Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods

ROBERT A. BRIDGES, NEENA IMAM, and TIFFANY M. MINTZ,
Oak Ridge National Laboratory

Modern graphics processing units (GPUs) have complex architectures that admit exceptional performance and energy efficiency for high-throughput applications. Although GPUs consume large amounts of power, their use for high-throughput applications facilitate state-of-the-art energy efficiency and performance. Consequently, continued development relies on understanding their power consumption. This work is a survey of GPU power modeling and profiling methods with increased detail on noteworthy efforts. As direct measurement of GPU power is necessary for model evaluation and parameter initiation, internal and external power sensors are discussed. Hardware counters, which are low-level tallies of hardware events, share strong correlation to power use and performance. Statistical correlation between power and performance counters has yielded worthwhile GPU power models, yet the complexity inherent to GPU architectures presents new hurdles for power modeling. Developments and challenges of counter-based GPU power modeling are discussed. Often building on the counter-based models, research efforts for GPU power simulation, which make power predictions from input code and hardware knowledge, provide opportunities for optimization in programming or architectural design. Noteworthy strides in power simulations for GPUs are included along with their performance or functional simulator counterparts when appropriate. Last, possible directions for future research are discussed.

Categories and Subject Descriptors: A.1 [**General Literature**]: Introductory and Survey; I.3.1 [**Hardware Architecture**]: Graphics Processors; H.3.4 [**Systems and Software**]: Performance Evaluation (Efficiency and Effectiveness)

General Terms: Experimentation, Performance

Additional Key Words and Phrases: GPU, GPGPU, power profile, power model, simulation

ACM Reference Format:
Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. 2016. Understanding GPU power: A survey of profiling, modeling, and simulation methods. ACM Comput. Surv. 49, 3, Article 41 (September 2016), 27 pages.
DOI: http://dx.doi.org/10.1145/2962131

1. INTRODUCTION

The state of the art in extreme scale computing has maintained exponential growth in performance for two decades, with peak system performance almost breaching 100

This work was supported by the United States Department of Defense and used resources of the Computational Research and Development Programs at Oak Ridge National Laboratory. This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan http://energy.gov/downloads/doe-public-access-plan.
Authors' addresses: R. A. Bridges, N. Imam, and T. M. Mintz, 1 Bethel Valley Road, Oak Ridge National Laboratory, Oak Ridge, TN 37831; emails: {bridgesra, imamn, mintztm}@ornl.gov.

DOI: http://dx.doi.org/10.1145/2962131

petaflops (floating point operations per second) [Anthony 2013, 2014; Top500 2016]. Specifically, China's Sunway TaihuLight at 93.0 petaflops tops the June 2016 Top500, which lists the world's most powerful supercomputers, followed by China's Tianhe-2 (MilkyWay-2) and next U.S. Department of Energy's Titan exhibiting performance of at 33.9 and 17.6 petaflops, respectively[1] [Top500 2016]. As high-performance computing (HPC) systems continue to grow, ballooning budgets and temperature management issues caused by the exorbitant power consumption of such large systems are proving a formidable obstacle to continuing this trend; for example, TaihuLight comsumes 15.4 megawatts with an efficiency (operations/watt) of 6.0 gigaflops/watt, while Tianhe 2 and Titan exhibit about a third of the efficiency (and performance). Currently, the world's most power-efficient supercomputer is Shoubu at RIKEN in Japan, exhibiting 7.03 gigaflops/watt (energy efficiency) and 0.413 petaflops (performance). Using these statistics and linearly scaling to exaflops puts exascale power requirements at over 142 megawatts [Green500 2015]; hence, the U.S. Department of Energy cites a ∼100-fold increase in power efficiency necessary to reach exascale performance [Ashby et al. 2010; Tolentino and Cameron 2012]. Modern computing resources are often *over-provisioned*, where temperature or power constraints dissuade running all processors at full load, and questions of resource optimization (e.g, how to allocate cores, frequency, bandwidth) are necessary for continued development [Rountree et al. 2012]. Emerging research is addressing the need for understanding power and performance profiles for such systems and applications.

Noteworthy strides in power-efficient hardware have come from graphics processing units (GPUs), especially for parallelizable scientific applications. Computational or general-purpose GPUs have exhibited greater performance and energy efficiency than CPUs for high-throughput, high-latency applications, for example, simulations involving partial differential equations or convolutions used in image processing [Chung et al. 2010; Huang et al. 2009; Rofouei et al. 2008]. Consequently, GPUs are now used in conjunction with CPUs that handle the more serial operations. Because of their energy efficiency, GPU accelerated HPC systems are increasingly prevalent in both the Top500 and Green500 lists, which rank the world's fastest and most energy efficient supercomputers, respectively [Green500 2015; Top500 2016]. As of November 2015, 72 of the Top500 computers employ GPU accelerators. Furthermore Top500 systems relying on coprocessor or GPU acceleration have increased from 1% (5 systems) in June 2009 to 18% (89 systems) in November 2015. Looking forward, NVIDIA Volta GPUs are in the building plans for Summit, a 150- to 300-petaflop computer that is expected to be deployed at Oak Ridge National Laboratory by 2018 [Smith 2014]. Although GPU accelerated systems are being quickly adopted by the HPC industry, power consumption remains a pressing issue [Hruska 2014]. Furthermore, as GPUs continue to develop, understanding their power and performance profiles on real applications is increasingly difficult [Song et al. 2013]. Analogously to the large-scale distributed systems, the GPU itself is often over-provisioned, and research for how to use GPUs both effectively and efficiently is in progress [Jia et al. 2015]. As GPUs are an increasing prevalent and power-hungry asset in HPC envrionments, understanding and predicting the energy cost of GPU sub-processes is a necessity in this power-constrained computing era.

Originally designed to assist the CPU with graphics, GPUs were historically a hardware pipeline composed of many different single-function units. As technology developed, the varied components increased programmability and parallelism. Unlike CPUs, which are designed for orchestrating a wide variety of quickly changing serial tasks, modern compute GPUs excel in performance and power efficiency when given highly repetitive tasks. Put simply, CPUs are optimized for programmability while compute

[1]Performance reported on the LINPACK benchmark. Details can be found here http://www.top500.org/project/linpack/.

GPUs are fashioned for high throughput; unsurprisingly, the physical architectures differ greatly. Instead of a few CPU cores, modern GPUs contain from dozens to a few thousand small processors, called streaming processors (SPs); for example, both the NVIDIA K40 GPU and the AMD FirePro S9710 GPUs sport nearly 2,900 cores. Depending on the GPU specific architecture, from 8 to 64 SPs are organized into a streaming multiprocessor (SM) along with a few special function units (SFUs), which handle the more complex math operations, such as square roots, but SMs do not have a branch unit, as is necessary in CPUs. With respect to memory hardware, each SM includes a multi-thread instruction fetch and issue unit, a read-only (texture) cache, and read/write shared (L1) memory. GPUs also contain an L2 memory, which is accessible by all SMs. For more detailed history and GPU architecture information, see Hong et al. [2009], Luebke and Humphreys [2007], McClanahan [2010], Robson [2008], or Singer [2013].

The GPU's complex internal memory hierarchy and thousands of processors lead to natural challenges for power modeling. For example, "How to attain accurate power prediction in the presence of asymmetric loads (where there is a disparity in the portions of the cores per SM used)?" and "How does compute-bound (memory access happens during and waits on computation) versus memory-bound applications (computation happens during and waits on memory access) affect the GPUs power requirements?" are a few questions beginning to surface in the research. Additional research is probing how to apply the power modeling research for an optimal balance of power and performance (e.g., see Jia et al. [2015]) and pioneering flexible profiling tools for monitoring GPU processes, (e.g., see Stephenson et al. [2015]). Only in the most recent architectures are a large number of the GPU hardware events observable, and how to harness these for accurate understanding of power is thinly addressed. This problem arises in the modeling research discussed in Section 4. Furthermore, the position GPUs have established in high-performance and scientific computing communities has increased the urgency of understanding the power cost of GPU usage.

This survey presents an overview of research involving methods to monitor, model, and simulate GPU power consumption. As most of the academic literature involving GPU developments focus at least experimentally on discrete (in particular, NVIDIA) GPUs, a similar focus is reflected in our discussions, although, when possible, more general GPU architectures, such as Intel's integrated GPUs, are discussed. Little mobile device GPU research is included in this survey. In order to facilitate an understanding of important products and methods, selected works are discussed in significant depth and organized into a narrative. We reference interested readers to a related work by Mittal and Vetter [2014], which presents a survey on GPU energy efficiency. To the best of our knowledge, this is the first survey on the various methods for obtaining and predicting GPU power profiles. Following efforts to understand GPU power consumption, our survey is divided into two main categories, (1) direct methods, which involve hardware products and research for monitoring system and component power (Section 3), and (2) indirect methods, which covers modeling (Section 4) and simulation (Section 5). This is a natural dichotomy into two subjects with necessary overlap—for all models rely on ground-truth power data, and understanding power when it is immeasurable requires accurate models. We also provide a brief discussion of GPU programming software and available benchmarks and software development kits (SDKs) (Section 2) with a goal of pointing interested readers to resources rather than providing comprehensive descriptions. Finally, we conclude with a few directions for future research.

2. PREREQUISITES: GPU SOFTWARE

This section gives an overview of the software used in the GPU power modeling literature. As software is not our focus, we do not give a comprehensive review of these

Table I. GPU Scientific Application Benchmark Suites

Suite	Source	Reference
Rodinia	http://lava.cs.virginia.edu/Rodinia	Che et al. [2009]
SHOC	https://github.com/vetter/shoc	Danalis et al. [2010]
Parboil	http://impact.crhc.illinois.edu/parboil/parboil.aspx	Strtton et al. [2012]
Modern GPU	https://nvlabs.github.io/moderngpu/	N/A
ISPASS	https://github.com/gpgpu-sim/ispass2009-benchmarks	Bakhoda et al. [2009]
Lonestar	http://iss.ices.utexas.edu/?p=projects/galois/lonestar	Kulkarni et al. [2009]
NUPAR	https://code.google.com/p/nupar-bench/	Ukidave et al. [2015]
MAGMA	http://icl.cs.utk.edu/projectsfiles/magma/doxygen/index.html	Dongarra, Tomov [2014]
CompuBench	https://cmopubench.com/benchmark.jsp	N/A

areas but rather strive to familiarize the reader with what is used in the literature and provide examples for those interested in taking up GPU research.

