Power Modeling for GPU Architectures Using McPAT

JIEUN LIM, Seoul National University
NAGESH B. LAKSHMINARAYANA, HYESOON KIM, WILLIAM SONG,
and SUDHAKAR YALAMANCHILI, Georgia Institute of Technology
WONYONG SUNG, Seoul National University

Graphics Processing Units (GPUs) are very popular for both graphics and general-purpose applications. Since GPUs operate many processing units and manage multiple levels of memory hierarchy, they consume a significant amount of power. Although several power models for CPUs are available, the power consumption of GPUs has not been studied much yet. In this article we develop a new power model for GPUs by utilizing McPAT, a CPU power tool. We generate initial power model data from McPAT with a detailed GPU configuration, and then adjust the models by comparing them with empirical data. We use the NVIDIA's Fermi architecture for building the power model, and our model estimates the GPU power consumption with an average error of 7.7% and 12.8% for the microbenchmarks and Merge benchmarks, respectively.

Categories and Subject Descriptors: I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; C.4 [**Performance of Systems**]: Modeling Techniques

General Terms: Measurement, Experimentation

Additional Key Words and Phrases: Design space exploration, Fermi architecture, simulation, validation

ACM Reference Format:
Jieun Lim, Nagesh B. Lakshminarayana, Hyesoon Kim, William Song, Sudhakar Yalamanchili, and Wonyong Sung. 2014. Power modeling for GPU architectures using McPAT. ACM Trans. Des. Autom. Electron. Syst. 19, 3, Article 26 (June 2014), 24 pages.
DOI: http://dx.doi.org/10.1145/2611758

1. INTRODUCTION

Graphics Processing Units (GPUs) have become increasingly popular. They are used in mobile devices, desktops, and servers as well as in supercomputers for high-performance computing. GPUs have wide single-instruction, multiple-data (SIMD) units, and simplified microarchitecture dedicating most transistor budget to floating-point operations, which are the essential component of applications for graphics, science, engineering, and other various domains.

This research was supported in part by the National Science Foundation under grant CCF 1054830, CNS 085511, the Semiconductor Research Corportation (Task ID# 2084.001) and Sandia National Laboratories. Jieun Lim and Wonyong Sung were partially supported by Brain Korea 21 Project and the National Research Foundation of Korea under Grant 2012R1A2A2A06047297. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NSF, Sandia Lab or SRC.
Authors' addresses: J. Lim, School of Electrical Engineering, Seoul National University San 56-1, Shilim-Dong, Kwanak-Gu, Seoul 151-742, South Korea; N. Lakshminarayana, H. Kim (corresponding author), W. Song, and S. Yalamanchili, Georgia Institute of Technology, Atlanta, GA 30332; email: hyesoon@cc.gat.edu; W. Sung, School of Electrical Engineering, Seoul National University San 56-1, Shilim-Dong, Kwanak-Gu, Seoul 151-742, South Korea.

DOI: http://dx.doi.org/10.1145/2611758

Power is one of the key design constraints for computing systems, and processors (i.e., CPUs, GPUs, and other processing units) consume a significant portion of total system power [Bircher and John 2012]. In the past couple of decades, great effort has been put into developing useful CPU power models such as Wattch [Brooks et al. 2000] and McPAT [Li et al. 2009], which brought about a surge of power-related research in the computer architecture community. However, relatively few works are found regarding GPU power modeling and analyses.

The difficulty of modeling and estimating GPU power is due to the following reasons. First, insufficient information about the GPU microarchitecture and implementation is known to the general research community. The GPU microarchitecture is substantially different from CPU architectures, where the GPUs include new components such as shared memory (scratchpad memory) and texture cache. GPUs specialize some components such as register files and functional units but simplify other components such as the instruction scheduler. The primary reason for the popularity of CPU power models such as Wattch, CACTI [Muralimanohart et al. 2007], and McPAT lies in the fact that they are configurable to explore several different microarchitectural designs. On the other hand, few proposed GPU power models in the past were based on statistical estimation and empirical measurement from specific GPU architectures [Hong and Kim 2010; Choi et al. 2013; Goswami et al. 2013], which are inadequate for generic design space exploration.

The contribution of our work is twofold: first, the GPU power model itself, and second and more importantly the method to construct a GPU power model using both empirical data and low-level power models such as McPAT. Please note that as a concurrent study, GPUWattch has been recently presented by Leng et al. [2013], which is also based on McPAT. The main difference between our work and GPUWattch is in the providing the details of power model construction methods.

We built an initial model and configured it to predict the power consumption for a NVIDIA GTX580 GPU. In this process, we utilized parameters available from NVIDIA and research publications while making educated choices of other unknown microarchitectural parameters. In addition, we tested several McPAT design parameters and models that were derived from empirical measurement of CPUs. The parameter adjustments were made by comparing estimations from the model for microbenchmarks with measured data from a GTX580 GPU running the same microbenchmarks. The models were also updated with adjusted parameters, and this process was repeated until both power models and design parameters converged to measured data of a GTX580. The finalized model was validated against other general GPU benchmarks; we present various experimental results in this work to provide insight to the nature of GPU power consumption.

The rest of the article is organized as follows. Section 2 provides background on McPAT and the GPU architecture. Section 3 introduces our methodology and the benchmarks used. We describe the modeling process that evaluates various possible configurations and adjusts McPAT parameters in Section 4. Section 5 presents the validation results and discusses experimental results. We conclude this work in Section 7.

