least an order of magnitude better performance than the DSP, its energy efficiency measured in terms of performance per watt is inferior to that of the DSP.

4. TECHNIQUES FOR IMPROVING GPU ENERGY EFFICIENCY

In this section, we discuss techniques for improving GPU energy efficiency.

4.1. Overview

For the purpose of this study, we classify the techniques into the following categories.

(1) DVFS (dynamic voltage/frequency scaling)-based techniques [Liu et al. 2011, 2012; Nam et al. 2007; Jiao et al. 2010; Lee et al. 2007, 2011; Ma et al. 2012; Cebrian et al. 2012; Sheaffer et al. 2005b; Chang et al. 2008; Wang et al. 2010; Ren 2011; Anzt et al. 2011; Ren et al. 2012; Lin et al. 2011; Zhao et al. 2012; Huo et al. 2012; Keller and Gruber 2010; Abe et al. 2012; Park et al. 2006; Leng et al. 2013; Paul et al. 2013]
(2) CPU-GPU workload division-based techniques [Takizawa et al. 2008; Rofouei et al. 2008; Ma et al. 2012; Luk et al. 2009; Liu et al. 2011, 2012; Hamano et al. 2009] and GPU workload consolidation [Li et al. 2011]
(3) Architectural techniques for saving energy in specific GPU components, such as caches [Wang et al. 2012; Lee et al. 2011; Lashgar et al. 2013; Arnau et al. 2012; Rogers et al. 2013; Lee and Kim 2012], global memory [Wang et al. 2013; Rhu et al. 2013], pixel shader [Pool et al. 2011], vertex shader [Pool et al. 2008], core data path, registers, pipeline and thread scheduling [Abdel-Majeed et al. 2013; Chu et al. 2011; Gebhart et al. 2011; Gilani et al. 2013; Jing et al. 2013; Yu et al. 2011; Abdel-Majeed and Annavaram 2013; Gilani et al. 2012; Sethia et al. 2013]
(4) Techniques that exploit workload variation to dynamically allocate resources [Jararweh and Hariri 2012; Liu et al. 2011; Lee et al. 2011; Hong and Kim 2010; Alonso et al. 2012; Cebrian et al. 2012; Wang and Ranganathan 2011; Keller and Gruber 2010]
(5) Application-specific and programming-level techniques for power analysis and management [Alonso et al. 2012; Chandramowlishwaran et al. 2010; Ren and Suda 2009; Datta et al. 2008; Jiao et al. 2010; Zandevakili et al. 2012; Anzt et al. 2011; Ren et al. 2012; Padoin et al. 2012; Wang et al. 2010; Ghosh et al. 2012; Dreßler and Steinke 2012; Zhang et al. 2012; Wang et al. 2010; Yang et al. 2012; Hsiao et al. 2013]

We now discuss these techniques in detail. As seen through the previous classification, several techniques can be classified into more than one group. For the sake of clarity, we discuss them in one group only.

4.2. DVFS-Based Techniques

DVFS is a well-known power management technique that works by dynamically adjusting the clock frequency of a processor to allow a corresponding reduction in the supply voltage to achieve power saving. The relation between power and frequency is captured by the following formula [Rabaey et al. 2002]:

$$P \propto FV^2. \tag{1}$$

Here F shows the operating frequency and V shows the supply voltage. By intelligently reducing the frequency, the voltage at which the circuit needs to be operated for stable operation can also be reduced, leading to power saving. However, since the reduction in frequency also harms the performance, the scaling of voltage/frequency needs to be carefully performed. Also note that in some of the works discussed later, the frequency scaling is actually applied to CPU; however, we still include these works since the power saving is achieved in the entire system and power management of CPU is done while taking into account the properties of GPU.

Nam et al. [2007] propose a low-power GPU for hand-held devices. The proposed GPU uses logarithmic arithmetic to optimize area and power consumption. The use of logarithmic arithmetic leads to some computation error; however, due to the small screen of the hand-held devices, the error can be tolerated. They divide the chip into three power domains, viz. vertex shader, rendering engine, and RISC processor, and DVFS is individually applied to each of the three domains. The power management unit decides the supply voltage and frequency of each domain based on its workload for saving power while maintaining the desired performance level.

Ren et al. [2012] discuss an approach for saving system energy in a heterogeneous CPU-GPU computing system. They suggest that, instead of using a single GPU with each CPU, using multiple GPUs with each CPU enables achieving speedup in execution time and improving the usage of the CPU, which improves the energy efficiency of the system. Further, since during the execution of the CUDA kernel the host CPU remains in the polling loop without doing useful work, the frequency of the CPU can be reduced for saving energy while always ensuring that CPU frequency is greater than the PCIe bus between the CPU and GPU. Since the range of high-performance CPU frequencies is generally larger than that of the PCIe bus, CPU frequency can be scaled without affecting GPU performance. They demonstrate their approach by parallelizing 3D finite element mesh refinement on GPU.

Anzt et al. [2011] propose techniques for reducing energy consumption in CPU-GPU heterogeneous systems for executing iterative linear solvers. They propose using DVFS for saving energy in the CPU while it stays in busy-wait waiting for the GPU to complete computations. Since during this time the CPU performs no useful work, use of DVFS gives large energy savings with little performance loss. Further, since the conjugate gradient iterative linear solver consumes nearly the same time in different iterations, by noting this duration once, the CPU can be transitioned to the sleep state for this duration in further calls to the kernel, which leads to further energy savings. They also remark that this technique is useful when the calls to kernels consume a sufficiently large amount of time.

Jiao et al. [2010] study the performance and power consumption of GPUs for three computationally diverse applications for varying processor and memory frequencies. Specifically, they study dense matrix multiplication (compute intensive), dense matrix transpose (memory intensive), and fast Fourier transform (hybrid). They have observed that the power consumption of GPUs is primarily dependent on the ratio of global memory transactions to computation instructions and the rate of issuing instructions. These two metrics decide whether an application is memory intensive or computation

intensive, respectively. Based on these characteristics, the frequency of GPU cores and memory is adjusted to save energy.

Lin et al. [2011] propose use of software prefetching and dynamic voltage scaling to save GPU energy. Software prefetching is a technique that aims to improve performance by overlapping the computing and memory access latencies. It works by inserting prefetch instructions into the program so that data is fetched into registers or caches well before time, and processor stall on memory access instructions is avoided. Since prefetching increases the number of instructions, it also increases the power consumption, and hence, it must be balanced with suitable performance enhancement. Their technique analyzes the program to insert prefetching instructions and then iteratively uses DVFS to find a suitable frequency such that performance constraint is met while saving the largest possible amount of energy.

4.3. CPU-GPU Work Division to Improve Energy Efficiency

Researchers have shown that different ratios of work division between CPUs and GPUs may lead to different performance and energy efficiency levels [Ma et al. 2012; Luk et al. 2009]. Based on this observation, several techniques have been proposed that dynamically choose between CPU and GPU as a platform of execution of a kernel based on the expected energy efficiency on those platforms.

Ma et al. [2012] propose an energy management framework for GPU-CPU heterogeneous architectures. Their technique works in two steps. In the first step, the workload is divided between the CPU and GPU based on the workload characteristics in a manner that both sides may complete their tasks approximately at the same time. As an example, the task shared by the CPU and GPU may be 15% and 85%, respectively. This step ensures load balancing, which also avoids energy waste due to idling. In the second step, the frequency of GPU cores and memory are adjusted, along with the frequency and voltage of the CPU, to achieve the largest possible energy savings with minimal performance degradation.