Popular software for programming for general purpose GPUs include CUDA, OpenCL, and OpenACC. NVIDIA's Compute Unified Device Architecture, known as CUDA, is the platform plus programming model built to harness parallel computing opportunities both internal to NVIDIA GPUs and in a distributed computing environment containing NVIDIA GPUs [NVIDIA 2014a]. We note that CUDA terminology includes an important concept of a *warp* used often in the literature. A *warp* is the execution of 32 parallel threads with single instruction multiple data (SIMD). CUDA programmers allocate threads to blocks, and blocks to grids, but the GPU allocates each block's threads into warps and assigns them to the SMs. Maintained by Khronos, OpenCL is an open-source, multi-platform programming model for parallel computing supporting GPUs by a variety of vendors including AMD and NVIDIA. In particular, Intel's GPUs and the Xeon Phi coprocessors now support OpenCL to expose parallel compute abilities [Cray 2014].[2] Additionally, there is OpenACC, a Cray developed programming model allowing specification of areas of C, C++, or Fortran code to be directed from CPUs to GPUs, co-processors, or for use with Accelerated Processing Units or APUs [Khronos 2014].

In order to acquire data for training and evaluating models, almost all works in the literature employ GPU benchmarks, programs that are designed to repeatedly execute a few kernels that either stress the GPU in a particular isolated manner or are representative activities of the GPU. Table I enumerates the GPU benchmarks suites used or developed in the literature. All of which are freely downloadable, and links are provided. In addition to the benchmark suites, NVIDIA provides a CUDA software development kit (SDK),[3] an OpenGL SDK,[4] and CUDA accelerated high-performance Linpack code.[5] Also referenced in the literature are Merge benchmarks discussed by Linderman et al. [2008], although to our knowledge, these are not publicly available.

3. POWER MEASUREMENT

Directly measuring power via internal or external hardware sensors is generally considered the most accurate source of power consumption information. Furthermore, such measurements are necessary for initially learning parameters for indirect methods as well as for quantifiable evaluation of a given power model's accuracy. To obtain energy estimates, direct methods use periodic meter readings to estimate power used during a time interval. An estimate of total energy over a time span is calculated via an integral of the power-versus-time graph over the appropriate intervals. Below we

[2]https://software.intel.com/en-us/articles/opencl-drivers.
[3]https://developer.nvidia.com/cuda-code-samples.
[4]https://www.opengl.org/sdk/.
[5]https://developer.nvidia.com/rdp/cuda-registered-developer-program.

Table II. Direct Power Monitoring Hardware

Type	AC External	DC External	Internal
Configuration	Between Wall and Device	Rails between PSU and Component	N/A
Granularity	Whole Device (Cluster, Node)	Component	Component
Sampling Rate	10–100 Hz	100–5000 Hz	∼50 Hz
Example Products	Kill A Watt, Wattsup, Schneider Electric PM800	Analog Devices ADM1191, National Instruments 9205	NVIDIA Fermi & Tesla GPUs

give an overview of external power sensors used either in practice or referenced in the literature for power observations, with particular attention to solutions for distributed computing systems. Additionally, for convenient and inexpensive monitoring of GPU power, the internal sensor provided in the NVIDIA Fermi and Tesla architecture GPUs is discussed. Details of power monitoring hardware is organized in Table II.

3.1. External Power Meters

External power meters include (1) inline universal meters for measuring AC (alternating current) power of a device and (2) DC (direct current) meters often connected between the power supply unit (PSU) and a component under investigation. While useful for general applications and in the computational power profiling literature, inline AC devices give too-coarse readings for component-level (e.g., GPU-, CPU-, DRAM-level) profiling. More fine-grained information (in both sampling rate and component-level readings) is obtainable via DC meters. In addition, hardware and software, designed specifically for profiling power consumption of distributed computing systems, have emerged in the research and commercially available products.

Universal external power meters such as Kill A Watt , WattsUp , and ITWatchDogs' inline power meter , connect between a device and its wall power source and give measurements such as voltage (volts), power (watts), current (amps), and energy (kilowatt hours) with sampling rates of 10 to 100 Hz.[6] While relatively cheap and simple to use, these devices are generally not suitable for comprehensive power profiling, especially in HPC settings. As in Ge et al. [2010], inline AC devices have been used to obtain node-level power profiles for small clusters but are generally not designed for large distributed environments. We also note that Exatech[7] offers many power meters that can easily clamp onto wires and offer readings for both AC and DC power. In general, internal sensors are needed for accurate component-level power profiling.

To obtain finer-scaled power measurements, a variety of Analog-to-Digital Converters (ADCs) such as Analog Devices ADM1191 or National Instruments 9205 are available.[8] These devices provide power and current data acquisition hardware that give multiple input channels with sampling rates on the order of 100 to 5000 Hz. This increase in frequency is necessary for accurate measurements of component-level power [Burtscher et al. 2014]; for example, it is calculated that at this sampling rate, GPUs can issue billions of instructions per sample [Bedard et al. 2010]. As used in works of Bedard et al. [2010] and Ge et al. [2010], such units can be connected to rails leaving the PSU to profile components serviced by that rail. Hence, understanding the power profile of each component is possible. We note that the Zalman VPM1 VGA Power Consumption Meter[9] is an external DC meter designed for monitoring GPU power.

[6]http://www.prodigit.com, http://www.wattsupmeters.com, http://www.itwatchdogs.com/in-linepowermeter-p30.html.
[7]www.exatech.com.
[8]http://www.analog.com/en/power-management/power-monitors/adm1191/products/product.html, http://sine.ni.com/nips/cds/view/p/lang/en/nid/208800.
[9]http://www.zalman.com/.

Table III. Multilevel Power Measurement Solutions for Distributed Systems

Name	Collection	Maturity	References
PowerPack	Out of Band	Research	[Feng et al. 2005; Ge et al. 2010]
PowerInsight	In & Out of Band	Commercial Product	[Laros et al. 2013, 2014; DeBonis et al. 2014, 2013]
PowerMon (1,2)	In Band	Commercial Product	[Bedard et al. 2010; DeBonis et al. 2013]

If only node-level or coarser power is desired, then in-line devices designed for distributed computing environments are available, for example, by APC and allow programmatic collection of power data for each node in a cluster.[10] Titan, the 17.6-petaflop supercomputer at Oak Ridge National Laboratory, uses Schneider Electric pm-800 meters[11] for monitoring and collection of power data at the cabinet (multiple nodes) level. When component-level power observation is necessary, the ADCs discussed above are ideal for rigging individual nodes, but the physical configuration is cumbersome and generally impractical for use in a working distributed HPC environment. Below we discuss PowerPack, an initial research effort for multilevel power observation of a cluster, and the commercially available solutions PowerInsight and PowerMon. Table III outlines the available multilevel power monitoring solutions for computing clusters.

3.1.1. PowerPack. Research to obtain comprehensive power profiling of distributed systems has been undertaken by Feng et al. [2005] and extended by Ge et al. [2010]. Comprised of a variety of hardware sensors and a software package, the profiling system, called PowerPack, simultaneously monitors component, node, and system level power meters at runtime. Inline methods for AC power measurement of each node are acquired using WattsUp, PSU power readings are given by Advanced Configuration and Power Interface (ACPI) [acp 2014] (which access internal sensors), and readings on each rail leaving the PSU are also attained using a National Instruments ADC (see Section 3). By using the external and internal meters simultaneously, power loss due to the AC/DC conversion is attained, and redundancy in power data collection allows for validation. In the case of multiple components sharing power supply lines, component-level power is ascertained by using micro-benchmarks and adding/removing components to discover specific pins servicing power to each component. In addition to the hardware monitors, PowerPack includes software that not only programmatically collects the comprehensive power data but also correlates power readings to specific lines in the application code. A prototype implementation on an eight-node cluster is the topic of Ge et al. [2010]. PowerPack, the ninth node, collects power and performance profiles and is used to evaluate energy efficiency and performance results (1) between single-core processing versus multi-core processing and (2) of dynamic voltage and frequency scaling (DVFS), an energy optimization technique. A full-scale implementation of PowerPack has been planned for System G [2008], a 22.8-teraflop computer, dubbed "the largest power aware cluster" [Burtscher et al. 2014; Grove 2011].

3.1.2. PowerInsight. PowerInsight, as introduced by Laros et al. [2013], is a customized power profiling and measurement device designed for distributed HPC systems. Commercially available from Penguin Computing [pen 2014], PowerInsight uses a sampling rate on the order of 1KHz and offers both *in-band* (data are collected and analyzed on the same computer) and *out-of-band* (data are collected and analyzed on a different computer) monitoring. Power data are collected at the system, node, and sub-node levels by sampling rails between the PSU and motherboard, and accurate measurements

[10]http://www.apc.com/products/category.cfm?id=6.
[11]http://www.schneider-electric.com/.

at the system and node level are reported. Specifically, evaluation of PowerInsight readings against voltmeters and ammeters shows less than 0.3% error and the coefficient of variation among repeated experiments is under 3%.

The HPC Power API defines a set of interfaces to support each layer of the HPC software stack [Laros et al. 2014; DeBonis et al. 2014]. These interfaces allow uniform interactions between two actors in the software stack, where an actor can be a system layer, person, or entity. In the HPC Power API, systems are described as a hierarchy of objects that may be heterogeneous. The objects have a set of attributes that enable power measurement and management within the system. Power measurements and statistics can be obtained in real time or retrieved from a data store. We have a prototype of the HPC Power API implemented and tested on a Penguin PowerInsight System. On the PowerInsight system, the PowerInsight API (PIAPI) acted as a plugin to the more generic HPC Power API, with PIAPI managing communication and sampling rates from the sensors. An additional feature of PIAPI is an internal framework for emulating hardware features on the embedded device.

3.1.3. PowerMon and PowerMon2. Similarly to PowerInsight, PowerMon/PowerMon2 [2010] are external performance and power data collection devices designed for commodity systems by the Renaissance Computing Institute. Both devices are installed by inserting ADC monitors on rails between the motherboard and PSU. PowerMon2 extends PowerMon's capabilities from six collection channels to eight and from a maximum sampling rate of 50Hz to 3KHz, and both are limited to in-band collection. Debonis et al. [2013] conduct an accuracy comparison, testing PowerMon2 and PowerInsight side by side using a digital oscilloscope to produce ground-truth power readings. As expected, both are inaccurate during low current draws but are within 6% accuracy under reasonable loads. Ultimately, PowerMon2 exhibited slightly higher accuracy and less variance than PowerInsight.