2. BACKGROUND

2.1. McPAT and Introspection Interface

In this work, we estimate GPU power consumption using McPAT [Li et al. 2009], an architecture-level power modeling tool for CPUs. McPAT models a processor with a CPU architecture hierarchy comprised of cores, shared caches, networks, memory controllers, and other I/O controllers. Each component is broken into subcomponents such as pipeline stages. For instance, a core is composed of five pipeline stages, and each

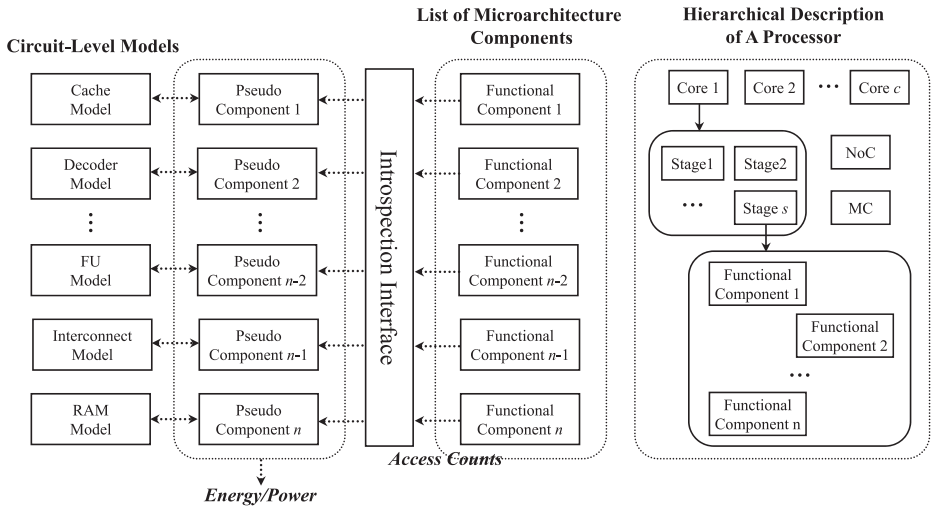

Fig. 1. Simulation framework for GPU power modeling through the introspection interface (left) and hierarchical description of CPU architecture in McPAT (right).

pipeline stage includes architectural elements such as an instruction decoder and data cache. Those architectural elements are the bottom-most components in the CPU architecture hierarchy defined in McPAT. The architectural elements are associated with appropriate circuit-level models. For example, instruction and data caches, translation lookaside buffers (TLBs), and branch target buffer (BTB) are all cache models with different configurations. Thus, we note that McPAT is essentially a model library that is a collection of various circuit-level models that can be rearranged to configure different microarchitectures.

We use a simulation interface [Song et al. 2012] to rearrange the circuit-level models into a GPU architecture. Figure 1 depicts the idea of configuring a microarchitecture via the introspection interface. The microarchitecture is viewed as a list of functional components instead of using a hierarchical description. Therefore, the simulation interface does not model upper-level components (e.g., cores, pipeline stages) in the hierarchy that have no effects on the power result. The introspection interface creates *pseudo components* [Song et al. 2012] that are the counterparts of the microarchitecture components whose power would be estimated. Each pseudo component is identified by a unique ID. The pseudo components are the abstract entities in the simulation environment to represent matching microarchitecture components. The introspection interface collects statistical data (i.e., access counters for each microarchitecture component) from timing/functional simulators. The interface differentiates four types of access counts for each component: *read, write, tag-read, tag-write* accesses. Read and write are typical read or write accesses to storage units such as arrays, buffer, caches, etc. When tag arrays are present in the modeled components, tag-read/write accesses are used to represent tag-array-only accesses. For logical components such as execution units whose access types are ambiguous, either read or write counts are used to indicate an architectural usage of the component. The energy consumption at a component is mainly broken into dynamic and leakage dissipations. The dynamic energy of a component is represented as the sum of energies spent on different types of accesses, and the energy of each access type is calculated as the product of access count and per-access energy. This calculation does not take the number of switching bits into account to calculate dynamic energy, but typical architecture simulations also do

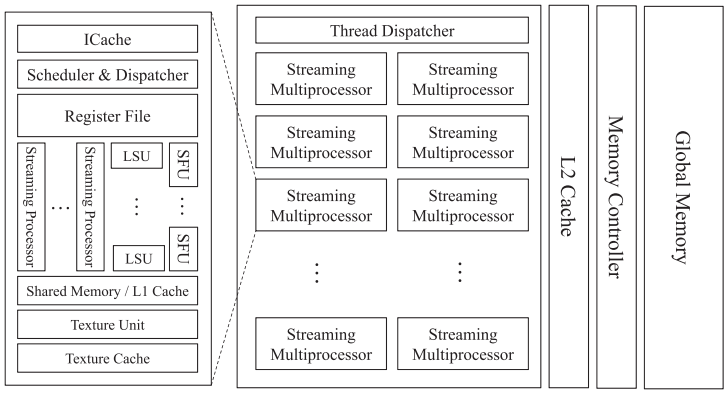

Fig. 2. NVIDIA's Fermi architecture. (Memory controller and global memory are off-chip components.)

not capture bit-level activities due to simulation complexity. The access counts can be acquired from functional simulations, and the energy per access can be estimated from circuit-level models linked via pseudo components. Clock frequency and execution time are used to convert the energy to power, and the total processor power is represented as the sum of the power of all modeled components.

2.2. GPU Architecture

In this section, we describe the GPU architecture used for the power modeling and compare it with CPU cases. As shown in Figure 2, a GPU consists of several cores called streaming multiprocessors (SMs), each of which performs multithreaded execution of warps. A warp is a group of 32 threads that execute in lockstep. A thread dispatcher located outside the SMs is responsible for assigning threads to SMs at block granularity. In every fetch cycle, the dispatcher unit in an SM fetches an instruction for one or more selected warps. The instruction is then decoded, and its readiness for execution is tracked using a scoreboard. Among the ready instructions, the scheduler selects one or more instructions and issues them for executions. Although recent CPUs also contain multiple cores with multithreaded executions, the degree of multithreading is typically between two to eight and much less than GPUs. CPUs primarily target the serialized executions of threads and dedicate considerable hardware resources to improve the performance by deploying complex architectural techniques and components such as branch prediction, register renaming, out-of-order execution, etc. On the other hand, GPUs do not exploit these mechanisms and perform simpler in-order execution.

An SM consists of multiple functional units of three types: streaming processor (SP), special functional unit (SFU), and load/store unit (LSU). In the Fermi architecture [NVIDIA 2009], each SM has 32 SPs that execute floating-point and integer instructions such as ADD, SUB, MAD, and so on. There are four SFUs in the Fermi to execute complex functions such as sine, cosine, and reciprocal. The LSUs are used for memory load and store instructions. A coalescing unit in LSUs is used to reduce the number of requests sent out to the memory system. The register file system is heavily banked, and the source operands of instructions are read before starting executions. The SPs, SFUs, and LSUs all operate in an SIMD mode. On the contrary, CPUs contain fewer functional units that mostly operate on scalars. Memory instructions are also scalar, and thus CPUs do not require coalescing units.[1] The execution width of

[1]CPUs with a wide vector width also have coalescing units.