Luk et al. [2009] propose an automatic technique for mapping computations of processing elements on a CPU-GPU heterogeneous system. Compared to other approaches that require the programmer to manually perform the computations to processor mapping, their technique uses a runtime adaptation to automatically perform the mapping. Their technique provides an API (application programming interface) for writing parallelizable programs. Through the API, the computations are explicitly expressed, and hence, the compiler is not required to extract parallelism from the serial code. While OpenMP can exploit parallelism only on the CPU, their technique can exploit parallelism on both the CPU and the GPU. Since the optimal mapping changes with different applications, hardware/software configurations, and input problem sizes, the adaptive mapping outperforms hand-tuned mapping in both performance and energy efficiency.

Liu et al. [2012] discuss a technique for finding power-efficient mappings of time-critical applications onto CPU/GPU heterogeneous systems. Their technique works in two steps. In the first step, their technique maps the application to either the CPU or the GPU, such that their deadlines are met and execution time is minimized. In the second step, DVFS techniques are applied to both the CPU and GPU to save energy. The mapping of applications can be done in both an offline and online manner. To keep the performance high and avoid resource idling, their technique also aims to achieve load balancing. Moreover, their technique utilizes the fact that typically average-case execution times are less than their worst-case execution time, and hence, early completion provides a slack, which can be exploited using DVFS to save a large amount of energy.

Takizawa et al. [2008] propose SPRAT (stream programming with runtime auto-tuning), a runtime environment for dynamically selecting a CPU or GPU with a view

to improve the energy efficiency. They introduce a performance model that takes into account the relative execution time and energy consumption on the CPU and GPU and the data transfer time between the CPU and GPU. This model is especially suited for applications that require frequent data transfers between the CPU and GPU. Based on the runtime behavior, SPRAT can dynamically select the computing platform (CPU or GPU) for executing a kernel such that system energy is minimized.

Rofouei et al. [2008] experimentally evaluate the power and energy cost of GPU operations and compare it with that of CPU for the convolution problem. They find the relation between execution time and energy consumption and show that that GPU is more energy efficient when it provides application performance improvement above a certain threshold. Based on this, the decision about running the application on a CPU or GPU can be taken. Ren and Suda [2009] discuss a scenario where the performance benefit provided by using two GPUs (instead of one) offsets the power consumption overhead of the extra GPU and leads to power saving. They demonstrate their approach for multiplication of large matrices.

Liu et al. [2011] develop an energy-saving algorithm for large-scale GPU cluster systems based on the waterfall model. In their cluster, each node may have many CPU–GPU pairs. Their method divides the energy consumption of the overall system into three different levels based on different energy-saving strategies deployed. Their method formulates the energy-saving problem as an optimization task, where the energy consumption needs to be minimized while meeting task deadlines. Their technique transitions the node in one among three states, namely, busy (all CPUs and GPUs inside a node are executing task), spare (at least one CPU–GPU pair is free), and sleep (all CPU–GPU pairs are free). At the time of reduced workload, the node in the sleep state is powered off to save energy, and at the time of additional workload, a node is woken up. Also, their technique selects an appropriate task from the set of available tasks and schedules it on an optimal CPU–GPU pair such that the execution time of the task is minimized. Further, the voltage of the CPU is adaptively scaled to save energy while meeting the task deadline. Finally, they also utilize the β-migration policy, where a small fraction (β) of the GPU's share of the task is migrated to the CPU in the same CPU–GPU pair for achieving load balancing.

4.4. Saving Energy in GPU Components

Several techniques make architecture-level changes to GPUs to optimize the energy spent in individual components of the GPU. These techniques utilize the specific usage pattern of GPU components to make runtime adaptations for saving energy.

Gebhart et al. [2011] present a technique for saving energy in the core datapath of GPU. Since GPUs employ a large number of threads, storing the register context of these threads requires a large amount of on-chip storage. Also, the thread scheduler in the GPU needs to select a thread to execute from a large number of threads. For these reasons, accessing large register files and scheduling among a large number of threads consumes a substantial amount of energy. To address this, Gebhart et al. present two improvements. First, a small storage structure is added to register files that acts like a cache and captures the working set of registers to reduce energy consumption. Second, the threads are logically divided into two types, namely, active threads (which are currently issuing instructions or waiting on relatively short latency operations) and pending threads (which are waiting on long memory latencies). Thus, in any cycle, the scheduler needs to consider only the active threads that are much smaller in number. This leads to significant energy savings.

Wang et al. [2012] propose a technique for saving static energy in both L1 and L2 caches. They propose putting the L1 cache (which is private to each core) in

state-preserving[1] low-leakage mode when there are no threads that are ready to be scheduled. Further, the L2 cache is transitioned to low-leakage mode when there is no memory request. They also discuss the microarchitectural optimizations that ensure that the latency of detecting cache inactivity and transitioning a cache to low power and back to normal power are completely hidden.

Lashgar et al. [2013] propose the use of a filter cache to save energy in GPUs by reducing accesses to the instruction cache. Their technique is based on "interwarp instruction temporal locality," which means that during short execution intervals, a small number of static instructions account for a significant portion of dynamic instructions fetched and decoded within the same stream multiprocessor. Thus, the probability that a recently fetched instruction will be fetched again is high. They propose using a small filter cache to cache these instructions and reduce the number of accesses to the instruction cache, which improves the energy efficiency of the fetch engine. Filter cache has been used in CPUs also; however, in GPUs, the instruction temporal locality is even higher. This is because GPUs interleave thousands of threads per core, which are grouped in warps. The warp scheduler continuously issues instructions from different warps, which fills the warp, thus fetching the same instruction for all warps during short intervals.

A unified local memory design for GPUs is presented by Gebhart et al. [2012]. The existing GPUs use rigid partition sizes of registers, cache, and scratchpad; however, different GPU workloads have different requirements of registers, caches, and scratchpad (also called shared memory). Based on the characterization study of different workloads, they observe that different kernels and applications have different requirements of cache, shared memory, and so forth. To address this issue, they propose a unified memory architecture that aggregates these three types of storage and allows for a flexible allocation on a per-kernel basis. Before the launch of each kernel, the system reconfigures the memory banks to change the partitioning of the memory. By effectively using the local storage, their design reduces the accesses to main memory. They have shown that using their approach broadens the range of applications that can be efficiently executed on GPUs and also provides improved performance and energy efficiency.

To filter a large fraction of memory requests that are serviced by the first-level cache or scratchpad memory, Sankaranarayanan et al. [2013] propose adding small sized caches (termed as tinyCaches) between each lane in a streaming multiprocessor (SM) and the L1 data cache, which is shared by all the lanes in an SM. Further, using some unique features of the CUDA/OpenCL programming model, these tinyCaches avoid the need for complex coherence schemes, and thus, they can be implemented with low cost. They have shown that their design leads to improvement in the energy efficiency of the GPU.

Rhu et al. [2013] propose a technique for finding the right data-fetch granularity for improving the performance and energy efficiency of GPUs. They observe that only a few applications use all four 32B sectors of the 128B cache block, which leads to overfetching of data from the memory. To address this issue, their technique first decides the appropriate granularity (coarse grain or fine grain) of data fetch. Based on this, a hardware predictor adaptively adjusts the memory access granularity without programmer or runtime system intervention. Thus, their approach enables adaptively adjusting the memory access granularity depending on the spatial locality present in the application.