3.2. Internal Power Sensors

In addition to using extra hardware for obtaining component-level power monitoring, built-in sensors are incorporated into many components, allowing users to directly query and monitor power data in situ. For example, the Advanced Configuration and Power Interface (ACPI) is an open-source interface for hardware monitoring and configuration [acp 2014]. In particular, as used in PowerPack (Section 3.1.1), ACPI can allow access to internal power sensors. Internal power sensors are emerging in commodity GPUs; in particular, NVIDIA Fermi and Tesla architecture GPUs are equipped with internal sensors allowing convenient access to power data via profiling software, with sampling rates around 50Hz. Accuracy of NVIDIA's internal sensors is a current area of research, for example, see Section 3.2.1 below, and NVIDIA documentation cites readings within a 5% window [NVIDIA 2014c]. Little documentation exists for these sensors, and exactly how they obtain power for the GPU board is unclear. Drawbacks of using built-in sensors are that they lack ubiquity, have relatively low sampling rates, and, as discussed below, exhibit idiosyncrasies possibly causing inaccurate readings. On the other hand, when available, built-in sensors give three distinct advantages as follows (1) they are simple to use, (2) they do not require any added expense or rigging of additional hardware, and (3) they allow component level profiling regardless of shared rails. Uses of internal sensors in the literature include PowerPack (Section 3.1.1), Song et al. [2013] (Section 4.2.3), and Burtscher et al. [2014], discussed next.

3.2.1. NVIDIA K20 GPU Power Profiling Using Internal Sensors. To the authors' knowledge, Burtscher et al. [2014] is the only work to examine power profiles of computational GPUs as given by the integrated sensor. Power data from the built-in sensor are acquired while running two benchmarking programs on three GPUs, namely, the NVIDIA

Tesla K20c, K20m, and K20x. The benchmarks chosen are both *n*-body simulations from the Lonestar GPU Suite (see Table I) [Kulkarni et al. 2009]. The first, NB, is a highly parallelizable program for calculating pairwise forces in the *n*-body problem, while the second, called BH, uses the Barnes-Hut algorithm to compute the forces and is much less parallelizable. The main discovery is that on the K20c and K20m, power readings "lag behind" the expected profile, similarly to the expected profile in the presence of a capacitor. Consequently, energy calculations obtained by integrating the power curve can vary wildly from what is believed to be the correct energy used. By correcting the raw data using a capacitor model, the power gives an expected, believable power profile. Ironically, the K20x power data experiments did not exhibit this capacitance-like behavior, and the observed profile and modeled profile were nearly identical (capacitance constant is approximately zero in this instance). In this work, no external power meter is used; rather, validation is inferred as the capacitance model gives expected, consistent results among all three processors. These findings suggest that an evaluation of this capacitance model on internal sensor data validated against a proven external power monitoring device is necessary. More generally, as the internal power sensors are increasingly prevalent and certainly the most convenient profiling tool, proper validation and understanding of their output is a necessity for trusting their readings. In all, this work shows that transforming the raw output of NVIDIA's internal sensors is necessary for accurate analysis.

4. COUNTER-BASED POWER MODELS

Obtaining and configuring external power acquisition hardware can be a costly option for profiling, especially on large-scale, distributed systems. Furthermore, as internal sensors are neither a standard feature nor consistent in accuracy across hardware, there is a need, especially in the HPC industry, for indirect power acquisition methods. This section discusses research efforts that propose and evaluate modeling and simulation techniques for GPU power and performance estimates.

4.1. Hardware Performance Counters

Modeling techniques estimate power and performance by correlating each with hardware events accessible through performance counters. Performance counters are tallies of hardware actions, (e.g., number of instructions, average time per operation, number of memory accesses) that give users access to low-level hardware activities. Metrics, which are statistics built from counters to explain hardware events, exhibit strong correlations to power consumption and performance. Thus, the quantification of these correlations can yield accurate power models.

On the other hand, some limitations inherent to counter-based models persist, namely, (1) the number and type of counters available are not uniform across hardware; hence, models dependent on architecture-specific counters do not admit immediate generalization. Necessary counters for accurate power profiling may not be available in some settings, for example, see the work of Nagasaka et al. [2010] where texture cache counters were unavailable and adversely affected their power model. (2) An upper bound on the number of counters simultaneously accessible is a limitation imposed by the number of hardware registers. Consequently, multiple runs of an application may be necessary to collect the performance counter data in which case live, runtime estimates are inhibited. (3) Last, a GPU-specific limitation is that some architectures only allow counters for a whole streaming multiprocessor (SM), usually consisting of 8, 16, or 32 cores. Hence, an application using an imbalanced number of cores per SM can result in inaccurate predictions. However, despite their limitations, performance counters are a primary source for gaining insight into hardware activities and, as such, yield strong correlations to power consumption, performance, and temperature.

For more information on hardware monitoring and performance counters, see Groeger [2005] and Sprunt [2002].

4.1.1. Accessing Performance Counters. There are a variety of interfaces for accessing and visualizing hardware counter data, and many are hardware dependent. Such software often gives counter data (1) on command, (2) in a streaming visualization for near real-time monitoring, and/or (3) allows recording of data at preset intervals (e.g., by a separate thread while running an application). For example, Performance Monitor, also known as System Monitor, is Windows hardware monitoring software that allows real-time programmatic collection of counter data and has a graphical user interface (GUI) for displaying graphs of performance counter data [Willhalm 2012]. Other options include IBM's Content Collector,[12] Intel's Performance Counter Monitor,[13] PerfMon2 (for Linux) [Jarp et al. 2008], and AMD's CodeAnalyst [Drongowski and Center 2008].

Perhaps the most comprehensive hardware monitoring software, the Performance API (PAPI), provides a platform- and OS-agnostic tool suite for monitoring performance counters and other hardware sensors[14] [Weaver et al. 2013]. Originally configured for investigating hardware counters in the diversity of CPUs on the market, PAPI has been extended to Component PAPI (PAPI-C). PAPI-C now offers access to thermal sensors, GPU counters, memory interface chips, network interface cards, and network switches to give comprehensive transparency of hardware activity. PAPI 5 also boasts extended support for energy monitoring in high-performance environments. A GPU specific PAPI component is in consideration for future development according to Terpstra et al. [2010]. Additionally, PAPI CUDA component [NVIDIA 2014] is a hardware measurement and observation technology for the NVIDIA GPUs.

Software for accessing GPUs' internal counters is also available, for instance, AMD's PerfApi, which supports their consumer-grade Radeon GPUs, and Mali GPU Shader Development Studio for ARM's Mali GPUs [AMD 2015; ARM 2011]. NVIDIA offers a suite of profiling tools providing a mélange of capabilities for accessing, visualizing, optimizing, monitoring, and profiling applications and GPU hardware, including their commodity GPUs. Compatible tools include NVIDIA Nsight, NVIDIA Visual Profiler, TAU Performance System, Vampir Trace, PAPI CUDA Component, NVIDIA CUDA Profiling Tools Interface (CUPTI)[15] and the NVIDIA Management Library [NVIDIA 2011, 2012]. Furthermore, programmatic collection of hardware counters, metrics, and profiling information is available with NVIDIA's `nvprof` and `nvvp` commands [NVIDIA 2015].

The rest of this section gives modeling and simulation efforts found in the literature that are informed by hardware counters and metrics.

4.2. Counter-Based GPU Power Models

Ample research on power modeling of traditional computing systems exists, with emphasis on CPU and memory power (e.g., see Bellosa [2000], Joseph and Martonosi [2001], Li and John [2003], and Singh et al. [2009]), but much less literature is focused on GPUs, which have much more computational parallelism and hierarchical memory. Below we present a progression of GPU power modeling research found in the literature. These works are summarized in Table IV.

4.2.1. Ma et al. [2009]—A GPU Power Profile Model. Pioneering work in GPU power modeling [Ma et al. 2009] compares the accuracy of GPU power predictions by two regression

[12] http://www-01.ibm.com/support/docview.wss?uid=swg27024515.
[13] http://technet.microsoft.com/en-us/library/cc749154.aspx.
[14] http://icl.cs.utk.edu/papi/index.html.
[15] https://developer.nvidia.com/performance-analysis-tools.

Table IV. Counter- and Event-Based GPU Power Models

Reference	Section	Input	Output	Model(s)
[Ma et al. 2009]	4.2.1	GPU Counters	Power Profile	Linear, **Support Vector**
[Nagasaka et al. 2010]	4.2.2	GPU Counters, Metrics	Kernel Ave. Power	Linear (various input sets tested)
[Song et al. 2013]	4.2.3	GPU Counters, Metrics	Kernel Ave. Power	Linear, **Neural Network**
[Chen et al. 2011]	4.2.5	GPGPU Sim Events	Kernel Energy	Linear, Linear Regression Tree, **Random Forest**

Note: Model(s) column gives regression techniques tested with most accurate model in bold font.

learning machines, Support Vector Regression (SVR) and Support Linear Regression (SLR). This model estimates the power used by the GPU over a small time window, hence the output is a step function giving the predicted power profile curve. To illustrate the models explicitly, power, p, is modeled as

$$p = \sum_{i=1}^{n} w_i a_i + w_0, \tag{1}$$

where a_i are counts of a given event and w_i weights to be learned during training. If $a^{(j)} = (a_1^{(j)}, \ldots, a_n^{(j)})$ denotes the counter observations for the jth test with corresponding observed power $p^{(j)}$, then linear regression seeks the weight vector $w = (w_0, \ldots, w_n)$ so $\sum_j (w \cdot (1, a^{(j)}) - p^{(j)})^2$ is minimal, whereas vector regression seeks the minimal length w given the constraint $\sum_j (w \cdot (1, a^{(j)}) - p^{(j)})^2 < \epsilon$. Mathematical details of the regression techniques are widely available, for example, see Smola and Schölkopf [2004].

To create training data, benchmarks that stress different GPU sub-units (e.g., texture units, vertex shaders, pixel shaders) are executed while a separate node collects GPU power and performance counter data. An external DC power meter (see Section 3.1) is used to monitor an NVIDIA GeForce 8800gt GPU's power, and NVIDIA PerfKit[16] gives access to five performance counters that quantify the percentage of use of GPU sub-units. Evaluation of the power models is performed on a held-out test set as well as on eight separate applications that used the GPU for graphics and general purposes. Specifically, for graphics benchmarks Nexuiz, an open-source game, and three of NVIDIA's OpenGL SDK applications are used, and for GPGPU benchmarks three NVIDIA CUDA SDK applications and GNN, a GPGPU neural network implementation, are used. Overall, the SVR model outperformed the linear model on all but one test, and both models struggled to predict spikes in power. Large variances in accuracy are observed. As mentioned in both Ma et al. [2009] and Nagasaka et al. [2010], acquisition of global memory access counts are necessary for accurate GPU power modeling but were not available with this architecture.