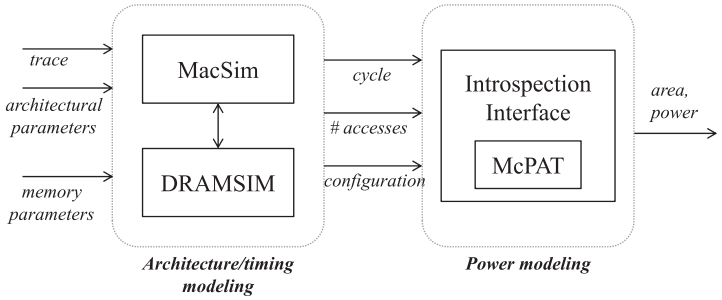

Fig. 3. Overview of the modeling framework.

SIMD units in CPUs is smaller than in GPUs, and register files are also significantly smaller and support only a few threads.

An SM includes a constant cache and a texture cache. A local scratchpad memory called shared memory is also available in each SM. Shared memory can be used for exchanging data between the threads within a block, while it also serves as low-latency memory. GPUs are distinguished from CPUs by a large degree of multithreading. GPU cores are kept busy by switching warps and thus are less reliant on caches, while CPUs dedicate more resources to caches to improve performance.

3. METHODOLOGY

3.1. Simulation Methodology

Figure 3 shows the simulation framework for timing, area, and power modeling. It consists of two major software component interactions: architecture/timing and power simulators. The architecture/timing simulator is comprised of MacSim [MacSim 2012] and DRAMSIM2 [Rosenfeld et al. 2011]. MacSim, a trace-driven and cycle-level simulator, models the behavior of the microarchitecture, tracks timing information, and interacts with DRAMSIM2, which simulates GPU device memory. The main memory is modeled as GDDR5 SGRAM with parameters obtained from the JEDEC standard [JEDEC 2014] and Hynix [Hynix 2006]. The power consumption of the main memory is directly computed from DRAMSIM2. The MacSim generates the access counts of all architectural components in the GPU to compute the power consumption via the introspection interface. The circuit-level models in McPAT that are reorganized by the introspection interface in accordance with GPU microarchitecture are used to estimate area and power dissipation.

3.2. Methodology for Measuring Empirical Power

The validation of the proposed model was performed by measuring the power of NVIDIA GTX580 with the Fermi architecture. The hardware specifications of NVIDIA GTX580 are listed in Table I. The Extech 380801 AC/DC Power Analyzer [Extech 2014] is used for measuring the power. We first measure the power consumption of the entire system when a GPU application is running (P_a). We also identify the idle power of the entire system (P_b). We then measure the CPU idle power dissipation by turning off the GPU modules and rendering the CPU into an idle state (P_c). With the GPU modules still off, we measure the power consumption when running the GPU application with the CUDA-related function calls removed (P_d); this value gives the CPU power. Consequently, P_b-P_c is the GPU idle power and P_a-P_d-(P_b-P_c) equals the GPU runtime power. Please note that the idle power and the static power are different. During idle time, some components are still active even though they are not doing any useful work. If

Table I. Hardware Configuration of the Modeled GPU Architecture (NVIDIA GTX580)

Model	GTX580
Architecture	Fermi
Clock freq.	1.544GHz
Feature size	40nm
Thermal design power (TDP)	244W
#SM	16
#SPs, #SFUs, #LSU/SM	32, 4, 16
#registers/SM	32,768
L1 cache	48KB
L2 cache	768KB
Shared memory	16KB
Constant cache	8KB
Texture cache	8KB
Memory bandwidth	192.4GB/s

the system has a perfect clock gating, idle power would be similar to static power, but this is not the case for the evaluated systems.

3.3. Benchmarks

We also used a set of benchmarks that are similar to those used in the previous GPU power model paper [Hong and Kim 2010], namely, the microbenchmarks and the Merge benchmarks [Linderman et al. 2008], to validate and simulate the proposed model. The characteristics of all the benchmarks are detailed in Table II. Note that misses per kilo-instructions (MPKI) represents the degree of memory intensiveness. The microbenchmarks are a set of synthetic kernels with loops that heavily access specific hardware units. Since the access patterns of microbenchmarks are easily characterizable, seven microbenchmarks are used to evaluate McPAT parameters. The rest of the microbenchmarks and the Merge benchmarks are employed for prediction benchmarks. The microbenchmarks for prediction are more complicated than the training microbenchmarks. The Merge benchmarks represent real-world kernels.

4. GPU ARCHITECTURE POWER MODELING

This section explains the process of building the GPU power model, as shown in Figure 4. Our approach is a combination of empirical modeling and McPAT's analytical modeling. The major difficulty of modeling the GPU power is lack of detailed information about GPUs such as hardware structure of SFUs and the detailed cell information of register files. In addition, McPAT has to be modified to reflect GPU designs. Thus, the following steps are iterated to find appropriate McPAT parameters to represent GPU implementation. Step 1 specifies known design parameters of McPAT according to the GPU architecture. In step 2, unknown design parameters such as the number of EXUs in SFUs and the number of ports in L1 cache are determined by testing different configurations and selecting the minimal-error designs compared with reference data from measurement and other published documents explained in the next sections. In the final step, the McPAT configuration and parameters are adjusted based on the results in the previous steps. Steps 2 and 3 are iterated so that the result converges to the empirically measured data.

4.1. Describing GPU Components in McPAT Format (step 1)

The configurable input parameters to McPAT are largely composed of general technology parameters (i.e., transistor size, clock frequency, etc.) and architecture

Table II. Characteristics of the Benchmarks

Micro	Description	MPKI[a]	Purpose
fp	FMAD operations	0.013	training
int	Integer Multiplication	0.017	training
const	Access Constant Memory	0.122	training
mb10same	Series of single dependent load, accesses L1	0.111	training
mb11same	Series of 4 dependent loads, accesses L1	0.411	training
mb12same	Series of 8 dependent loads, accesses both L1 and L2	0.817	training
mb14same	Series of 8 dependent loads, accesses both L1 and L2	0.817	training
shared	Access Shared Memory	0.013	prediction
mb11diff	Series of 4 dependent loads, mainly accesses memory	11.00	prediction
mb12diff	Series of 8 dependent loads, mainly accesses memory	10.11	prediction
mb14diff	Series of 8 dependent loads, access L1, L2, and memory	9.226	prediction
dotp	Matrix dot product	5.026	prediction
dmadd	Matrix double memory multiply add	17.89	prediction
madd	Matrix multiply-add	19.89	prediction
mmul	Matrix single multiply	19.93	prediction
cmem	Matrix add FP operations	0.751	prediction
Merge	Description	Type	Purpose
Binomial	Option pricing using Binomial algorithm	0.713	prediction
Blackscholes	Option pricing using BlackScholes algorithm	0.654	prediction
Convolve	2D Separable image convolution	11.57	prediction
Nmat	Naive matrix multiplication	6.751	prediction
Sepia	Filter to artificially age image	25.94	prediction
SVM	SVM-based face classifier	1.275	prediction

[a]MPKI: misses per kilo-instruction.