[1]State preserving refers to the low-power state where the contents stored in the block are not lost. This is in contrast with the state-destroying low-power state, where the block contents are lost in the low-power mode [Mittal et al. 2013].

In a CPU-GPU heterogeneous computing system (HCS) with shared last-level cache (LLC), interference between CPU and GPU threads can lead to degradation in performance and energy efficiency. This is especially critical since the GPU has a much larger number of threads than the CPU, and hence, the large number of accesses from the GPU are likely to evict data brought in cache by the CPU threads. Some authors propose techniques to address this issue [Lee and Kim 2012; Mekkat et al. 2013]. Lee and Kim [2012] propose a thread-level parallelism (TLP)-aware cache management policy for such systems. Due to the presence of deep multithreading, a cache policy does not directly affect the performance in GPUs. Hence, to estimate the effect of cache behavior on GPU performance, they propose a core-sampling approach, which leverages the fact that most GPU applications show symmetric behavior across the running cores. Based on this, core sampling applies a different policy (e.g., a cache replacement policy) to each core and periodically collects samples to see how the policies work. A large difference in performance of these cores indicates that GPU performance is affected by the cache policy and vice versa. Using this, the best cache management policy can be chosen. Further, to alleviate the interference, they introduce a cache block lifetime normalization approach, which ensures that statistics collected for each application are normalized by the access rate of each application. Using this, along with a cache partitioning mechanism, cache is partitioned between the CPU and GPU, such that cache is allocated to the GPU only if it benefits from the cache.

Mekkat et al. [2013] propose a technique that leverages the GPU's ability to tolerate memory access latency to throttle GPU LLC accesses to provide cache space to latency-sensitive CPU applications. Based on the observation that the TLP available in an application is a good indicator of the cache sensitivity of an application, their technique allows GPU memory traffic to selectively bypass the shared LLC if GPU cores exhibit sufficient TLP to tolerate memory access latency or when the GPU is not sensitive to LLC performance. A large number of wavefronts that are ready to be scheduled indicates a higher amount of TLP. Using core sampling, they apply two different bypassing thresholds to two different cores to find the impact of bypassing on GPU performance. Also, using cache set sampling, the effect of GPU bypassing on CPU performance is estimated. Using these, the rate of GPU bypassing is periodically adjusted to improve performance and save energy.

4.5. Dynamic Resource Allocation-Based Techniques

It is well known that there exists large intra-application and interapplication variation in the resource requirements of different applications. In fact, several real-world applications rarely utilize all the computational capabilities of the GPU. Thus, significant amount of energy savings can be achieved by dynamically adapting the components that exhibit low utilization levels.

Hong and Kim [2010] propose an integrated power and performance prediction system to save energy in GPUs. For a given GPU kernel, their method predicts both performance and power and then uses these predictions to choose the optimal number of cores that can lead to the highest performance per watt value. Based on this, only the desired number of cores can be activated, while the remaining cores can be turned off using power gating. Note that power gating is a circuit-level scheme to remove leakage by shutting off the supply voltage to unused circuits.

Wang et al. [2011] propose power-gating strategies for saving energy in GPUs. In graphics applications, different scenes have different complexities (e.g., number of objects), and hence, the amount of computing resources that are required to provide a satisfactory visual perception varies across different frames. By predicting the required shader resources for providing the desired frame rate, the extra shader resources can be turned off using power gating. To avoid the overhead of power gating, their technique

ensures that the idle period of the unused circuits is long enough to compensate the switching overhead.

Wang and Ranganathan [2011] present an offline profiling-based technique to estimate the appropriate number of GPU cores for a given application to save energy. Their technique uses the profile of PTX (parallel thread execution) codes generated during compilation of the application to decide the number of cores to be used for achieving the highest energy efficiency. During the actual run, in place of using the programmer-specified number of cores, only the desired number of cores can be activated to save energy.

Among the commercial products, AMD uses PowerPlay technology [AMD PowerPlay 2013] for dynamic power management. It dynamically transitions the GPU between low, medium, and high states, based on the load on the GPU. For example, while a graphics application is running, the demand on the GPU is high, and hence, it runs in a high-power state. Conversely, while typing emails, the load on the GPU is minimal, and hence, it runs in a low-power state. The power saving also reduces system temperatures and fan noise. Similarly, NVIDIA uses PowerMizer technology for dynamic power management [NVIDIA PowerMizer 2013].

4.6. Application-Specific and Programming-Level Techniques

It has been observed that source-code-level transformations and application-specific optimizations can significantly improve the resource utilization, performance, and energy efficiency of GPUs. Thus, by performing manually or automatically optimizing GPU implementation and addressing performance bottlenecks, large energy savings can be obtained.

Wang et al. [2010] propose a method for saving energy in GPUs using kernel fusion. Kernel fusion combines the computation of two kernels into a single thread. Thus, it leads to balancing the demand of hardware resources, which improves utilization of resources and thus improves the energy efficiency. The authors formulate the task of kernel fusion as a dynamic programming problem, which can be solved using conventional tools.

Alonso et al. [2012] propose a technique to save energy in task-parallel execution of dense linear algebra operations (viz. Cholesky and LU factorization) by intelligently replacing the busy waits with a power-friendly blocking state. Execution of these tasks involves a CPU thread issuing the kernel (for execution on the GPU) and then waiting for the next ready task in a busy wait polling loop. This leads to wastage of energy. To avoid this, their technique blocks the CPU thread on a synchronization primitive when waiting for the GPU to finish work, thus leading to saving of energy.

Ghosh et al. [2012] study the energy efficiency of HPC application kernels (viz. matrix-matrix multiplication, FFT, pseudo-random number generation, and 3D finite difference) on multi-GPU and multicore CPU platforms. The kernel implementations are taken from standard libraries. They observe that while the absolute "power" consumption (in watts) of multi-GPU is larger than that of the multicore CPU, the "energy efficiency" (in gigaflops per watt) of GPUs is far superior to that of CPUs. They observe that for GPUs, the number of global memory accesses and operations per unit time have a significant influence on the power consumption. Also, a large computation to the communication ratio per device is important for hiding data transfer latency and realizing energy efficiency in GPUs.

Yang et al. [2012] evaluate several open-source GPU projects and suggest ways to change the program code to improve GPU usage, performance, and energy efficiency. These projects are taken from a wide range of disciplines, such as atmosphere science, computational physics, machine learning, bioinformatics, and mathematics. They identify the common code patterns that lead to inefficient hardware use. For example,

adjustment of thread-block dimension can improve the way global memory data are accessed and reused in either shared memory or hardware caches. Further, choice of global memory data types and use of texture and constant memory has a significant effect on achieved bandwidth. Also, by optimizing the program for specific GPUs (e.g., AMD GPU or NVIDIA GPU), the hardware-specific features can be exploited to obtain higher performance and energy efficiency.

5. FUTURE RESEARCH TRENDS AND CONCLUSION

We believe that in the near future, the challenges of GPU power consumption will need to be simultaneously addressed at different levels at the same time. At the chip design level, researchers are aiming to develop energy-efficient throughput cores and memory design to exploit instruction-level, data-level, and fine-grained task-level parallelism. At the architecture level, CPUs and GPUs need to be efficiently integrated on the same chip with a unified memory architecture [Foley et al. 2012; Yuffe et al. 2011]. This will address the memory bandwidth bottleneck and also avoid the replicated chip infrastructure and the need for managing separate memory spaces. At the programming level, per-application tuning is inevitable to achieve a fine balance between demands of the application and the resources of the GPU. Finally, at the system level, policies for intelligent scheduling and work division between CPU and GPU are required, so that their individual competencies are integrated and they complement each other.