4.2.2. Nagasaka et al. [2010]—Linear Regression Model for GPU Kernel Average Power. Building on the Ma et al. contribution, Nagasaka et al. [2010] continue exploration of linear regression power models. Although, like most succeeding works, their model's output is the average power for a single kernel (not a power profile curve for a whole benchmark). By using the CUDA Profiler [2011], 13 counters are observed while benchmarks are run on an NVIDIA GeForce 285 GTX GPU, and an external DC power acquisition system gives GPU power data. Power and counter profiles for 49 kernel applications taken from the NVIDIA CUDA SDK suite and the Rodinia benchmark suite [2009] are created as training data. Seven different linear regression models, varying only by their input metrics, are tested and compared. Experimentation with using metrics

[16]https://developer.nvidia.com/nvidia-perfkit.

from the counters is performed; for example, six different memory instruction counters are summed to create a memory metric and used as input to a linear regression model along with other counters or metrics. As this architecture did not support texture cache counters, tests with and without kernels that use texture cache provide empirical evidence that this counter event is important for accurate power prediction of some kernels. To summarize results, the model using all available counters was most accurate; additionally, global memory access is identified as the single largest correlated metric to GPU power consumption. We note that because only 13 counters were available, using all of them is feasible, yet modern architectures provide much larger number of counters presenting modeling efforts with a feature selection problem.

Limitations of this implementation include the following: (1) As only 4 counters can be collected in a given run, each program must be run at least four times to obtain data from the 13 counters. (2) Another limitation is that counters are read per streaming multiprocessor. In the presence of an asymmetric kernel (e.g., all 32 streaming processors on the first multiprocessor are used, but only 17 of the second are used), the counters will have unknown correlations to actual power consumed. (3) Last, no counters to monitor texture reads are accessible from this GPU.

4.2.3. Neural Network Power Model of Song et al. [2013]. In order to omit the linear assumptions inherent in the previously discussed power models, Song et al. [2013] use a neural network to predict the average power of a kernel from performance metrics. The power model is then used in conjunction with a counter-based performance model to estimate energy consumption. In this section, we examine only the power model, with the performance and energy work following.

Following previous works, Song et al. select kernels that stress the GPU in different ways. Specifically, kernels are chosen from the NVIDIA CUDA SDK suite, the SHOC suite [2010], and from GEM, a free scientific application introduced by Gordon et al. [2008]. The NVIDIA Fermi C2075 GPU is used in this work, and this more modern architecture provides two main advantages over previous efforts. First, during runtime of the kernels, a separate thread collects power data using the NVIDIA internal power sensor via the NVIDIA Management Library [2012]; hence, power data are accessible programmatically with no extra hardware or configuration. Second, the Fermi architecture provides greater insight into the GPU than previous models. As greater than 60 counters are accessible, feature selection is necessary before application of a regression algorithm to prevent a biased model (or underdetermined set of equations in the linear case). Hence, correlation with power is computed for each counter collected, and the top 14 counters are selected. Similar counters are then summed to create 10 performance events used as inputs to a neural network; for example, three counters related to global memory store actions are summed to create a single input. In all, the neural network uses an input layer of size 10 and two hidden layers of size 4. Mathematical details of artificial neural networks and back-propagation training are available; for example, see Russell et al. [1995].

For evaluation, a direct comparison to the linear model of Nagasaka et al. [2010] is conducted; specifically, two linear regression models—the previous model and a natural extension of the previous model that also includes texture events as input—are trained and tested alongside the new neural network model. Perhaps unsurprisingly, the neural network model excels in accuracy on 19 of 20 kernels in the test set, exhibiting often a third of the error of the linear models. We note that neural network accuracy is highly dependent on many configuration settings, such as size of each layer, activation functions, and optimization options, which can be costly to test and configure. Nonetheless, this work clearly shows that nonlinear models when properly configured are much more accurate across the board.

Limitations of this approach are similar to previous efforts; namely, the counters are for a full streaming multiprocessor, so asymmetric kernels cannot be accurately profiled. Additionally, multiple runs of each kernel are necessary to acquire counter data as only four counters can be accessed simultaneously. To the authors' knowledge, Song et al. [2013] is the first work to use neural nets and, more generally, any nonlinear GPU power models; although similar ideas, namely, exploring more robust machine-learning techniques for power models, exist in the non-GPU power modeling literature. Specifically, Singh et al. [2009] apply nonlinear functions ($\log(\cdot)$, $\exp(\cdot)$) to performance metrics that are in turn input to piecewise linear functions.

4.2.4. Performance and Energy Model of Song et al. [2013]. While the models discussed above focused only on power prediction, Song et al. additionally propose a performance model to predict estimated runtime of the kernel. Used together, a model of energy (i.e., power integrated over time) for distributed systems is given [Song et al. 2013].

To estimate runtime of a kernel, Song et al. also propose a counter-based performance model. As the counter values correspond to hardware processes, a series of equations, similar to simulation models (such as in Hong and Kim [2009, 2010] discussed in Section 5), are used to calculate the number of cycles for executing the kernel. To validate the method, predicted times are compared against runtimes of kernels. An immediate advantage of this performance model is that time used by GPU sub-units are calculated to estimate the total time; consequently, architectural units involved in bottlenecks can be pinpointed.

An energy estimate for a GPU executing a kernel is given by the product of the average power, as predicted by the neural network power model discussed in Section 4.2.3, with the runtime predicted by the performance model. To obtain total energy used by a distributed system, the sum of the energy on each GPU executing the kernel is summed with the remaining systems idle energy (as estimated from direct measurements) and overhead energy, such as network energy. This systemwide energy estimate is tested on the truly parallel SHOC [2010] benchmarks. Further research for how such an energy model will scale to full programs will be an interesting and integral next step.

4.2.5. Tree-Based Energy Model of Chen et al. [2011]. Chen et al. [2011] provide a GPU energy model similar to the counter-based models discussed above. Unlike previous works, Chen et al. do not use counters observed during runtime on a GPU as model inputs but instead use instruction types as well as GPU counters and metrics observed when running the kernel in GPGPU-Sim (see Table V; GPGPU-Sim is an open-source and well-developed GPU simulator). Their method uses linear regression tree and random forest methods to predict energy of GPU kernels. Linear regression trees provide a decision tree by iteratively dividing the feature space, and random forests use an ensemble of trees to give the predicted output. Algorithmic details can be found in Breiman et al. [1984, 2001], among other sources. Power measurements are obtained using Yokogawa WT210 Digital Power Meter [Yokogawa Electric Corporation 2015] on an NVIDIA GeForce GTX 280 GPU with a total of 52 kernels from the NVIDIA CUDA SDK, Rodinia [2009], and Parboil [2012] benchmark suites and using graphics applications from their previous work [Chen et al. 2010]. For comparative evaluation, linear regression, regression tree, and random forest models are trained and results are obtained using leave-one-out cross validation, where each algorithm is iteratively tested on each benchmark after training on the remaining benchmarks. Results show superiority of random forests with average error of 7.77%, followed by regression trees, and finally linear regression with errors of 11.68% and 11.70%, respectively. This provides further evidence that the relationship between GPU power and GPU sub-processes is nonlinear, as did the neural network power model of Song

Table V. GPU Simulation Works

	Name	Description	Reference
1	Qsilver	Functional GPU Power and Temperature Simulator	Sheaffer et al. [2004, 2005a, 2005b], Section 5.1.1
2	Attila	GPU Functional Simulator	Del Barrio et al. [2006]
3	PowerRed	GPU Architecture-Level Simulator	Ramani et al. [2007], Section 5.1.2
4	UNISIM	Simulation Environment	August et al. [2007]
5	Barra	UNISIM-Based GPU Functional	Collange et al. [2009, 2010]
6	McPAT	General Simulation Environment	Li et al. [2009], Section 5.2
7	N/A	McPat-Based GPU Functional Simulator	Lim et al. [2013], Section 5.2.1
8	GPU-Wattch	McPat-Based GPU Power Simulator, Mature Open-Source Product that Integrates with GPGPU-Sim	Leng et al. [2013], Section 5.2.2
9	GPGPU-Sim	McPat-Based GPU Functional and Performance Simulator, Mature Open-Sourced Product	Bakhoda et al. [2009]; Fung et al. [2007], Section 5.2.2
10	N/A	GPU Power Model built on GPGPU-Sim Simulated Events	Chen et al. [2011], Section 4.2.5
11	MWP-CWP Performance Model	GPU Performance Model	Hong and Kim [2009], Section 5.3.1
12	Integrated Power & Performance Model	GPU Power & Performance Model	Hong and Kim [2010], Section 5.3.2
13	N/A	CPU Vs. GPU Performance Simulation using MWP-CWP Performance Model	Meng et al. [2011]
14	GPUPerf	GPU Performance Simulation and Optimization Framework Using MWP-CWP Performance Model	Sim et al. [2012]
15	N/A	PTX-Based GPU Power Model for Optimal User Settings	Wang and Ranganathan [2011], Section 5.4.1
16	POIGEM	PTX-Based GPU Energy Model Using MWP-CWP Performance Model	Zhao et al. [2013], Section 5.4.2
17	Multi2Sim	Heterogeneous System Simulator, Mature Open-Source Product	Ubal et al. [2012]
18	MacSim	Heterogeneous Architecture Cycle-Level Simulator, Mature Open Source Product	Kim et al. [2012]

Note: Table itemizes GPU simulation research works. Those with power predictions are discussed in detail, and the reference to their section number is provided.

et al. [2013]. Furthermore, the possible overfitting of the model induced by a large ratio of model inputs to data points is cited as a potential problem.

Although Chen et al.'s work is indeed a power model, their work is built on GPU simulation; hence, the power predictions can inform implementation decisions without use of the GPU, for example, for *a priori* optimization. This is an advantage and motivation of GPU power simulation, the topic of Section 5.

5. GPU POWER SIMULATORS

Counter-based modeling research gives promising techniques for estimation of power and performance, but the reliance on data collected during runtime inhibits their predictive capability. By using software to model GPU components, GPU simulations seek to give actual output and/or behavioral predictions of running a GPU application without the actual GPU. Functional simulators produce output of a given workload and are often supplemented with profiling information, while power/performance

simulators focus solely on predicting the power/performance profile of the application without the actual functional output. Such tools are useful for exploring hardware options and software configuration in the absence of the actual GPU, especially when optimizing for performance and energy efficiency. While simulations can give valuable predictions, the modeling involved is often very complicated, requiring a deep understanding of architecture and processes. This is most evident in the influential but complicated models of Hong and Kim [2010]. In these cases implementation across systems is inhibited, and configuration can be costly. As is the case with modeling efforts, simulations of power, temperature, and other computing characteristics were initially developed for CPUs (e.g., see Brooks et al. [2000], Skadron et al. [2003], and Zhang et al. [2003]), yet the complicated and evolving GPU architecture requires tailored simulation research. For example, see the power simulation work of Lim et al. [2013] where GPU and CPU architectural differences are itemized. GPU power simulators are often built on performance and power models using hardware data that is predicted rather than observed, for example, from code analysis.