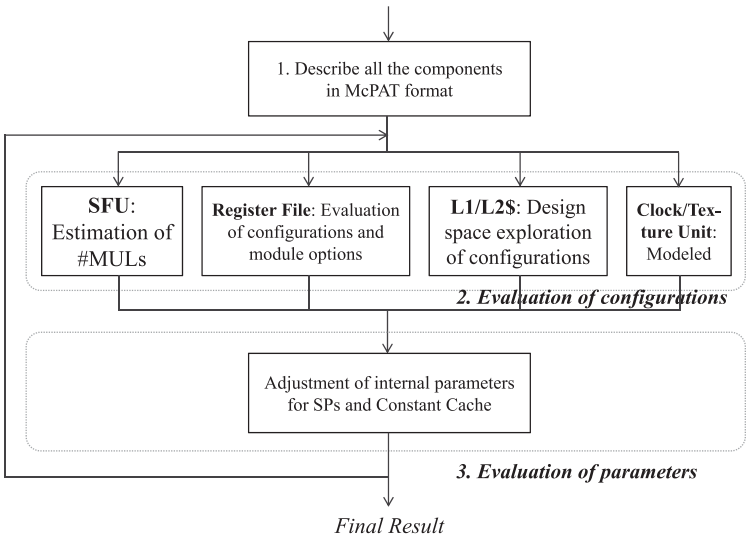

Fig. 4. Process of modeling GPU power.

component-level values such as cache-line size, decoder width, etc. These McPAT parameters are dynamically adjustable through the simulation interface. Table III lists the input technology parameters used by McPAT to define device-level characteristics of the chip and the values assigned to these parameters in this work. Several

Table III. McPAT Design Parameters

Parameter	Available Options	GPU	Xeon in McPAT
clock_frequency	in unit of Hz	1.544e9	3.4e9
feature_size	16nm to 180nm	40e-9	65e-9
core_type	Out-of-Order, in-order	in-order	Out-of-Order
embedded	true, false	false	false
wire_type	global, global_5, global_10, global_20, global_30, low_swing, semi_global, transmission, optical	global	global
device_type	hp[a], lstp[b], lop[c]	hp	hp
interconnect_projection	aggressive, conservative	aggressive	aggressive
wiring_type	local, semi_global, global	global	global
component_type	core, llc[d], uncore[e]	core, llc	core, llc
opt_local	true, false	false	false
longer_channel_device	true, false	true	true

The parameters in the available options are the parameters in McPAT.
[a]hp: high-performance type.
[b]lstp: low standby power type.
[c]lop: low operating power type.
[d]llc: last-level cache.
[e]uncore:any logic except for core and LLC.

Table IV. Description of Modeled GPU Hardware Structure

Hardware structure	Model type	Model parameters
Block/Warp States, Fetch Queue, Instruction Queue, Register File	array (RAM)	input line width, output line width, associativity, #banks, #entries, tag width, #ports(R, W, RW), cycle time, access time
Instruction TLB, Instruction Cache, Scoreboard, Data TLB, L1 Cache, L2 Cache, Cache Buffers, Shared Memory, Constant Cache, Texture Cache	array (Cache)	input line width, output line width, associativity, #banks, #entries, tag width, #ports(R, W, RW), cycle time, access time
Instruction Decoder	instruction_decoder	decoded opcode width
Instruction Issue Selection Logic	selection_logic	selection input size, selection output size
SP, SFU, LD/ST units	functional_unit	-
Memory Controller	memory_controller	buffer line size, request window entries, I/O buffer entries, #memory channels, peak transfer rate, #ranks, data bus width, address bus width
NoC	network	router or bus, flit bits, #ports (in, out), #virtual channels, duty cycle, link throughput, link latency, chip coverage, percentage of pipelining
Pipeline Latches	pipeline	pipeline stages, width

Model parameters are the parameters in McPAT.

known parameters (Table I) such as clock_frequency, feature_size, core_type, and component_type are defined as the target GPU model, but the rest of the parameter values are left the same as for Intel Xeon, which is the latest processor model provided by McPAT. Table IV summarizes all the hardware structures modeled for the GPU architecture along with the model types used and the module parameters supported for each structure. The first column in Table IV lists all the functional components that

Table V. Description of Architectural Differences between CPUs and GPUs

Module name	GPU-specific characteristics
Dispatcher outside SMs	Not modeled
Block/Warp states	Modeled by array (ram) type (8B/warp)
Branch Predictor, Branch Target Buffer, Return Address Stack	No branch predictor modules
Decode	No pre-decoder, sequencer
Scheduler	Scoreboard, output signal * #SP_per_SM
EX - SP	1 ALU + 1 FPU (determined in Step 3)
EX - SFU	6 MULs (determined in Step 2)
Register file	1r1w/bank, 32 banks
Shared memory	newly added module, 16/48 KB per SM
Constant cache	newly added module, 8KB per SM
Texture cache	newly added module, 8KB per SM
TPC Texture unit	leakage is included in the new parameter for the clock network

are assumed to be in the target GPU. Each module is described by using circuit-level model type and model parameters, as explained in Section 2.1.

Table V shows the list of distinctive GPU components. A global thread block dispatcher is not modeled in this work since it is comparably smaller than other components and it is rarely executed, such as 1 out of 1 million cycles.[2] Also, CPU-specific components such as branch predictors, instruction pre-decoder, and micro-op sequencer are disregarded. An instruction scheduler in an SM has an extended width of control and output signals to support SIMT execution features. The output signal widths are equal to the number of SPs connected to scheduler units. Extra GPU-specific components such as shared memory and texture cache are newly added since the original McPAT does not have them.