The 3D die stacking technology holds the promise of mitigating the memory bandwidth bottleneck in GPUs, as it enables use of shorter, high-bandwidth, and power-efficient global interconnect and provides a denser form factor. 3D stacking also enables integration of heterogeneous technologies, which allows use of nonvolatile memory (NVM), such as phase change RAM (PCM) and spin transfer torque RAM (STT-RAM), in the design of GPU memory [Mittal 2013]. NVMs consume negligible leakage power and provide higher density than SRAM and DRAM; however, their write latency and energy are significantly higher than those of SRAM and DRAM. It is expected that leveraging the benefits of 3D stacking and NVM would be a major step in improving the energy efficiency of GPUs and it would require novel solutions at the device, architecture, and system level.

As GPUs become deployed in large-scale data centers and supercomputers, the challenges of power management are expected to grow. For such large systems, power management needs to be done at the level of both intranode and internode. These nodes may be remotely situated and may have different configurations (e.g., CPU, GPU, FPGA, etc., or different interconnection). Managing power consumption of such systems while taking into account load balancing, temperature reduction, and performance target will be an interesting research problem for designers. On the other side of the spectrum, in battery-operated devices such as smartphones, where the need for processing visually compelling graphics within a small power budget increases with each new generation, the requirement for aggressive energy optimization will pose novel challenges for computer architects.

Virtualization technology enables multiple computing environments to be consolidated in a single physical machine and thus increases resource utilization efficiency and reduces total cost of ownership (TCO). Specifically, in cloud computing, virtualization is a key enabling technology since flexible resource provisioning is essential for unpredictable user demands. Very recently, GPUs have been used in cloud computing and virtual machine (VM) platforms [NVIDIA 2014; Jo et al. 2013; Shi et al. 2012; Amazon EC2 2013]. By adding or removing GPUs in each VM in an on-demand manner, VMs in the same physical host can use the GPUs in a time-sharing manner [Jo et al. 2013], which also leads to significant reduction in idle power of GPUs. We believe

that much research still needs to be done to leverage virtualization for minimizing power and TCO of the GPU computing infrastructure.

In this article, we surveyed several methods aimed at analyzing and improving the energy efficiency of GPUs. We underscored the need for power management in GPUs and identified important trends that are worthy of future investigation. Further, we presented a classification of different research works to highlight the underlying similarities and differences between them. We believe that this survey will provide the researchers valuable insights into the state of the art in GPU power management techniques and motivate them to create breakthrough inventions for designing green GPUs of the future exascale era.

REFERENCES

Mohammad Abdel-Majeed and Murali Annavaram. 2013. Warped register file: A power efficient register file for GPGPUs. In *HPCA*. 412–423.

Mohammad Abdel-Majeed, Daniel Wong, and Murali Annavaram. 2013. Warped gates: Gating aware scheduling and power gating for GPGPUs. In *International Symposium on Microarchitecture (MICRO'13)*. 111–122.

Yuki Abe, Hiroshi Sasaki, Martin Peres, Koji Inoue, Kazuaki Murakami, and Shinpei Kato. 2012. Power and performance analysis of GPU-accelerated systems. In *USENIX Conference on Power-Aware Computing and Systems (HotPower'12)*.

P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Orti. 2012. Reducing energy consumption of dense linear algebra operations on hybrid CPU-GPU platforms. In *International Symposium on Parallel and Distributed Processing with Applications (ISPA'12)*. 56–62.

Amazon EC2. 2013. Hompage. Retrieved from http://aws.amazon.com/hpc-applications/.

AMD PowerPlay. 2013. Homepage. Retrieved from http://www.amd.com/us/products/technologies/ati-power-play.

D. Anderson, J. Dykes, and E. Riedel. 2003. More than an interface-SCSI vs. ATA. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*. 245–257.

Hartwig Anzt, Vincent Heuveline, José I Aliaga, Maribel Castillo, Juan C. Fernandez, Rafael Mayo, and Enrique S. Quintana-Orti. 2011. Analysis and optimization of power consumption in the iterative solution of sparse linear systems on multi-core and many-core platforms. In *International Green Computing Conference and Workshops (IGCC'11)*. IEEE, 1–6.

Hartwig Anzt, Björn Rocker, and Vincent Heuveline. 2010. Energy efficiency of mixed precision iterative refinement methods using hybrid hardware platforms. *Computer Science-Research and Development* 25, 3 (2010), 141–148.

José-María Arnau, Joan-Manuel Parcerisa, and Polychronis Xekalakis. 2012. Boosting mobile GPU performance with a decoupled access/execute fragment processor. In *International Symposium on Computer Architecture (ISCA'12)*. 84–93.

Zachary K. Baker, Maya B. Gokhale, and Justin L. Tripp. 2007. Matched filter computation on FPGA, Cell and GPU. In *IEEE Symposium on Field-Programmable Custom Computing Machines*. 207–218.

Khaled Benkrid, Ali Akoglu, Cheng Ling, Yang Song, Ying Liu, and Xiang Tian. 2012. High performance biological pairwise sequence alignment: FPGA versus GPU versus Cell BE versus GPP. *International Journal of Reconfigurable Computing* 2012, Article 7.

Brahim Betkaoui, David B. Thomas, and Wayne Luk. 2010. Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing. In *International Conference on Field-Programmable Technology (FPT'10)*. IEEE, 94–101.

Ricardo Bianchini and Ram Rajamony. 2004. Power and energy management for server systems. *Computer* 37, 11 (2004), 68–76.

Matthias Birk, Matthias Balzer, Nicole Ruiter, and Juergen Becker. 2012. Comparison of processing performance and architectural efficiency metrics for FPGAs and GPUs in 3D ultrasound computer tomography. In *International Conference on Reconfigurable Computing and FPGAs (ReConFig'12)*. 1–7.

Andre R. Brodtkorb, Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik, and Olaf O. Storaasli. 2010. State-of-the-art in heterogeneous computing. *Scientific Programming* 18, 1 (2010), 1–33.

Guilherme Calandrini, Alfredo Gardel, Pedro Revenga, and Jose Luis L'zaro. 2012. GPU acceleration on embedded devices. A power consumption approach. In *14th IEEE International Conference on High Performance Computing and Communication and 9th International Conference on Embedded Software and Systems (HPCC-ICESS'12)*. 1806–1812.

J. M. Cebrian, G. D. Guerrero, and J. M. Garcia. 2012. Energy efficiency analysis of GPUs. In *International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW'12)*. 1014–1022.

Aparna Chandramowlishwaran, Samuel Williams, Leonid Oliker, Ilya Lashuk, George Biros, and Richard Vuduc. 2010. Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures. In *International Symposium on Parallel & Distributed Processing (IPDPS'10)*. 1–12.

Chia-Ming Chang, Shao-Yi Chien, You-Ming Tsao, Chih-Hao Sun, Ka-Hang Lok, and Yu-Jung Cheng. 2008. Energy-saving techniques for low-power graphics processing unit. In *International SoC Design Conference*, Vol. 1. IEEE, 242–245.

Thomas C. P. Chau, Xinyu Niu, Alison Eele, Wayne Luk, Peter Y. K. Cheung, and Jan Maciejowski. 2013. Heterogeneous reconfigurable system for adaptive particle filters in real-time applications. In *International Symposium on Applied Reconfigurable Computing (ARC'13)*. 1–12.