A very general framework for power modeling introduced by Isci and Martonosi [2003] (originally designed for CPU research) is a basis for many GPU simulation works and reconfigurations for the new setting vary. Isci and Martonosi's framework defines total power as idle power + leakage power + dynamic power. Dynamic power, p_d, is modeled linearly as

$$p_d = \sum_i w_i \alpha_i p_{max_i}, \tag{2}$$

where i ranges through the sub-components, α_i denotes the access rate of sub-component i (i.e., the number of accesses of that component per application runtime), p_{max_i} is that sub-component's max power, and w_i are weights to be learned. Generally, micro-benchmarks are used with a power meter to acquire training data for learning the weights. Very complicated variants of this basic framework exist, for example, see Hong and Kim's model in Section 5.3.1.

Below we discuss in detail the works in which GPU power is directly simulated or is modeled from a GPU hardware simulation; furthermore, these are itemized in Table V. As GPU simulation is a large and growing field, we also include in Table V those GPU simulation efforts not directly addressing power prediction.

5.1. Early GPU Power Simulation Research

Early work in GPU power simulation occurs in works by Sheaffer et al. [2004, 2005a] and Ramani et al. [2007]. As the development of GPU architectures necessitated significant increases in power consumption, these works address the need for simulation research targeting GPUs with an end goal of optimizing power consumption via architectural design decisions.

5.1.1. Qsilver. Sheaffer et al. [2004, 2005a] introduce a GPU simulation framework called Qsilver and exhibited its use for optimizing energy efficiency. This research work illustrates the power and performance analysis that is possible via such a GPU simulation framework but uses a prototypical GPU pipeline to show the possibilities afforded by such a simulation. To obtain conclusive results for a specific GPU architecture, more detailed knowledge of the given GPU hardware is required. The framework, Qsilver, takes as input commands for the GPU along with statistics about the activity of the GPU necessary to complete the command. For example, the number of writes to a buffer may be a statistic accompanying a command. Next, a cycle-timer model simulates the workflow of the GPU and outputs operational counts for each GPU sub-unit. The counts are then used as input to a power model. To exhibit the predictive

capabilities of the simulator, predicted power, energy, and performance of a given task are plotted as GPU configuration details are changed. Hence, the optimal settings are identified, which result in predicted energy efficiency gains. Sheaffer et al. [2005b] build on this work to investigate temperature profiles. Although not immediately applicable to modern general purpose GPUs, Qsilver's overall idea of first predicting hardware events (from a model of the GPU architecture and the application code) and, second, using these as input to a power and/or performance model, is present in much of the simulation literature.

5.1.2. PowerRed. Ramani et al. [2007] develop PowerRed, a GPU architecture-level power simulation framework. Power models for GPU sub-components, such as buses and caches, are developed based on previous low level simulation research and explicit details of the models are not discussed. As the goal of this framework is to inform power-optimal GPU hardware designs, simulated power is used to explore the impact of changing the architecture; for example, the impact of the global bus's wire length on power consumption is simulated to inform how architectural changes can save energy. As in Sheaffer et al. [2004, 2005a], the goal is not a definitive model that is ready for use but to introduce a framework that, when equipped with specific GPU architectural details, can inform future design decisions for power savings.

5.2. McPAT-Based GPU Power Simulations

McPat is an architecture-level power profiling simulator introduced by Li et al. [2009] that allows users to configure abstractions of hardware sub-components to flexibly simulate a variety of CPU (and now GPU) designs. McPAT's library of sub-component abstractions, such as instruction decoders or data caches, can be hierarchically combined to represent diverse hardware [Li et al. 2009]; hence, recent strides in GPU simulation have occurred by configuring the McPat (abstractions of) sub-components in a way that will accurately model a GPU.

5.2.1. GPU Simulator and Power Model of Lim et al. [2013]. In order to build a GPU power simulator, Lim et al. [2013] configure McPAT to model each sub-component of an NVIDIA GTX580 GPU. More specifically, sub-components of the GPU are represented by combinations of McPAT sub-component models using a simulation interface from Song et al. [2011]. MacSim,[17] a publicly available heterogeneous architecture simulator (see Table V and Kim et al. [2012]), and DRAMSim2, a memory simulator, are also used [Rosenfeld et al. 2011]. Lim et al. use an NVIDIA GTX 580 GPU and an inline power meter to acquire power data. More specifically, running benchmarks while turning on and off sub-components is necessary to estimate CPU and GPU power in their configuration. In order to discover unknown McPat parameters (e.g., number of ports in L1 cached) necessary to accurately model sub-components, iterative experimentation and comparison to reference data are performed and discussed. Finally, each sub-component's power is modeled, and a linear combination is used for total GPU power. Evaluation of the simulation's power prediction yields 7.7% and 12.8% geometric averaged error over 10 micro-benchmarks and six Merge benchmarks (discussed by Linderman et al. [2008]), respectively.

5.2.2. GPUWattch. GPUWattch, developed by Leng et al. [2013], is (to our knowledge the only) open-source GPU power simulator and is built on and available with GPGPU-Sim (see Table V). (GPGPU-Sim is a thoroughly developed, open source GPU

[17]https://code.google.com/p/macsim/.

performance simulator [Bakhoda et al. 2009; Fung et al. 2007]).[18] Specifically, GPU micro-architectural components are modeled by corresponding McPAT simulators, and new micro-architectural simulated components are incorporated to account for GPU components lacking a McPAT analogue. Power is modeled as in the paragraph discussing Equation (2), and an NI Data Acquisition system (DC power meter) is used with micro-benchmarks to gather power data. Least-squares estimation is used for learning weights of the dynamic power, and idle power is determined by using micro-benchmarks with an imbalanced load—a known number of SMs compute while another known number sit idle. Evaluation of the power simulator against ground-truth power is performed on NVIDIA GTX 480 and Quadro FX5600 GPUs. Very comprehensive initial evaluation is performed, using 80 micro-benchmarks that stress different components, and a 15% and 16.2% average error is reported, respectively, for the two GPUs. Next, evaluation on 18 kernels from the RODINA Benchmarks (see Che et al. [2009]), three from the ISPASS suite (See Bakhoda et al. [2009]), and four others produce average error in power prediction of 9.9% and 13.4%, among the two GPUs tested, respectively.

As GPUWattch is created to integrate with GPGPU-Sim, a well-developed GPU performance simulator, these simulation tools facilitate optimization and testing opportunities for developing code. Perhaps the greatest operational boon of these works, as illustrated by this evaluation, is that these power simulators claim to be portable across architectures, unlike the MWP-CWP models discussed next.

5.3. Hong and Kim's Power and Performance Models

Hong and Kim developed a GPU performance model [2009] and, later, an integrated power model [2010]. These two influential works use a detailed understanding of intra-GPU parallelism to illuminate not only predictive capabilities but also *a priori* optimization techniques for performance, power, and energy efficiency. Further, their works have contributed to a flurry of GPU simulation research. The remainder of this subsection gives detailed descriptions of Hong and Kim's GPU power and performance simulation works.

5.3.1. Hong and Kim's MWP-CWP Performance Model. Hong and Kim's memory warp parallelism–computational warp parallelism (MWP-CWP) performance simulator illuminates intra-GPU parallelism from static code analysis and has lead to power simulation; hence, we give a description including the main ideas [Hong and Kim 2009]. The model is based on the intra-GPU process, where, first, a warp computational period is completed, and then the SM executes a memory instruction from the warp, and that warp waits until the memory request is completed. We note that the use of warps, defined as 32 SIMD (single instruction parallel data) threads to be simultaneously executed, is dependent on the CUDA programming architecture and hence the NVIDIA hardware. Estimating the number of computational periods, number of memory waiting periods, and the parallelism (overlap) possible yields an accurate prediction of the time necessary for the kernel to complete.

Inputs to the model, obtained from code at compile time, are as follows: number of instructions, number of memory requests, memory access patterns, GPU architecture parameters (such as, bandwidth, DRAM latency, etc.), and warp information. A *warp* is a batch of 32 threads and is executed as a unit on an SM. There are two metrics that drive the simulator: (1) Memory Warp Parallelism (*MWP*) defined as the number

[18]GPUWattch, and its counterpart GPGPU-Sim provide websites with evaluation results, support documentation, compilation information and further descriptions. http://gpgpu-sim.org/ http://gpgpu-sim.org/gpuwattch/.

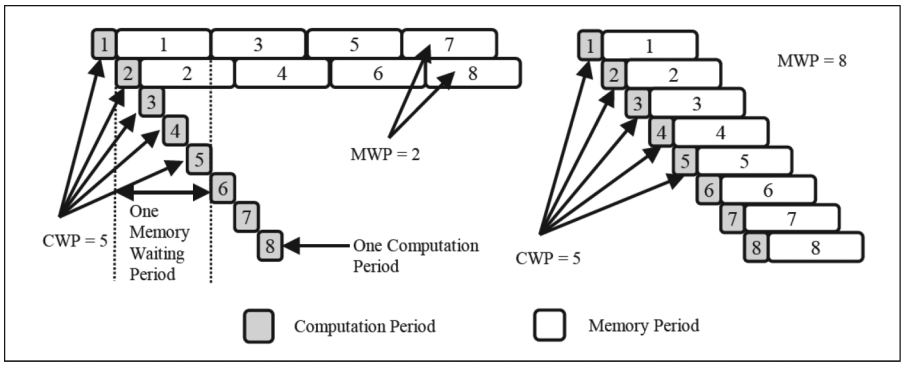

Fig. 1. MWP-CWP Model illustration. MWP < CWP on the left and CWP < MWP on the right. Taken from both [Hong and Kim 2009; Sim et al. 2012].

of warps per SM that can access memory simultaneously, and (2) Computational Warp Parallelism (*CWP*), defined as the number of warps that an SM can execute during one memory warp waiting period plus one (plus one to account for the warp that is waiting on memory). See Figure 1.

The model of GPU activity involves three cases and illustrates how power and performance costs accrue in GPUs.

(1) **Case 1 MWP < CWP.** In this case, multiple computational periods are executed while waiting on memory requests from a previous computational period, hence total time is determined mainly by memory waiting time.
(2) **Case 2 MWP ≥ CWP.** In this case, memory requests occur directly following each computational period with no backup, and so total time is determined mainly by the computation.
(3) **Case 3 Insufficient Warps.** The third case is when a lack of parallelism results in computational and memory periods acting in succession.

Derivation of MWP and CWP involves dozens of formulas constructed from an even greater number of parameters inferred from input code (e.g., counts of a variety of instructions, overhead parameters, waiting times, and various types of parallelism); hence, the model involves expert understanding of the architecture and kernels involved. Tests of the simulated performance against actual runtimes yields geometric mean of absolute error as 5.4% on micro-benchmarks and 13.3% on GPU applications. An immediate boon of this simulation is an understanding of the intra-GPU parallelism at compile time, allowing performance bottlenecks to be identified as either computational or memory expense. Performance optimization is also informed by this model and is the topic of Sim et. al [2012]. The intuition behind increasing performance via the model follows the cases outlined in Section 5.3.1. For example, if MWP < CWP (runtime is determined almost exclusively by memory), optimizing computational performance will have almost no effect, but optimizing memory performance yields worthwhile performance gains. Additionally, Meng et al. [2011] use the MWP-CWP performance simulator to predict from CPU code the potential benefits of implementation with general purpose GPUs [Meng et al. 2011]. In the 2010 work, Hong and Kim couple this performance simulator with a power simulator and predict the optimal number of SPs for energy efficiency *a priori*. This is the topic of their integrated power and performance model, discussed next.