4.2. Choosing and Evaluating Configurations (step 2)

GPUs have a distinct architecture compared to CPUs, and detailed microarchitectural information is not exposed. Therefore, several possible configurations are tested to identify unknown architectural parameters such as the number of execution units for modeling SFUs and the number of ports and banks for describing the register file and the caches. Although the number of execution units and SFU units can be inferred from the performance, modeling those numbers using McPAT is not straightforward. McPAT itself also used empirical power data to model execution units because execution units are heavily dependent on circuit designs. The following shows the identified configurations of SFUs, register file, L1 and L2 caches, clock network, and texture unit. Please note that some parameters do not necessarily correspond to the actual number in the modeled GPU system such as the number of execution units. These numbers are the modeled numbers that result in power numbers from empirical data.

4.2.1. Special Functional Units (SFUs). The Fermi architecture has four SFUs per SM for the efficient execution of special functions. Special functions are also one of the highly customized structures similar to execution units. SFU is beyond a collection of simple lookup tables [Lindholm et al. 2008]. Since it is even harder to find the detailed hardware structure information about the SFU, we use multipliers (MUL type

[2]The thread block dispatcher is executed when a thread block is finished.

Table VI. Parameters to Estimate the Number of MULs for SFU (Equations (2)–(5))

Parameter	Value
#SP_per_SM_GTX280	8
#SFU_per_SM_GTX280	2
$\text{PeakPower}_{\text{FP\_GTX280}}$	0.2 [Hong and Kim 2010]
$\text{PeakPower}_{\text{ALU\_GTX280}}$	0.2 [Hong and Kim 2010]
$\text{PeakPower}_{\text{INT\_GTX280}}$	0.25 [Hong and Kim 2010]
$\text{PeakPower}_{\text{SFU\_GTX280}}$	0.5 [Hong and Kim 2010]

Table VII. Peak Power of the Execution Units from McPAT

Component	Peak Power (mW)
ALU	145
MUL	309
FPU	504

in McPAT) to model the SFU and search for the most reasonable number based on empirical data.

The basic method to estimate the number of SFU units is to isolate the SFU power consumption from empirical data. However, instead of naively matching the absolute SFU power values, we measure the relative power ratio of an SP to an SFU in the empirical data and search for the appropriate number of SFU units, since matching the absolute values can easily skew the results. Hence, as shown in Eq. (1), it is assumed that the peak power ratio of an SP to an SFU modeled with McPAT is the same as that from the empirical data. In these equations, *PP* denotes *Peak Power* in Hong and Kim's work [2010]. Note that the numbers of SPs and SFUs in GTX280 are different from those in GTX580; thus we use per-unit peak powers (i.e., per-SP, per-SFU) to equally compare the units in GTX280 and GTX580. Each term in Eq. (1) can be computed as Eqs. (2)–(5). Eqs. (2) and (3) compute the peak powers for FPU and ALU using the McPAT model. Eqs. (4) and (5) compute the empirical peak powers. These equations are used to compute $\#MUL\_per\_SFU$.

All the parameters for the empirical data are selected from GTX280 specifications [NVIDIA 2014a] and are listed in Table VI.

After substituting Eqs. (2)–(5) by Eq. (1), the number of MULs to model an SFU can be calculated. Based on the reference values in Table VI and the peak power of each execution unit of McPAT in Table VII, we found that six MULs are suitable to model an SFU. The total number of MULs used to model an SFU is a bit more than the count mentioned by NVIDIA [Lindholm et al. 2008] due to the discrepancy of execution unit models between NVIDIA implementation and McPAT models. Note that the execution unit models in McPAT are based on the empirical data of embedded processors from Sun Microsystems [Leon et al. 2006].

$$PP_{SP\_model} : PP_{SFU\_model} = PP_{SP\_emp} : PP_{SFU\_emp} \tag{1}$$

$$PP_{SP\_model} = PP_{FPU\_McPAT} + PP_{ALU\_McPAT} \tag{2}$$

$$PP_{SFU\_model} = \#MUL\_per\_SFU \times PP_{MUL\_McPAT} \tag{3}$$

$$PP_{SP\_emp} = \frac{PP_{FP\_GTX280} + PP_{ALU\_GTX280} + PP_{INT\_GTX280}}{\#SP\_per\_SM\_GTX280} \tag{4}$$

$$PP_{SFU\_emp} = \frac{PP_{SFU\_GTX280}}{\#SFU\_per\_SM\_GTX280} \tag{5}$$

Table VIII. Possible Register File Configurations for a 128-Bit Access

Parameter	Gebhart's work[a]	hp			lstp			lop		
		1r/1w	2r/1w	3r/1w	1r/1w	2r/1w	3r/1w	1r/1w	2r/1w	3r/1w
bank area (μm^2)	38,000	57,883	107,615	149,087	98,968	187,081	248,353	98,960	187,104	248,759
unit_energy. read (pJ)	8	5.1	12.8	19.7	10.7	27.9	42.1	3.8	9.9	14.9
unit_energy. write (pJ)	11	5.4	13.2	20.1	10.8	27.9	42.2	3.8	9.9	14.9
unit_energy. leakage (pJ)	-	4.1	7.5	10.1	0.002	0.003	0.004	0.7	1.3	1.7

[a][Gebhart et al. 2011].

4.2.2. Register File. The register files in GPUs differ from those in CPUs in that there are tens of thousands of registers to hold the register state of all the threads running on an SM. Moreover, the register file has to support many concurrent accesses (reads and writes by different warps in the same cycle). This heavily accessed register file can be implemented using 32-bank dual-ported RAM blocks (1 read and 1 write) as explained by Gebhart et al. [2011]. Various port configurations, including the dual-port architecture described before, are explored along with different device-type options for the purpose of verifying the configuration recommended in Gebhart et al. [2011] with McPAT parameters shown in Table VIII. As can be seen in the table, configurations having more than 1r/1w ports consume very large area when compared to the reference data from Gebhart et al.'s paper. For the configuration employing 1r/1w ports, the lstp option provides unit energy values that are quite similar to those of reference data, but small leakage energy does not seem to be reasonable because the register file is expected to have fast and large hardware dissipating a certain amount of leakage power. Therefore, based on the data in Table VIII, we choose the 1r/1w port configuration with the hp (high-performance) option, which produces reasonable results when considering both area and energy consumption.