Jianmin Chen, Bin Li, Ying Zhang, Lu Peng, and Jih-kwon Peir. 2011. Tree structured analysis on GPU power study. In *International Conference on Computer Design (ICCD'11)*. IEEE, 57–64.

Gary Chun Tak Chow, Anson Hong Tak Tse, Qiwei Jin, Wayne Luk, Philip H. W. Leong, and David B. Thomas. 2012. A mixed precision monte carlo methodology for reconfigurable accelerator systems. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 57–66.

Slo-Li Chu, Chih-Chieh Hsiao, and Chiu-Cheng Hsieh. 2011. An energy-efficient unified register file for mobile GPUs. In *International Conference on Embedded and Ubiquitous Computing (EUC'11)*. 166–173.

Eric S. Chung, Peter A. Milder, James C. Hoe, and Ken Mai. 2010. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? In *MICRO*. 225–236.

Sylvain Collange, David Defour, and Arnaud Tisserand. 2009. Power consumption of GPUs from a software perspective. In *International Conference on Computational Science (ICCS'09)*. 914–923.

Jason Cong, Muhuan Huang, and Yi Zou. 2011. 3D recursive Gaussian IIR on GPU and FPGAs: A case for accelerating bandwidth-bounded applications. In *9th Symposium on Application Specific Processors (SASP'11)*. IEEE, 70–73.

Jason Cong and Yi Zou. 2009. FPGA-based hardware acceleration of lithographic aerial image simulation. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 2, 3 (2009), 17:1–17:29.

Mayank Daga, Ashwin M. Aji, and Wu-chun Feng. 2011. On the efficacy of a fused cpu+ gpu processor (or apu) for parallel computing. In *Symposium on Application Accelerators in High-Performance Computing (SAAHPC'11)*. IEEE, 141–149.

Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. 2010. The scalable heterogeneous computing (SHOC) benchmark suite. In *3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 63–74.

Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. 2008. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *ACM/IEEE Conference on Supercomputing*. 1–12.

Christian De Schryver, Ivan Shcherbakov, Frank Kienle, Norbert Wehn, Henning Marxen, Anton Kostiuk, and Ralf Korn. 2011. An energy efficient FPGA accelerator for monte carlo option pricing with the heston model. In *2011 International Conference on Reconfigurable Computing and FPGAs (ReConFig'11)*. IEEE, 468–474.

Sebastian Dreßler and Thomas Steinke. 2012. Energy consumption of CUDA kernels with varying thread topology. *Computer Science-Research and Development* (2012), 1–9.

Bo Duan, Wendi Wang, Xingjian Li, Chunming Zhang, Peiheng Zhang, and Ninghui Sun. 2011. Floating-point mixed-radix FFT core generation for FPGA and comparison with GPU and CPU. In *International Conference on Field-Programmable Technology (FPT'11)*. IEEE, 1–6.

Jeremy Enos, Craig Steffen, Joshi Fullop, Michael Showerman, Guochun Shi, Kenneth Esler, Volodymyr Kindratenko, John E Stone, and James C. Phillips. 2010. Quantifying the impact of GPUs on performance and energy efficiency in HPC clusters. In *International Green Computing Conference*. 317–324.

Hadi Esmaeilzadeh, Emily Blem, Renée St Amant, Karthikeyan Sankaralingam, and Doug Burger. 2013. Power challenges may end the multicore era. *Communications of the ACM* 56, 2 (2013), 93–102.

Wu-Chun Feng. 2003. Making a case for efficient supercomputing. *Queue* 1, 7 (2003).

Denis Foley, Pankaj Bansal, Don Cherepacha, Robert Wasmuth, Aswin Gunasekar, Srinivasa Gutta, and Ajay Naini. 2012. A low-power integrated x86–64 and graphics processor for mobile computing devices. *IEEE Journal of Solild-State Circuits* 47, 1 (2012), 220–231.

Jeremy Fowers, Greg Brown, John Wernsing, and Greg Stitt. 2013. A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors. *ACM Transactions on Architecture and Code Optimization (TACO)* 9, 4 (2013), 25.

Mark Gebhart, Daniel R. Johnson, David Tarjan, Stephen W. Keckler, William J. Dally, Erik Lindholm, and Kevin Skadron. 2011. Energy-efficient mechanisms for managing thread context in throughput processors. *ACM SIGARCH Computer Architecture News* 39, 3 (2011), 235–246.

Mark Gebhart, Stephen W. Keckler, Brucek Khailany, Ronny Krashinsky, and William J. Dally. 2012. Unifying primary cache, scratch, and register file memories in a throughput processor. In *Annual IEEE/ACM International Symposium on Microarchitecture*. 96–106.

GeForce GTX 590. 2013. Specifications. Retrieved from http://www.geforce.com/hardware/desktop-GPUs/geforce-gtx-590/specifications.

Sayan Ghosh, Sunita Chandrasekaran, and Barbara Chapman. 2012. Energy analysis of parallel scientific kernels on multiple GPUs. In *Symposium on Application Accelerators in High Performance Computing (SAAHPC'12)*. IEEE, 54–63.

Syed Zohaib Gilani, Nam Sung Kim, and Michael J. Schulte. 2012. Power-efficient computing for compute-intensive GPGPU applications. In *21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. 445–446.

Syed Zohaib Gilani, Nam Sung Kim, and Michael J. Schulte. 2013. Exploiting GPU peak-power and performance tradeoffs through reduced effective pipeline latency. In *International Symposium on Microarchitecture (MICRO'13)*. 74–85.

Dominik Goddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick McCormick, Hilmar Wobker, Christian Becker, and Stefan Turek. 2008. Using GPUs to improve multigrid solver performance on a cluster. *International Journal of Computational Science and Engineering* 4, 1 (2008), 36–55.

D. Gohringer, M. Birk, Y. Dasse-Tiyo, N. Ruiter, M. Hubner, and J. Becker. 2011. Reconfigurable MPSoC versus GPU: Performance, power and energy evaluation. In *IEEE International Conference on Industrial Informatics (INDIN'11)*. 848–853.

Tsuyoshi Hamada, Khaled Benkrid, Keigo Nitadori, and Makoto Taiji. 2009. A comparative study on ASIC, FPGAs, GPUs and general purpose processors in the O (N^2) gravitational n-body simulation. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS'09)*. 447–452.

Tomoaki Hamano, Toshio Endo, and Satoshi Matsuoka. 2009. Power-aware dynamic task scheduling for heterogeneous accelerated clusters. In *International Symposium on Parallel & Distributed Processing (IPDPS'09)*. IEEE, 1–8.

Daniel Hefenbrock, Jason Oberg, Nhat Thanh, Ryan Kastner, and Scott B. Baden. 2010. Accelerating viola-jones face detection to FPGA-level using GPUs. In *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'10)*. 11–18.

Sunpyo Hong and Hyesoon Kim. 2010. An integrated GPU power and performance model. *ACM SIGARCH Computer Architecture News* 38, 3 (2010), 280–289.

Chih-Chieh Hsiao, Slo-Li Chu, and Chen-Yu Chen. 2013. Energy-Aware hybrid precision selection framework for mobile GPUs. *Computers & Graphics* 37, 5 (2013), 431–444.

Song Huang, Shucai Xiao, and W. Feng. 2009. On the energy efficiency of graphics processing units for scientific computing. In *International Symposium on Parallel & Distributed Processing (IPDPS'09)*. 1–8.