5.3.2. Hong and Kim's Integrated Power and Performance Model. Hong and Kim's Integrated Power and Performance (IPP) model [Hong and Kim 2010] builds a power simulator on their MWP-CWP performance predictor (Section 5.3.1). Together, the Integrated Power and Performance model is used to decide the number of cores, that is, multiprocessors of the GPU needed for optimal energy efficiency (operations/watt). The intuition behind the study is that energy efficiency will increase with the number of cores until the memory bandwidth is maximized. After this point, adding additional cores will decrease energy efficiency, as additional cores will be consuming energy while waiting for memory bandwidth. As with the performance model discussed above, the power simulation is based on instructions garnered from static analysis of input code. Thus, in practice, such a simulator would inform *powergating*, in which the software engineer or compiler imposes limits on the number of cores allowed to execute the kernel.

Given a kernel, the MWP-CWP model (as discussed in detail in Section 5.3.1) predicts runtimes and is subsequently used to estimate power by a model, which at a high level is identical to that discussed with Equation (2), although estimates of runtime power for sub-components are complicated. As with the MWP-CWP model, the technicalities involved in the power estimate involve detailed, architecture-dependent models with over 70 parameters; hence, a precise description of the model is outside the scope of this survey. A high-level description is as follows. GPU power is defined as idle power plus runtime power. To estimate runtime power, instructions from input code are sorted by the architectural sub-unit(s) (e.g., SFU, global memory, etc.) they call, and component power is estimated by a combination of logarithmic and linear functions from the instructions rather than performance counters. It is worth noting that a latent temperature model is also estimated and used as input to the runtime power model to account for their high correlation. Micro-benchmarks, which stress particular sub-units and are mostly from the Merge benchmark suite (Linderman et al. [2008]), are used to fit model parameters to measured power. Evaluation of the power simulator on 11 benchmark kernels gives 9% geometric average error against measured power.

In order to maximize energy efficiency (operations/watt), the MWP-CWP cases give insight into the optimal number of cores to use. Intuitively, in the case where MWP $<$ CWP (computational periods are intermittently waiting on memory), dialing back the number of cores can conserve power without affecting performance. Applying the model to five benchmarks that stress memory bandwidth shows a predicted average energy savings of 11%. The interested reader is encouraged to consult the original documents for details of both models and the optimization equations.

The drawbacks of the IPP model lie in the high degree of technicality of the model and the dependence of the model on the hardware setup. As a research contribution, the work has important implication for GPU power conservation. First, it exhibits strong evidence that managing of the number of cores is both worthwhile and necessary for power conservation. Second, it gives results suggesting exactly how to optimize the number of cores. Third, it exhibits optimization results at compile time, so configuration details can be set *a priori*. Last, the MWP-CWP models intra-GPU parallelism into, evidently, useful cases with respect to performance and efficiency optimization.

5.4. PTX-Based Power Models

As an intermediate step in the NVIDIA CUDA compilation between human-written CUDA code (written in C) and the compiled binary, "parallel thread execution" (PTX), a pseudo-assembly code, gives the list of instructions to be executed by the GPU. For example, *add, xor, bra (branch)* are a few of the ∼100 PTX instructions. To access PTX code, one can simply run `nvcc -ptx file-name`, which produces the PTX code during compilation [NVIDIA 2014b]. In order to predict power *a priori*, PTX instructions have been leveraged to simulate the power of a CUDA program. Such an approach gives

powerful optimization capabilities to programmers before running a program, but this relies on the sometimes untenable hypothesis that the program has a transparent set of instructions. More specifically, the disadvantage of any predictive model that depends on static code analysis is in the inability to know *a priori* how many instructions will be executed, and theoretical results have proven this to be insurmountable—see Turing's work on *the halting problem*, where it is proven that it is undecidable to know whether a given program and input will run forever [Turing 1936]. As an illustrative example, consider a program that counts the number of independent and identically distributed coin flips before observing a tail. In a simple implementation, any given run of this program will have an indeterminable number of instructions; hence, an accurate energy prediction of a sample run is inhibited. In short, when optimizing CUDA code for many tasks, such as matrix multiplication, PTX-based power models can yield valuable and informative predictions, but users should be wary of conditional statements that can inadvertently cloak the instruction-set that will be run by the GPU.

5.4.1. Wang and Ranganathan's Instruction Level Energy Optimization. As CUDA programming necessitates the programmer to specify the number of threads per block and number of blocks per grid, Wang and Ranganathan [2011] addresses the problem of finding the optimal user settings for energy optimization. To develop an energy model, a subset of the most common PTX instructions along with their frequency are taken as input to a linear model. Explicitly, the energy used by a thread is

$$E_t := \sum_i e_i n_i + o_1,$$

where i ranges over the PTX instructions modeled, e_i is the energy used to execute instruction i, n_i is the frequency of instruction i, and o_1 is thread overhead. Altogether, total energy of a program is modeled as

$$E := [ME_t + o_2(M)]x + o_3(M, x), \tag{3}$$

where M is number of threads per block and x is the number of blocks. The function $o_2(M)$ denotes the overhead energy on the block level, and $o_3(M, x)$ is the overhead energy for the creation of all the blocks. Both o_2 and o_3 are assumed to be linear. In order to learn the model parameters, test programs stressing the instructions in the model are run on an NVIDIA GTX 460 GPU using Zalman VPM1 VGA Power Consumption Meter[19] (for monitoring the GPU power only). Evaluation is performed on four benchmarks created from the NVIDIA CUDA SDK and one Advanced Encryption Standard (AES) program [Manavski et al. 2007]. Results show energy consumption savings between 7.31% to 11.76% with worst-case performance loss of 4.92%. While the authors remark that this energy model is likely less accurate in predicting energy than counter-based models (which require repeated running of a program to observe hardware events), their approach gives users the ability to optimize energy efficiency before running the program and specifically answers the question of how to set necessary user design parameters.

Although outside the scope of this survey, we point the interested reader to a related work of Jia et al. [2015], where tree-based methods are used to explore the design space (number of blocks, number of threads per block) for energy- and performance-optimal CUDA programming.

5.4.2. POIGEM. Influenced by Hong and Kim's MWP-CWP performance model, Zhao et al. [2013] also develop an energy model for the NVIDIA GE Force GTX 470 GPU.

[19]http://www.zalman.com/.

Their model, called A Programming-Oriented Instruction Level GPU Energy Model for CUDA Program (POIGEM), gives energy prediction for the GPU with the counts of each PTX instruction as input. Specifically, a subset of PTX instructions are selected and divided into two categories—memory instructions and arithmetic computation instructions. Energy, E is modeled as

$$E = k_a T \sum_{i=1}^{m} \alpha_i c_i + k_m T \sum_{i=m+1}^{m+n} \alpha_i c_i + \beta t, \tag{4}$$

where α_i denotes the energy consumed to perform instruction i, c_i gives the number of times instruction i occurs, and instructions $i = 1, \ldots, m$ are the arithmetic computation instructions, while instructions $i = m + 1, \ldots, n + m$ are the memory instructions. Parameters k_a and k_m give the contribution of the arithmetic and memory instructions parts, respectively, and T gives the number of threads. Last, t is the time it takes for the kernel to run as determined by the MWP-CWP model (see Section 5.3.1), and β is the overhead energy consumption.

Training of the model is performed on 15 synthetic benchmarks, and α_i, k_a, k_m, and β as learned via linear regression are given in the article. Ground-truth energy data are obtained using two PSUs, one serving only the GPU and the other serving the remaining components (CPU, motherboard, etc.). An Everfine[20] PF9805 external power meter is used between the GPU's PSU and the wall outlet. Evaluation is performed using 16 Rodinia benchmarks [Che et al. 2009]. On 13 of the 16 benchmarks, less than 10% error is achieved, yet on the other three benchmarks, greater than 12% error results with a maximum error over 25%. While increasing the training size will presumably increase accuracy on most benchmarks, the cited problem for the three problematic benchmarks is the inability to know from PTX code how many times each instruction will occur. On the other hand, given a simple program for which the instruction set is transparent, such as a matrix multiplication task that does not contain conditional statements, this work promises an accurate energy prediction on compilation. Moreover, by training the model on benchmarks with bare-faced instruction sets, POIGEM gives valuable insight into the energy usage of the GPU, specifically, by estimating the energy cost of each instruction.

5.4.3. Activity-Based Model for GPUs. By modeling the power costs of basic GPU subprocesses, such as global memory access and floating point operations, Kasichayanula et al. [2012] develop a model of GPU sub-component power called the Activity-Based Model for GPUs (AMG). Their model defines GPU total power as idle power + runtime power, with runtime power the sum of the active components power. More specifically, runtime power of component i is defined as

$$N * P_i * U_i + B_i * U_i,$$

where N is the number of SMs executing the given sub-process, P_i is the power used for that sub-component, U_i is the utilization factor, and B_i is the base power of that sub-component. The Kill A Watt inline power meter is used to obtain ground-truth power readings via a second PSU that only services the GPU. Micro-benchmarks are crafted to stress different sub-components, and constants B_i, U_i, and P_i are observed. PTX code is needed to see what instructions and hence what sub-components will be called. Evaluation of the AMG power prediction is performed using an NVIDIA C2075 GPU and four MAGMA kernels (see Table I, Dongarra and Tomov [2014]), and error is near 10%. After training this model, runtime power can be predicted given knowledge of what sub-components are called, for example, from PTX code.

[20]http://www.everfine.net/.

6. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Altogether, this survey seeks to give a comprehensive picture of methods to attain power data of GPUs, with an in-depth view of important pieces of modeling and simulation research. As all power decisions are ultimately grounded in accurate measurements, direct methods are extremely important and still exhibit a tradeoff between accuracy and ease of use. The internal GPU power sensors (in particular, in NVIDIA Fermi and Tesla architectures) are the most convenient to use as they require no extra hardware cost or configuration and are supported by vendor profiling software. Although the use of internal sensors is becoming common in the research literature, the vendor documentation is scarce. Further, as illustrated by Burtscher et al. [2014], more thorough understanding is necessary before precise measurements from internal sensors can be used. Considering the influence internal power sensors are poised to have, a third-party research effort to compare internal sensor results to a ground-truth external meter is necessary for researchers and practitioners to rely on the accuracy of their readings.