4.2.3. L1 and L2 Caches. In order to model caches with the models provided by McPAT, we need to know configuration details such as the size, the number of banks, and the number of ports. However, only the sizes of the various caches are known and no other configuration information is available. Therefore, we explore various configurations to find a combination that minimizes the error between the modeled and the measured total power. First, four configurations for the L1 cache and eight configurations for the L2 cache, which have either 1r/1w or 2r/2w ports and consist of one to eight banks, are considered. We then measure the total power of all the possible combinations of the configurations of the two caches for the `mb10same`, `mb11same`, `mb12same`, and `mb14same` benchmarks and normalize them to the measured power, as shown in Figures 5 and 6. As a result, two combinations show less than 5% errors for all the benchmarks, as highlighted in Figure 6. In order to choose between the two candidates with low error, we consider the area of an SM, which is known to be $16mm^2$ [Gebhart et al. 2011], and the throughput of the L2 cache. The area of the L1 cache (listed in Table IX) has to be smaller than the area of one SM, and the L2 cache should have enough ports or banks to support concurrent requests from all the SMs. Based on these criteria, we choose the configuration employing 1r/1w port and two banks for the L1 cache and 2r/2w ports and eight banks for the L2 cache. This final configuration results in a smaller L1 cache area and a higher L2 cache throughput. Please note that the timing simulator already models memory request coalescing units to aggregate requests from all execution units within an SM and broadcasting units that send a memory request

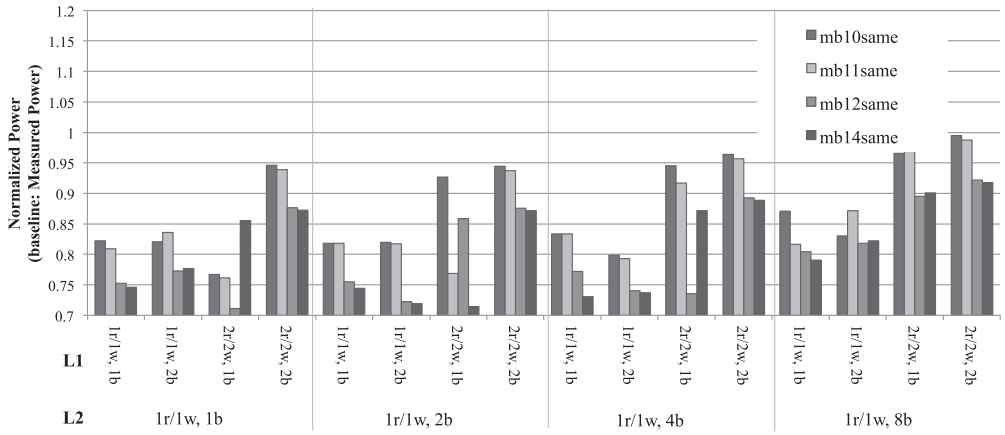

Fig. 5. Normalized total power for various L1 and L2 configurations.

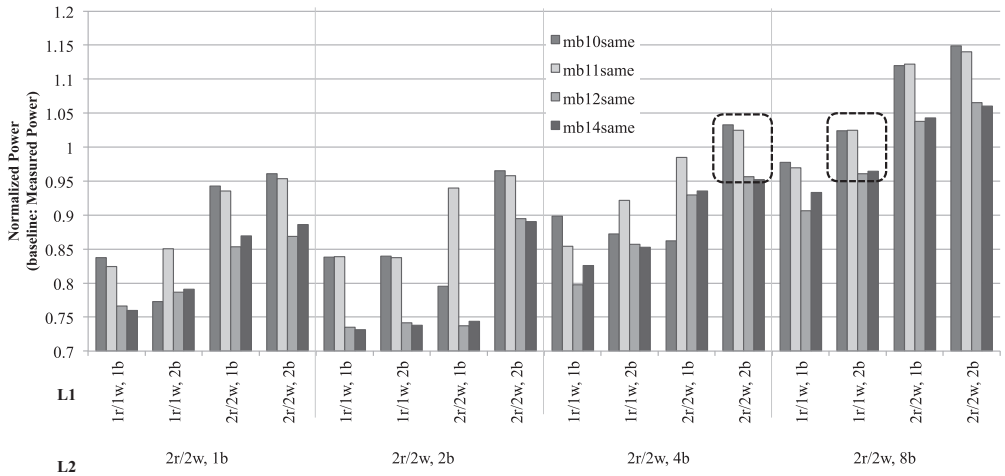

Fig. 6. Normalized total power for various L1 and L2 configurations.

Table IX. Estimated L1 Area for Various Cache Configurations

Configuration	L1 area per SM (mm^2)
1r/1w, 1banks	3.194
1r/1w, 2banks	7.754
2r/2w, 2banks	10.235
2r/2w, 2banks	45.035

to individual threads. This is the main reason to have fewer read and write ports for caches.

4.2.4. Clock and Texture Unit. The power dissipation on the clock network contributes to a considerable fraction of the total power. For example, the power consumptions of the clock network reported by Wattch and McPAT are 8%, 34%, 16%, and 20% of the total power for Pentium Pro, Alpha 21264, Alpha 21364, and Niagara2, respectively [Brooks et al. 2000; Li et al. 2009]. However, the publicly available version of McPAT does not

support a separate clock network model. To model core gating (i.e., applications use only a few number of cores and the remaining cores become idle/power gated), we also need to model the clock power.

Modeling leakage power for the texture unit is critical in the GPGPU power model because of the unique characteristics of texture units. Texture units mainly work on the graphics applications, but signals from the EXUs to the texture cache go through the texture unit and dissipate a small amount of the leakage power [NVIDIA 2014c]. Although GTX580 seems to employ an aggressive power-gating technique, which will be described later, the texture cache still incurs a leakage cost for graphics processing and is accessed by general-purpose workloads as well. Wattch suggests that the power cost of the clock network be at least 8% of the total power, and we need to consider the texture unit as well. Therefore, we assume that 10% of the thermal design power (TDP)[3] of GTX580 [NVIDIA 2014b] is consumed by the clock network and the texture unit, which corresponds to 24.4W and approximately 15% of the measured total power. Accordingly, this constant value for modeling the clock power and texture unit consumption is included in all of our results.

4.3. Evaluation of Parameters (step 3)

After determining the configurations for all the components, we adjust the internal McPAT parameters such as the power ratio between ALUs and FPUs and the scaling factor for the constant cache to make our power model suitable for the GPU architecture.