Hongpeng Huo, Chongchong Sheng, Xinming Hu, and Baifeng Wu. 2012. An energy efficient task scheduling scheme for heterogeneous GPU-enhanced clusters. In *International Conference on Systems and Informatics (ICSAI'12)*. IEEE, 623–627.

Hanaa M. Hussain, Khaled Benkrid, Ahmet T. Erdogan, and Huseyin Seker. 2011. Highly parameterized k-means clustering on FPGAs: Comparative results with GPPs and GPUs. In *International Conference on Reconfigurable Computing and FPGAs (ReConFig'11)*. 475–480.

Intel Core i7. 2013. Specifications. Retrieved from http://ark.intel.com/products/65525/Intel-Core-i7-3770T-Processor-8M-Cache-up-to-3_70-GHz.

Intel Xeon E7. 2013. Specifications. Retrieved from http://ark.intel.com/products/53580/Intel-Xeon-Processor-E7-8870-30M-Cache-2_40-GHz-6_40-GTs-Intel-QPI.

Yaser Jararweh and Salim Hariri. 2012. Power and performance management of GPUs based cluster. *International Journal of Cloud Applications and Computing (IJCAC'12)* 2, 4 (2012), 16–31.

Y. Jiao, H. Lin, P. Balaji, and W. Feng. 2010. Power and performance characterization of computational kernels on the GPU. In *Int'l Conference on Green Computing and Communications (GreenCom) & Int'l Conference on Cyber, Physical and Social Computing (CPSCom'10)*. 221–228.

Naifeng Jing, Yao Shen, Yao Lu, Shrikanth Ganapathy, Zhigang Mao, Minyi Guo, Ramon Canal, and Xiaoyao Liang. 2013. An energy-efficient and scalable eDRAM-based register file architecture for GPGPU. In *International Symposium on Computer Architecture (ISCA'13)*. 344–355.

Heeseung Jo, Jinkyu Jeong, Myoungho Lee, and Dong Hoon Choi. 2013. Exploiting GPUs in virtual machine for biocloud. *BioMed Research International* 2013 (2013), 11. doi:10.1155/2013/939460.

Kiran Kasichayanula, Dan Terpstra, Piotr Luszczek, Stan Tomov, Shirley Moore, and Gregory Peterson. 2012. Power aware computing on GPUs. In *Symposium on Application Accelerators in High-Performance Computing*. 64–73.

Stephen W. Keckler, William J. Dally, Brucek Khailany, Michael Garland, and David Glasco. 2011. GPUs and the future of parallel computing. *IEEE Micro* 31, 5 (2011), 7–17.

Vincent Keller and Ralf Gruber. 2010. One joule per GFlop for BLAS2 Now!. In *American Institute of Physics Conference Series*, Vol. 1281. 1321–1324.

Srinidhi Kestur, John D. Davis, and Oliver Williams. 2010. BLAS comparison on FPGA, CPU and GPU. In *IEEE Annual Symposium on VLSI* (2010), 288–293.

Holger Lange, Florian Stock, Andreas Koch, and Dietmar Hildenbrand. 2009. Acceleration and energy efficiency of a geometric algebra computation using reconfigurable computers and GPUs. In *IEEE Symposium on Field Programmable Custom Computing Machines*. 255–258.

Ahmad Lashgar, Amirali Baniasadi, and Ahmad Khonsari. 2013. Inter-Warp instruction temporal locality in deep-multithreaded GPUs. In *Architecture of Computing Systems (ARCS'13)*. 134–146.

Jaekyu Lee and Hyesoon Kim. 2012. TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture. In *18th International Symposium on High Performance Computer Architecture (HPCA'12)*. IEEE, 1–12.

Jeabin Lee, Byeong-Gyu Nam, and Hoi-Jun Yoo. 2007. Dynamic voltage and frequency scaling (DVFS) scheme for multi-domains power management. In *IEEE Asian Solid-State Circuits Conference*. 360–363.

Jungseob Lee, Vijay Sathisha, Michael Schulte, Katherine Compton, and Nam Sung Kim. 2011. Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling. In *International Conference on Parallel Architectures and Compilation Techniques (PACT'11)*. IEEE, 111–120.

JunKyu Lee, Junqing Sun, Gregory D. Peterson, Robert J. Harrison, and Robert J. Hinde. 2010. Power-aware performance of mixed precision linear solvers for FPGAs and GPGPUs. In *Symposium on Application Accelerators in High Performance Computing*.

Victor W Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, et al. 2010. Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU. In *ACM SIGARCH Computer Architecture News* 38, 451–460.

Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling energy optimizations in GPGPUs. In *International Symposium on Computer Architecture (ISCA'13)*. 487–498.

Dong Li, Surendra Byna, and Srimat Chakradhar. 2011. Energy-Aware workload consolidation on GPU. In *International Conference on Parallel Processing Workshops (ICPPW'11)*. IEEE, 389–398.

Yisong Lin, Tao Tang, and Guibin Wang. 2011. Power optimization for GPU programs based on software prefetching. In *International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom'11)*. 1339–1346.

Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. 2012. Power-efficient time-sensitive mapping in heterogeneous systems. In *International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. ACM, 23–32.

Wenjie Liu, Zhihui Du, Yu Xiao, David A. Bader, and Chen Xu. 2011. A waterfall model to achieve energy efficient tasks mapping for large scale GPU clusters. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW'11)*. 82–92.

Daniel Llamocca, Cesar Carranza, and Marios Pattichis. 2011. Separable FIR filtering in FPGA and GPU implementations: Energy, Performance, and Accuracy considerations. In *International Conference on Field Programmable Logic and Applications (FPL'11)*. IEEE, 363–368.

Miguel López-Portugués, Jesús A López-Fernández, Alberto Rodríguez-Campa, and José Ranilla. 2011. A GPGPU solution of the FMM near interactions for acoustic scattering problems. *Journal of Supercomputing* 58, 3 (2011), 283–291.

Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. 2009. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *International Symposium on Microarchitecture (MICRO'09)*. 45–55.

Cheng Luo and Reiji Suda. 2011. A performance and energy consumption analytical model for GPU. In *International Conference on Dependable, Autonomic and Secure Computing (DASC'11)*. IEEE, 658–665.

Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. GreenGPU: A holistic approach to energy efficiency in GPU-CPU heterogeneous architectures. In *International Conference on Parallel Processing (ICPP'12)*. 48–57.

Arian Maghazeh, Unmesh D. Bordoloi, Petru Eles, and Zebo Peng. 2013. *General Purpose Computing on Low-Power Embedded GPUs: Has It Come of Age?* Technical Report. Linkping University, Software and Systems.

Simon McIntosh-Smith, Terry Wilson, Amaurys Ávila Ibarra, Jonathan Crisp, and Richard B. Sessions. 2012. Benchmarking energy efficiency, power costs and carbon emissions on heterogeneous systems. *Computer Journal* 55, 2 (2012), 192–205.

Vineeth Mekkat, Anup Holey, Pen-Chung Yew, and Antonia Zhai. 2013. Managing shared last-level cache in a heterogeneous multicore processor. In *22nd International Conference on Parallel Architectures and Compilation Techniques (PACT'13)*. 225–234.

Rich Miller. 2013. Exascale Computing. Retrieved from http://www.datacenterknowledge.com/archives/2010/12/10/exascale-computing-gigawatts-of-power/.

Sparsh Mittal. 2012. A survey of architectural techniques for DRAM power management. *International Journal of High Performance Systems Architecture* 4, 2 (2012), 110–119.

Sparsh Mittal. 2013. *Energy Saving Techniques for Phase Change Memory (PCM)*. Technical Report. Iowa State University.