Our survey also showed a progression of power observation hardware targeting HPC clusters. Ideally, the multi-level power data made possible by these hardware advances will inform greater understanding and optimizing of whole system power. This presents an opportunity for integration of component-level (in particular, GPU) power models into a larger, more comprehensive framework.

While significant progress has been made in the counter-based power modeling research (Section 4), many research questions for correlating power to GPU hardware events have arisen. First, modern architectures are giving unprecedented insight into the GPU's hardware activity—for example, NVIDIA Tesla architectures can access over 200 counters and metrics. Furthermore, we expect the trend of increased GPU hardware transparency to continue; for example, new research to create more flexible tools for profiling of GPU hardware events is underway (see SASSI of Stephenson et al. [2015]). Yet most power modeling research seeks to learn model parameters from power observations of $\sim$50 benchmarks. From a power modeling point of view, this increased visibility to hardware events is both a boon and a curse. The result is an imbalance between the number of model inputs (hardware events) and the number of data points, and this presents an overfitting problem. New methods or much larger benchmark-to-power data sets are needed to harness the hardware event information now available. Second, an advantage of the power modeling research is understanding the power costs of GPU sub-processes. How to leverage this insight for software or hardware design is a natural follow-on question. Last, accurately modeling power costs of imbalanced loads (where there are inconsistent portions of the cores per SM used) is an outstanding issue that was cited by many modeling works.

Modeling efforts are complemented by simulation work, which seeks to predict GPU events, power, performance, and/or functional output without running code on actual hardware. Such simulations are necessary for informing novel hardware designs; furthermore, programmers benefit from the power modeling and simulation research as it sheds light on their code design space decisions. Unfortunately, much of the simulation work requires a large amount of architecture-specific knowledge, and the dependence on such details inhibits the simulation's application to different GPU architectures. Research to provide a GPU power simulation framework, which is both flexible and accurate but requires less expertise, is a natural direction for future research. This survey revealed that GPU power simulation from static code analysis has seen many exciting advancements, which are extremely valuable for programmers looking to optimize power, energy efficiency, or performance via their design space. Advancing these ideas to an open source or commercially available and widely accepted tool will assist programmers in such decisions.

For general computer power monitoring, software-based power measurement tools, made possible by the modeling and simulation developments, are emerging. These technologies provide similar convenience to internal sensors and are also not well evaluated; for example, see the work of Bekaroo et al. [2014] that examines two software computer power meters, Microsoft's Joulemeter and HP's Power Assistant. Aside from the Intel Power Gadget [2014], which gives software power measurement for Intel processors with onboard GPUs, such technologies are not yet available for GPUs. In addition to convenience, such tools open powerful opportunities not just for in situ monitoring but also for on-the-fly reconfiguration (e.g., to assist with power constraints) and for refining models from real-time observations (e.g., from an internal sensor). To the best of the authors' knowledge, in situ software tools that use and update power modeling efforts do not exist for GPUs yet. The advancements in modeling and simulation discussed above are the prerequisites for such tools, and examples exist in CPU-focused market. For instance, Intel's power-clamping tool, called Running Average Power Limit (RAPL), uses a linear counter-based power model to predict memory power in situ and updates the model in an online fashion. This allows automated restrictions on memory bandwidth to prevent spikes in power [David et al. 2010]. How to harness the GPU modeling and simulation advancement to create automated, in situ technologies for reigning in GPU power consumption while maximizing performance is a natural next step to the research discussed in this survey.

Finally, just as system architectures are trending toward heterogeneity (e.g., with CPUs orchestrating and GPUs or coprocessors computing), intra-processor architectures are integrating different processing units designed to compliment each other for more optimal performance and power efficiency. We have observed that the CPU modeling and simulation research provide a jumping-off point for GPU-focused research. Similarly, we expect the current power modeling and simulation research to not just be tailored to more architectural designs but also to generalize, enveloping a more general set of architectures and components, perhaps hierarchically (at the processor level and the system level). For example, is there a generalizable set of processes that can be used to model any type of processor (e.g., CPU, GPU, APU, FPGA) or system of processors and give accurate estimates of performance and power? Such a general model would ideally be validated on multiple, different heterogeneous architectures. Success in this endeavor would provide predictive capabilities for how to combine different architectures intelligently.

ACKNOWLEDGMENTS

The authors thank the reviewers for their helpful suggestions and to the editorial committee for the opportunity to address reviewers' suggestions.

REFERENCES

2014. Advanced Configuration and Power Interface (ACPI) website. Retrieved from http://www.acpi.info

2014. Penguin Computing Releases New Power Monitoring Device. Retrieved from http://www.penguin computing.com/ resources/press-releases/penguin-computing-releases-new-power-monitoring-device.

AMD. 2015. AMD GPU Performance API User Guide. Retrieved from http://developer.amd.com/tools-and-sdks/graphics-development/GPUperfapi/.

Sebastian Anthony. 2013. China's Tianhe-2 supercomputer, twice as fast as DoE's Titan, shocks the world by arriving two years early. Retrieved from http://goo.gl/oQQcrT.

Sebastian Anthony. 2014. Supercomputer stagnation: New list of the world's fastest computers casts shadow over exascale by 2020. Retrieved from http://goo.gl/gqFp6f.

ARM. 2009-2011. Mali GPU Shader Development Studio User Guide. Retrieved from http://infocenter. arm.com/help/topic/com.arm.doc.dui0504c/DUI0504C_mali_.

Steve Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, and others. 2010. The opportunities and challenges of exascale computing. (2010). http://science.energy.gov/~/media/ascr/ascac/pdf/reports/exascale_subcommittee_report.pdf.

David August, Jonathan Chang, Sylvain Girbal, Daniel Gracia-Perez, Gilles Mouchard, David Penry, Olivier Temam, and Neil Vachharajani. 2007. Unisim: An open simulation environment and library for complex architecture design and collaborative development. *Comput. Arch. Lett.* 6, 2 (2007), 45–48.

Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'09)*. IEEE, 163–174.

Daniel Bedard, Min Yeol Lim, Robert Fowler, and Allan Porterfield. 2010. PowerMon: Fine-grained and integrated power monitoring for commodity computer systems. In *Proceedings of the IEEE SoutheastCon 2010*. IEEE, 479–484.

Girish Bekaroo, Chandradeo Bokhoree, and Colin Pattinson. 2014. Power measurement of computers: Analysis of the effectiveness of the software based approach. *Int. J. Emerg. Technol. Adv. Eng.* 4, 5 (2014), 755–762.

Frank Bellosa. 2000. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European workshop: Beyond the PC: New Challenges for the Operating System*. ACM, 37–42.

Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. 1984. *Classification and Regression Trees*. CRC Press.

David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. *ACM* 28, 2 (2000), 83–94.

Martin Burtscher, Ivan Zecena, and Ziliang Zong. 2014. Measuring GPU power with the K20 built-in sensor. In *Proceedings of Workshop on General Purpose Processing Using GPUs*. ACM, 28.

Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'09)*. IEEE, 44–54.

Jianmin Chen, Zhuo Huang, Feiqi Su, Jih-Kwon Peir, Jeff Ho, and Lu Peng. 2010. Weak execution ordering-exploiting iterative methods on many-core GPUs. In *Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS)*. IEEE, 154–163.

Jianmin Chen, Bin Li, Ying Zhang, Lu Peng, and Jih-kwon Peir. 2011. Statistical GPU power analysis using tree-based methods. In *Proceedings of the 2011 International Green Computing Conference and Workshops (IGCC)*. IEEE, 1–6.

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 *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 225–236.

Sylvain Collange, Marc Daumas, David Defour, and David Parello. 2010. Barra: A parallel functional simulator for GPGPU. In *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 351–360.

Sylvain Collange, David Defour, and David Parello. 2009. Barra, a modular functional GPU simulator for GPGPU. Retrieved from http://hal.archives-ouvertes.fr/hal-00359342.

Cray. 2014. OpenACC Website. Retrieved from http://www.openacc-standard.org.

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 *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 63–74.

Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: Memory power estimation and capping. In *Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. IEEE, 189–194.

David DeBonis, Ryan E. Grant, Stephen L. Olivier, Michael Levenhagen, Suzanne M. Kelly, Kevin T. Pedretti, and James H. Laros. 2014. A power API for the HPC community. In *Supercomputing 2014 Posters*.

David DeBonis, James H. Laros, and Kevin Pedretti. 2013. *Qualification for PowerInsight Accuracy of Power Measurements*. Technical Report. Sandia National Laboratory.

Victor Moya Del Barrio, Carlos González, Jordi Roca, Agustín Fernández, and Roger Espasa. 2006. ATTILA: A cycle-level execution-driven simulator for modern GPU architectures. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 231–241.

Jack J. Dongarra and Stanimire Tomov. 2014. *Matrix Algebra for GPU and Multicore Architectures (MAGMA) for Large Petascale Systems*. Technical Report. University of Tennessee, Knoxville.

Paul J. Drongowski and Boston Design Center. 2008. *Basic Performance Measurements for AMD Athlon 64, AMD Opteron and AMD Phenom Processors*. Technical Report. Advanced Micro Devices, Inc. Boston Design Center.

Xizhou Feng, Rong Ge, and Kirk W. Cameron. 2005. Power and energy profiling of scientific applications on distributed systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*. IEEE, 34–34.

Wilson W. L. Fung, Ivan Sham, George Yuan, and Tor M. Aamodt. 2007. Dynamic warp formation and scheduling for efficient GPU control flow. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40)*. IEEE Computer Society, Washington, DC, 407–420. DOI: http://dx.doi.org/10.1109/MICRO.2007.12

Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. 2010. PowerPack: Energy profiling and analysis of high-performance systems and applications. *IEEE Trans. Parallel Distrib. Syst.* 21, 5 (2010), 658–671.

John C. Gordon, Andrew T. Fenley, and Alexey Onufriev. 2008. An analytical approach to computing biomolecular electrostatic potential. II. Validation and applications. *J. Chem. Phys.* 129, 7 (2008), 075102.

Green500. 2015. Green500 List. Retrieved from http://www.green500.org/lists/green201511.

Michael Groeger. 2005. An Introduction to Performance Counters. Retrieved from http://www.codeproject.com/Articles/8590/An-Introduction-To-Performance-Counters.

Matthew Grove. 2011. System G PowerPack. Retrieved from http://scape.cs.vt.edu/~mjeg/blog/2011/07/15/systemg-powerpack.

Sunpyo Hong and Hyesoon Kim. 2009. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. In *ACM SIGARCH Computer Architecture News*, Vol. 37. ACM, 152–163.

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

Joel Hruska. 2014. Supercomputing director bets $2,000 that we won't have exascale computing by 2020. Retrieved from http://www.extremetech.com/computing/155941-supercomputing-director-bets-2000-that-we-wont-have-exascale-computing-by-2020.

Song Huang, Shucai Xiao, and Wu-chun Feng. 2009. On the energy efficiency of graphics processing units for scientific computing. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009*. IEEE, 1–8.