Eq. (6) represents the total power as a sum of the power consumption of all the components. In order to focus on the modules that need to be adjusted, in this section we decompose a GPU into the evaluated components, such as SPs and the constant cache, and the others, as shown in Eq. (7). Since one approach is applied separately to the dynamic and the leakage power of the EXUs, Eq. (7) is more detailed in Eq. (8). Finally, Eq. (9) represents the adjusted total power with the scaling factors α, β, γ, δ, and ϵ and will be discussed next.

$$\begin{aligned} TotalPower &= \sum P_{component} && (6)\\ &= P_{SP\_fpu} + P_{SP\_alu} + P_{ConstMem} + P_{Others} && (7)\\ &= P_{SP\_fpu\_dyn} + P_{SP\_fpu\_lkg} + P_{SP\_alu\_dyn} + P_{SP\_alu\_lkg} && (8)\\ &\quad + P_{ConstMem} + P_{Others} \end{aligned}$$

$$\begin{aligned} Total\ Power_{adjusted} = {} & \alpha \cdot P_{SP\_fpu\_dyn} + \beta \cdot P_{SP\_fpu\_lkg} + \gamma \cdot P_{SP\_alu\_dyn} \\ & + \delta \cdot P_{SP\_alu\_lkg} + \epsilon \cdot P_{ConstMem} + P_{Others} \end{aligned} \quad (9)$$

The SP, which is the main execution unit that performs arithmetic operations, contains functional units. NVIDIA [Lindholm et al. 2008; NVIDIA 2009] explains that each SP includes a scalar multiply-add (MAD) unit in Tesla, and one ALU and one FPU in Fermi. Because McPAT provides only fixed-width ALU, MUL, and FPU types for EXUs, we model one SP using one ALU and one FPU. However, using the McPAT types without any modifications may not be suitable for GPUs because EXUs are highly customized hardware with their structure varying from one vendor to another. Moreover, NVIDIA GPUs have many execution lanes, from 8 to 32, meaning a high possibility of hardware optimization. The estimated SP area provides evidence that the EXUs in GPUs are considerably different from those of McPAT. The area of 32 SPs in an SM modeled with the original EXU types of McPAT is $61mm^2$, whereas the estimated area of an SM from

[3]TDP is the maximum power that can be dissipated by a device when applications are running.

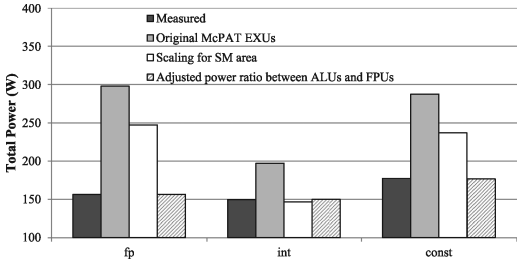

Fig. 7. Comparison of various configurations for the training benchmarks.

available die shots is approximately $16mm^2$ [Gebhart et al. 2011]. This overestimation is largely due to the modeling method employed by McPAT, which is based on empirical models using published data [Mathew et al. 2005; Leon et al. 2006]. McPAT takes the area of EXUs from actual designs by Intel [Mathew et al. 2005] and Sun [Leon et al. 2006], estimates leakage power consumption to be proportional to the area, and scales the power values to different technologies. Figure 7 shows the total power of all the possible implementations modeled with various parameter values. The result labeled "Original McPAT EXUs" shows the estimated power consumption when the original EXUs of McPAT are used without any modifications. The light-gray bars show much larger power variation among benchmarks and a considerably different trend compared to the measured data, supporting our claim that EXUs need to be adjusted.

In order to enhance the approach of McPAT, we reduce the area of EXUs while maintaining the size ratio of all the units until the total SM area becomes around $16mm^2$. Because in McPAT's model, a leakage power is dependent on an area; this approach only reduces the scaling factors (β and δ in Eq. (9)) for the leakage power of the EXUs. After this modification, the total power consumption shown by the result labeled "Scaling for SM area" in Figure 7 decreases due to the reduced leakage power. However, the trend of the estimated results still does not match well with measured data because reducing the area does not affect the dynamic power, which is related to the variations among the power consumption of the applications. The other possible source of error is the power ratio of ALUs to FPUs in McPAT's modeling. We can see this problem by comparing the total power of `int` and `fp` benchmarks, which have the same behavior except for accessing the integer or the floating-point part of the SP unit. In Figure 7, the gap between the two benchmarks for the result labeled "original McPAT EXUs" and "Scaling for SM area" is much larger than that of the real data because the FPUs in McPAT are modeled with much larger parameters than the ALUs, and this might overestimate the FPU power for the GPU hardware. To adjust this overestimation, assuming ALUs in McPAT are modeled correctly, we reduce the power ratio between ALUs and FPUs until the power difference of the two benchmarks is close to that from the measured data. This adjustment only modifies the scaling factors for FPUs, α and β in Eq. (9). Note that we assume the difference in the measured data for the two benchmarks is due to the difference in the power consumption of FPUs and ALUs and is not due to the use of a specific instruction such as add.cc instead of another instruction such as addc (or vice versa)[4]. As can be seen in Figure 7, the result labeled "Adjusted power ratio between ALUs and FPUs" gives the closest results to the measured data.

Since constant memory is only used by the `const` benchmark, scaling for the constant cache is performed after EXUs are tested. In other words, ϵ in Eq. (9) is adjusted until the estimated power is the same as the measured power for the `const` benchmark.

[4]add.cc:add two values with carry-out, addc:add two values with carry-in and optional carry-out.