Sparsh Mittal, Yanan Cao, and Zhao Zhang. 2013. MASTER: A multicore cache energy saving technique using dynamic cache reconfiguration. *IEEE Transactions on VLSI Systems* 22, 8 (2013), 1653–1665.

Shuai Mu, Chenxi Wang, Ming Liu, Dongdong Li, Maohua Zhu, Xiaoliang Chen, Xiang Xie, and Yangdong Deng. 2011. Evaluating the potential of graphics processors for high performance embedded computing. In *2011 Design, Automation Test in Europe Conference Exhibition (DATE'11)*. 1–6.

Hitoshi Nagasaka, Naoya Maruyama, Akira Nukada, Toshio Endo, and Satoshi Matsuoka. 2010. Statistical power modeling of GPU kernels using performance counters. In *International Conference on Green Computing*. 115–122.

Byeong-Gyu Nam, Jeabin Lee, Kwanho Kim, Seung Jin Lee, and Hoi-Jun Yoo. 2007. A low-power handheld GPU using logarithmic arithmetic and triple DVFS power domains. In *ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, Vol. 4. 73–80.

NVIDIA. 2014. Dedicated GPU Technology for Virtual Desktops. Retrieved from http://www.nvidia.com/object/dedicated-gpus.html.

NVIDIA PowerMizer. 2013. NVIDIA PowerMizer Technology-Extend Battery Life on Notebook PCs. Retrieved from http://www.nvidia.com/object/ feature_powermizer.html.

Edson Luiz Padoin, Laércio Lima Pilla, Francieli Zanon Boito, Rodrigo Virote Kassick, Pedro Velho, and Philippe O. A. Navaux. 2012. Evaluating application performance and energy consumption on hybrid CPU+ GPU architecture. *Cluster Computing* (2012), 1–15.

Chanmin Park, Hyunhee Kim, and Jihong Kim. 2006. A low-power implementation of 3D graphics system for embedded mobile systems. In *Workshop on Embedded Systems for Real Time Multimedia*. 53–58.

C. D. Patel, C. E. Bash, R. Sharma, M. Beitelmal, and R. Friedrich. 2003. Smart cooling of data centers. In *Pacific RIM/ASME International Electronics Packaging Technical Conference and Exhibition (IPACK'03)*.

Indrani Paul, Vignesh Ravi, Srilatha Manne, Manish Arora, and Sudhakar Yalamanchili. 2013. Coordinated energy management in heterogeneous processors. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'13)*. 59.

Karl Pauwels, Matteo Tomasi, Javier Diaz Alonso, Eduardo Ros, and Marc M. Van Hulle. 2012. A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features. *IEEE Transactions on Computers* 61, 7 (2012), 999–1012.

Ardavan Pedram, Robert A. van de Geijn, and Andreas Gerstlauer. 2012. Co-Design tradeoffs for high-performance, low-power linear algebra architectures. *IEEE Transactions on Computers* 61, 12 (2012), 1724–1736.

Jeff Pool, Anselmo Lastra, and Montek Singh. 2008. Energy-precision tradeoffs in mobile graphics processing units. In *International Conference on Computer Design (ICCD'08)*. IEEE, 60–67.

Jeff Pool, Anselmo Lastra, and Montek Singh. 2010. An energy model for graphics processing units. In *IEEE International Conference on Computer Design (ICCD'10)*. 409–416.

Jeff Pool, Anselmo Lastra, and Montek Singh. 2011. Precision selection for energy-efficient pixel shaders. In *ACM SIGGRAPH Symposium on High Performance Graphics*. 159–168.

Jan M. Rabaey, Anantha P. Chandrakasan, and Borivoje Nikolic. 2002. *Digital Integrated Circuits*. Vol. 2. Prentice Hall, Englewood Cliffs, NJ.

RADEON. 2013. http://www.amd.com/US/PRODUCTS/DESKTOP/GRAPHICS/ATI-RADEON-HD-5000/HD-5970/Pages/ati-radeon-hd-5970-overview.aspx#2. (2013).

Karthik Ramani, Ali Ibrahim, and Dan Shimizu. 2007. PowerRed: A flexible modeling framework for power efficiency exploration in GPUs. In *Workshop on General Purpose Processing on GPUs (GPGPU'07)*.

Da Qi Ren. 2011. Algorithm level power efficiency optimization for CPU–GPU processing element in data intensive SIMD/SPMD computing. *Journal of Parallel and Distributed Computing* 71, 2 (2011), 245–253.

Da Qi Ren, E. Bracken, S. Polstyanko, N. Lambert, R. Suda, and D. D. Giannacopulos. 2012. Power aware parallel 3-D finite element mesh refinement performance modeling and analysis with CUDA/MPI on GPU and multi-core architecture. *IEEE Transactions on Magnetics* 48, 2 (2012), 335–338.

Da Qi Ren and R. Suda. 2009. Power efficient large matrices multiplication by load scheduling on multi-core and GPU platform with CUDA. In *International Conference on Computational Science and Engineering*, Vol. 1. IEEE, 424–429.

Minsoo Rhu, Michael Sullivan, Jingwen Leng, and Mattan Erez. 2013. A locality-aware memory hierarchy for energy-efficient GPU architectures. In *International Symposium on Microarchitecture (MICRO'13)*. 86–98.

Justin Richardson, Steven Fingulin, Diwakar Raghunathan, Chris Massie, Alan George, and Herman Lam. 2010. Comparative analysis of HPC and accelerator devices: Computation, memory, I/O, and power. In *International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA'10)*. IEEE, 1–10.

Mahsan Rofouei, Thanos Stathopoulos, Sebi Ryffel, William Kaiser, and Majid Sarrafzadeh. 2008. Energy-aware high performance computing with graphic processing units. In *Workshop on Power Aware Computing and System*.

Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2013. Divergence-Aware warp scheduling. In *46th IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. 99–110.

A. Sankaranarayanan, E. K. Ardestani, J. L. Briz, and J. Renau. 2013. An energy efficient GPGPU memory hierarchy with tiny incoherent caches. In *IEEE International Symposium on Low Power Electronics and Design (ISLPED'13)*. 9–14.

T. R. W. Scogland, Heshan Lin, and Wu-chun Feng. 2010. A first look at integrated GPUs for green high-performance computing. *Computer Science-Research and Development* 25, 3 (2010), 125–134.

A. Sethia, G. Dasika, M. Samadi, and S. Mahlke. 2013. APOGEE: Adaptive prefetching on GPUs for energy efficiency. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 73–82.

Jeremy W. Sheaffer, Kevin Skadron, and David P. Luebke. 2005a. Fine-grained graphics architectural simulation with Qsilver. In *ACM SIGGRAPH 2005 Posters*. ACM, 118.

Jeremy W. Sheaffer, Kevin Skadron, and David P. Luebke. 2005b. Studying thermal management for graphics-processor architectures. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'05)*. IEEE, 54–65.

Lin Shi, Hao Chen, Jianhua Sun, and Kenli Li. 2012. vCUDA: GPU-accelerated high-performance computing in virtual machines. *IEEE Transactions on Computers.* 61, 6 (2012), 804–816.

Larry Smarr. 2010. Project greenlight: Optimizing cyber-infrastructure for a carbon-constrained world. *Computer* 43, 1 (2010), 22–27.