Canturk Isci and Margaret Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 93.

Sverre Jarp, Ryszard Jurga, and Andrzej Nowak. 2008. Perfmon2: A leap forward in performance monitoring. In *Journal of Physics: Conference Series*, Vol. 119. IOP Publishing, 042017.

Wenhao Jia, Elba Garza, Kelly A. Shaw, and Margaret Martonosi. 2015. GPU performance and power tuning using regression trees. *ACM Trans. Archit. Code Optim.* 12, 2, Article 13 (May 2015), 26 pages. DOI: http://dx.doi.org/10.1145/2736287

Russ Joseph and Margaret Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*. ACM, 135–140.

Kiran Kasichayanula, Dan Terpstra, Piotr Luszczek, Stan Tomov, Shirley Moore, and Gregory D. Peterson. 2012. Power aware computing on GPUs. In *Proceedings of the 2012 Symposium on Application Accelerators in High Performance Computing (SAAHPC)*. IEEE, 64–73.

Khronos. 2014. Khronos OpenCL Website. Retrieved from https://www.khronos.org/opencl.

Hyesoon Kim, Jaekyu Lee, Nagesh B. Lakshminarayana, Jaewoong Sim, Jieun Lim, and Tri Pho. 2012. MacSim: A CPU-GPU heterogeneous simulation framework user guide. *Georgia Institute of Technology* (2012).

Seung-Woo Kim, Joseph Jin-Sung Lee, Vardhan Dugar, and Jun De Vega. 2014. Intel® power gadget. *Intel Corporation* 7 (2014).

Milind Kulkarni, Martin Burtscher, Calin Casçaval, and Keshav Pingali. 2009. Lonestar: A suite of parallel irregular programs. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'09)*.

James H. Laros, David DeBonis, Ryan Grant, Suzanne M. Kelly, Michael Levenhagen, Stephen Olivier, and Kevin Pedretti. 2014. *High Performance Computing Power Application Programming Interface Specification*. Technical Report SAND2014-17061. Sandia National Laboratory.

James H. Laros, Phil Pokorny, and David DeBonis. 2013. PowerInsight-A commodity power measurement capability. In *2013 International Green Computing Conference (IGCC)*. IEEE, 1–6.

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 *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ACM, 487–498.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009. MICRO-42.* IEEE, 469–480.

Tao Li and Lizy Kurian John. 2003. Run-time modeling and estimation of operating system power consumption. *ACM SIGMETRICS Perf. Eval. Rev.* 31, 1 (2003), 160–171.

Jieun Lim, N. Lakshminarayana, Hyesoon Kim, William Song, Sudhakar Yalamanchili, and Wonyong Sung. 2013. *Power Modeling for GPU Architecture using McPAT*. Technical Report. Georgia Institute of Technology.

Michael D. Linderman, Jamison D. Collins, Hong Wang, and Teresa H. Meng. 2008. Merge: A programming model for heterogeneous multi-core systems. In *ACM SIGOPS Operating Systems Review*, Vol. 42. ACM, 287–296.

David Luebke and Greg Humphreys. 2007. How GPUs work. *IEEE Comput.* 40, 2 (2007), 96–100.

Xiaohan Ma, Mian Dong, Lin Zhong, and Zhigang Deng. 2009. Statistical power consumption analysis and modeling for GPU-based computing. In *Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems (HotPower)*.

Svetlin Manavski and others. 2007. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography. In *Proceedings of the IEEE International Conference on Signal Processing and Communications (ICSPC) 2007.* IEEE, 65–68.

Chris McClanahan. 2010. History and Evolution of GPU Architecture. Retrieved from http://mcclanahoochie.com/blog/wpcontent/uploads/2011/03/.

Jiayuan Meng, Vitali A. Morozov, Kalyan Kumaran, Venkatram Vishwanath, and Thomas D. Uram. 2011. GROPHECY: GPU performance projection from CPU code skeletons. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 14.

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 (Aug. 2014), 23 pages. DOI: http://dx.doi.org/10.1145/2636342

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

NVIDIA. 2011. CUDA tools SDK CUPTI user's guide. Retrieved from http://hpc.oit.uci.edu/nvidia-doc/sdk-.

NVIDIA. 2012. NVML API reference manual. *NVIDIA Management Library* Retrieved from http://developer.download.nvidia.com/assets/.

NVIDIA. 2014a. NVIDIA CUDA Website. Retrieved from http://www.nvidia.com/object/CUDA_home_new.html.

NVIDIA. 2014b. NVIDIA Parallel Thread Execution ISA Version 4.1. Retrieved from http://docs.nvidia.com/cuda/parallel-thread-execution/#axzz3NglOuYX2.

NVIDIA. 2014c. NVML API Reference Guide. Retrieved from http://docs.nvidia.com/deploy/nvml-api/.

NVIDIA. 2014. PAPI CUDA Component Website. Retrieved from https://developer.nvidia.com/papi-.

NVIDIA. 2015. CUDA Toolkit Documentation Version 6.5: Profiler User's Guide. (2015). http://docs.nvidia.com/cuda/profiler-users-guide#axzz3NglOuYX2 accessed 2015-03-25.

Lynn Nystrom. 2008. University partners with Apple and Mellanox for energy efficient 22.8 Tflop supercomputer. Retrieved from http://www.vtnews.vt.edu/articles/2008/11/2008-745.html.

Karthik Ramani, Ali Ibrahim, and Dan Shimizu. 2007. PowerRed: A flexible modeling framework for power efficiency exploration in GPUs. In *Proceedings of theWorskshop on GPGPU.*

David Robson. 2008. From CPU to GPU, David Robson on the processing burden being shared by GPUs. Retrieved from http://www.scientific-computing.com/hpcforscience/feature-gpu.html.

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*.

Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *Comput. Arch. Lett.* 10, 1 (2011), 16–19.

Barry Rountree, Dong H. Ahn, Bronis R. de Supinski, David K. Lowenthal, and Martin Schulz. 2012. Beyond DVFS: A first look at performance under a hardware-enforced power bound. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. IEEE, 947–953.

Stuart Russell, Peter Norvig, and Artificial Intelligence. 1995. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ (1995).

Jeremy W. Sheaffer, David Luebke, and Kevin Skadron. 2004. A flexible simulation framework for graphics architectures. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*. ACM, 85–94.

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 *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 54–65.

Jaewoong Sim, Aniruddha Dasgupta, Hyesoon Kim, and Richard Vuduc. 2012. A performance analysis framework for identifying potential benefits in GPGPU applications. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 11–22.

Graham Singer. 2013. The history of the modern graphics processor. *Techspot* Retrieved from http://www.techspot.com/article/650-history-of-the-GPU/.

Karan Singh, Major Bhadauria, and Sally A. McKee. 2009. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Comput. Arch. News* 37, 2 (2009), 46–55.

Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. 2003. Temperature-aware microarchitecture. In *ACM SIGARCH Computer Architecture News*, Vol. 31. ACM, 2–13.

Ryan Smith. 2014. NVIDIA Volta IBM POWER9 Land Contracts for New US Governmemt Supercomputers. AnandTech. Retrieved from http://www.anandtech.com/show/8727/nvidia-ibm-supercomputers.

Alex J. Smola and Bernhard Schölkopf. 2004. A tutorial on support vector regression. *Stat. Comput.* 14, 3 (2004), 199–222.

Shuaiwen Song, Chunyi Su, Barry Rountree, and Kirk W. Cameron. 2013. A simplified and accurate model of power-performance efficiency on emergent GPU architectures. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 673–686.

William J. Song, Minki Cho, Sudhakar Yalamanchili, Saibal Mukhopadhyay, and Arun F. Rodrigues. 2011. Energy introspector: Simulation infrastructure for power, temperature, and reliability modeling in manycore processors. *SRC TECHCON* (September 2011), 4.

Brinkley Sprunt. 2002. The basics of performance-monitoring hardware. *IEEE Micro* 22, 4 (2002), 64–71.

Mark Stephenson, Siva Kumar Sastry Hari, Yunsup Lee, Eiman Ebrahimi, Daniel R. Johnson, David Nellans, Mike OConnor, and Stephen W. Keckler. 2015. Flexible software profiling of GPU architectures. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 185–197.

John A. Stratton, Christopher Rodrigues, I.-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen-mei W. Hwu. 2012. Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing* (2012), 12.

Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. 2010. Collecting performance data with PAPI-C. In *Tools for High Performance Computing 2009*. Springer, 157–173.

Matthew Tolentino and Kirk W. Cameron. 2012. The optimist, the pessimist, and the global race to exascale in 20 megawatts. *Computer* 45, 1 (2012), 0095–97.

Top500. 2016. Top500 List. Retrieved from http://www.top500.org/list/2015/11.

Alan Mathison Turing. 1936. On computable numbers, with an application to the Entscheidungsproblem. *J. Math.* 58 (1936), 345–363.

Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. 2012. Multi2Sim: A simulation framework for CPU-GPU computing. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. ACM, 335–344.

Yash Ukidave, Fanny Nina Paravecino, Leiming Yu, Charu Kalra, Amir Momeni, Zhongliang Chen, Nick Materise, Brett Daley, Perhaad Mistry, and David Kaeli. 2015. NUPAR: A benchmark suite for modern GPU architectures. In *ICPE*, Vol. 2015.

Yue Wang and Nagarajan Ranganathan. 2011. An instruction-level energy estimation and optimization methodology for GPU. In *Proceedings of the 2011 IEEE 11th International Conference on Computer and Information Technology (CIT)*. IEEE, 621–628.

Vincent M. Weaver, Daniel Terpstra, Heike McCraw, Matt Johnson, Kiran Kasichayanula, James Ralph, John Nelson, Philip Mucci, Tushar Mohan, and Shirley Moore. 2013. PAPI 5: Measuring power, energy, and the cloud. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 124–125.

Thomas Willhalm. 2012. Overview of Windows Performance Monitor. Retrieved from https://software.intel.com/en-us/articles/intel-performance-counter-monitor-a-better-way-to-measure-.

Yokogawa Electric Corporation. 2015. WT210/WT230 Digital Power Meter User's Manual. Retrieved from http://exodus.poly.edu/~kurt/manuals/manuals/Other/YOKOGAWA\%20WT210,\%20WT230\%20USER.pdf.

Yan Zhang, Dharmesh Parikh, Karthik Sankaranarayanan, Kevin Skadron, and Mircea Stan. 2003. *Hotleakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects*. Technical Report. University of Virginia, Department of Computer Science.

Qi Zhao, Hailong Yang, Zhongzhi Luan, and Depei Qian. 2013. POIGEM: A programming-oriented instruction level GPU energy model for CUDA program. In *Algorithms and Architectures for Parallel Processing*. Springer, 129–142.

Received May 2015; revised June 2016; accepted June 2016