Kyle L Spafford, Jeremy S. Meredith, Seyong Lee, Dong Li, Philip C. Roth, and Jeffrey S. Vetter. 2012. The tradeoffs of fused memory hierarchies in heterogeneous computing architectures. In *9th Conference on Computing Frontiers*. ACM, 103–112.

L. Stolz, H. Endt, M. Vaaraniemi, D. Zehe, and W. Stechele. 2010. Energy consumption of graphic processing units with respect to automotive use-cases. In *International Conference on Energy Aware Computing (ICEAC)*. IEEE, 1–4.

Lars Struyf, Stijn De Beugher, Dong Hoon Van Uytsel, Frans Kanters, and Toon Goedemé. 2014. The battle of the giants: A case study of GPU vs FPGA optimisation for real-time image processing. In *4th International Conference on Pervasive and Embedded Computing and Communication systems (PECCS'14)*.

Reiji Suda and Da Qi Ren. 2009. Accurate measurements and precise modeling of power dissipation of CUDA kernels toward power optimized high performance CPU-GPU computing. In *International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, 432–438.

Hiroyuki Takizawa, Katsuto Sato, and Hiroaki Kobayashi. 2008. SPRAT: Runtime processor selection for energy-aware computing. In *International Conference on Cluster Computing*. IEEE, 386–393.

David Barrie Thomas, Lee Howes, and Wayne Luk. 2009. A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation. In *International Symposium on Field Programmable Gate Arrays*. 63–72.

Constantin Timm, Andrej Gelenberg, F. Weichert, and P. Marwedel. 2010. *Reducing the Energy Consumption of Embedded Systems by Integrating General Purpose GPUs*. Technische Universität Dortmunde, Department of Computer Science.

Top500. 2013. Top500 List—November 2013. Retrieved from http://www.top500.org/lists/2013/11/.

Takuro Udagawa and Masakazu Sekijima. 2011. The power efficiency of GPUs in multi nodes environment with molecular dynamics. In *International Conference on Parallel Processing Workshops (ICPPW'11)*. IEEE, 399–405.

Brian Van Essen, Chris Macaraeg, Maya Gokhale, and Ryan Prenger. 2012. Accelerating a random forest classifier: Multi-core, GP-GPU, or FPGA? In *International Symposium on Field-Programmable Custom Computing Machines (FCCM'12)*. IEEE, 232–239.

Stéphane Vialle, Sylvain Contassot-Vivier, Thomas Jost, et al. 2011. Optimizing computing and energy performances in heterogeneous clusters of CPUs and GPUs. *Handbook of Energy-Aware and Green Computing.* Chapman Hall/CRC.

Hasitha Muthumala Waidyasooriya, Yasuhiro Takei, Masanori Hariyama, and Michitaka Kameyama. 2012. Low-Power heterogeneous platform for high performance computing and its application to 2D-FDTD computation. In *International Conference on Reconfigurable Systems and Algorithms.*

Bin Wang, Bo Wu, Dong Li, Xipeng Shen, Weikuan Yu, Yizheng Jiao, and Jeffrey S. Vetter. 2013. Exploring hybrid memory for GPU energy efficiency through software-hardware co-design. In *International Conference on Parallel Architectures and Compilation Techniques (PACT'13)*. 93–102.

Guibin Wang, YiSong Lin, and Wei Yi. 2010. Kernel fusion: An effective method for better power efficiency on multithreaded GPU. In *2010 IEEE/ACM International Conference on Green Computing and Communications (GreenCom) and International Conference on Cyber, Physical and Social Computing (CPSCom)*. IEEE, 344–350.

Haifeng Wang and Qingkui Chen. 2012. An energy consumption model for GPU computing at instruction level. *International Journal of Advancements in Computing Technology* (2012), 192–200.

Po-Han Wang, Chia-Lin Yang, Yen-Ming Chen, and Yu-Jung Cheng. 2011. Power gating strategies on GPUs. *ACM Transactions on Architecture and Code Optimization* 8, 3 (2011), 13:1–13:25.

Wendi Wang, Bo Duan, Wen Tang, Chunming Zhang, Guangming Tang, Peiheng Zhang, and Ninghui Sun. 2012. A coarse-grained stream architecture for cryo-electron microscopy images 3D reconstruction. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 143–152.

Yue Wang and N. Ranganathan. 2011. An instruction-level energy estimation and optimization methodology for GPU. In *International Conference on Computer and Information Technology (CIT'11)*. 621–628.

Yue Wang, S. Roy, and N. Ranganathan. 2012. Run-time power-gating in caches of GPUs for leakage energy savings. In *Design, Automation Test in Europe Conference Exhibition (DATE'12)*. 300–303.

Zhuowei Wang, Xianbin Xu, Naixue Xiong, Laurence T. Yang, and Wuqing Zhao. 2010. Analysis of parallel algorithms for energy conservation with GPU. In *International Conference on Green Computing and Communications (GreenCom) & International Conference on Cyber, Physical and Social Computing (CPSCom'10)*. IEEE, 155–162.

Jason Williams, Alan D. George, Justin Richardson, Kunal Gosrani, and Siddarth Suresh. 2008. Computational density of fixed and reconfigurable multi-core devices for application acceleration. In *Proceedings of Reconfigurable Systems Summer Institute.*

Henry Wong, M.-M. Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. 2010. Demystifying GPU microarchitecture through microbenchmarking. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'10)*. 235–246.

Yi Yang, Ping Xiang, Mike Mantor, and Huiyang Zhou. 2012. Fixing performance bugs: An empirical study of open-source GPGPU programs. In *International Conference on Parallel Processing*. 329–339.

Wing-kei S. Yu, Ruirui Huang, Sarah Q. Xu, Sung-En Wang, Edwin Kan, and G. Edward Suh. 2011. SRAM-DRAM hybrid memory with applications to efficient register files in fine-grained multi-threading. In *ACM SIGARCH Computer Architecture News*, Vol. 39. 247–258.

Marcelo Yuffe, Ernest Knoll, Moty Mehalel, Joseph Shor, and Tsvika Kurts. 2011. A fully integrated multi-CPU, GPU and memory controller 32nm processor. In *International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'11)*. 264–266.

Pooya Zandevakili, Ming Hu, and Zhaohui Qin. 2012. GPUmotif: An ultra-fast and energy-efficient motif analysis program using graphics processing units. *PloS One* 7, 5 (2012), e36865.

Changyou Zhang, Kun Huang, Xiang Cui, and Yifeng Chen. 2012. Energy-aware GPU programming at source-code levels. *Tsinghua Science and Technology* 17, 3 (2012), 278–286.

Ying Zhang, Yue Hu, Bin Li, and Lu Peng. 2011. Performance and power analysis of ATI GPU: A statistical approach. In *International Conference on Networking, Architecture and Storage (NAS'11)*. IEEE, 149–158.

Ying Zhang, Lu Peng, Bin Li, Jih-Kwon Peir, and Jianmin Chen. 2011. Architecture comparisons between Nvidia and ATI GPUs: Computation parallelism and data communications. In *IEEE International Symposium on Workload Characterization (IISWC'11)*. 205–215.

Jishen Zhao, Guangyu Sun, Gabriel H. Loh, and Yuan Xie. 2012. Energy-efficient GPU design with reconfigurable in-package graphics memory. In *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12)*. 403–408.

Dan Zou, Yong Dou, and Fei Xia. 2012. Optimization schemes and performance evaluation of Smith–Waterman algorithm on CPU, GPU and FPGA. *Concurrency and Computation: Practice and Experience* 24, 14, 1625–1644.

Received May 2013; revised February 2014; accepted June 2014