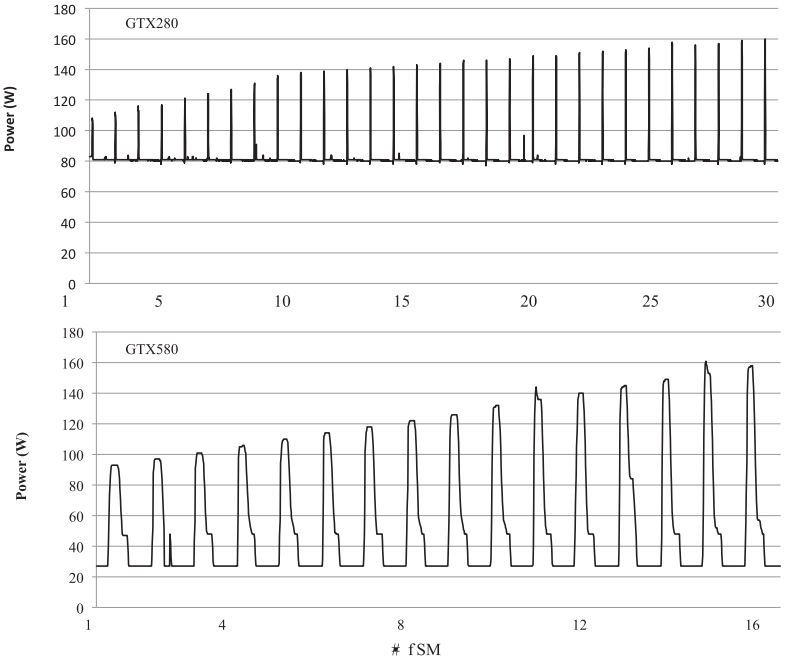

Fig. 8. Total power vs. varying the number of active SMs.

4.4. Introducing Activation Power

In this section, we introduce an additional term that is not modeled in McPAT. This is just a conceptual term to model the constant increase in power consumption shown in the empirical data. Figure 8 shows the total power consumption when the number of active SMs grows. We measure the idle and the runtime power by repeating a process that turns on a few cores and then put the cores into an idle state to cool down the system, as indicated in Hong and Kim [2010]. Although the idle power is greater than the leakage power, we used leakage power outcomes from the power model for first order comparisons because we cannot exactly measure the leakage power. For GTX580, the real GPU consumes only 27W in the idle state, and Xbitlabs [Stepin and Lyssenko 2014] also measured the idle power as 26.5W. However, our model predicts that the leakage power of GTX580 is 100W. On the other hand, for GTX280[5], the measured idle power is 83W and our leakage estimate is 94W, which is an error of 13%. While our model can guess the idle power of GTX280 with an acceptable error by estimating the leakage power of the system, there is a huge gap between the measured idle power and modeled leakage power of GTX580. We consider that this gap is caused by the power-gating technology applied to GTX580, which is not modeled in McPAT but significantly affects the power dissipation on GTX580. Since GTX580 is known to be more power efficient and to have more power control, we expect that only active SMs are turned on and the unused SMs are turned off. When we activate the first SM, the power consumption of the GPU system in Figure 8 becomes 92W, which is 65W higher than the idle power of 27W. To model this, we introduce an additional term of 65W, called *activation power*. We can use this term when predicting the actual idle or the runtime

[5]The focus of this article is GTX580 but just for the idle power comparisons, we also compare GTX280.

Table X. Summary of Changes in McPAT to Model GPUs

Unit	Changes	Process
Activation power	65W estimated	Experimental
Clock & Texture unit	24.4W modeled	Experimental
SP	Power adjustment between ALUs and FPUs (α, β: 0.27, γ, δ:1)	Experimental
SFU	Model using six MULs	Experimental
Register file	Design space exploration for the port configuration (1r/1w with hp option)	Design space exploration
L1/L2 caches	Design space exploration for the port and bank configuration (L1: 1r/1w ports and two banks, L2:2r/2w ports and eight banks)	Design space exploration
Constant cache	Scaling (ϵ:5.4)	Experimental

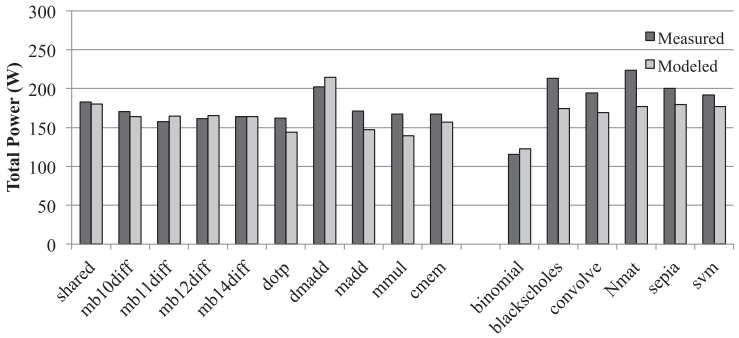

Fig. 9. Total power for the prediction benchmarks.

power. In other words, when there is no active core, the idle power is 65W less than the leakage power of our model.

The summary of changes for a more consistent match between the model and the measured data is listed in Table X.

5. EVALUATION

5.1. Validation and Discussion

Sixteen workloads are used for the prediction benchmarks to validate the correctness of the power model: ten from the microbenchmarks and six Merge benchmarks. Figure 9 compares the output of our model against the measured total power for the prediction benchmarks. It shows that the power estimates for the microbenchmarks correlate very well with the measured data with satisfactory accuracy—the geometric mean of the error is 7.7%. Looking at the errors between the modeled and measured data in Figure 9, the power model is not systematically overestimating nor underestimating the total power. The relative accuracy in predicting the power consumption of Merge benchmarks is lower than that of the microbenchmarks, as seen by the right side of Figure 9. The average error for these kernels is 12.8%. Although the predicted data tracks moderately with the actual power data, the gap between the measured and the modeled data for Merge benchmarks tends to be larger than that for the microbenchmarks. The explanation of this discrepancy comes from the performance errors in the timing simulator, mainly due to the complicated behavior of the workloads. In fact, this mismatch is an expected result because there is unknown logic on the real hardware that is not included in the known architecture. For example, modeling is limited by the lack of information on the memory hierarchy such as the specification of the

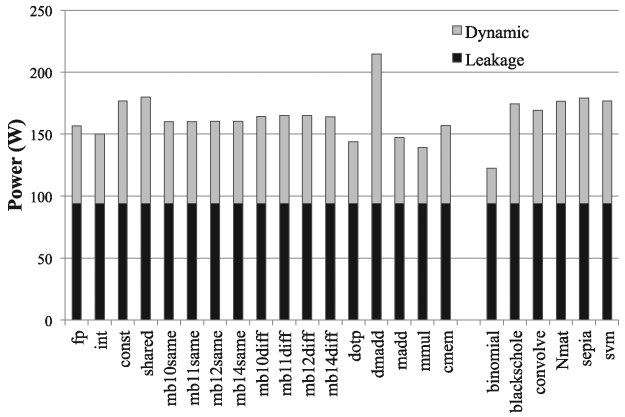

Fig. 10. Distribution between dynamic and leakage power.

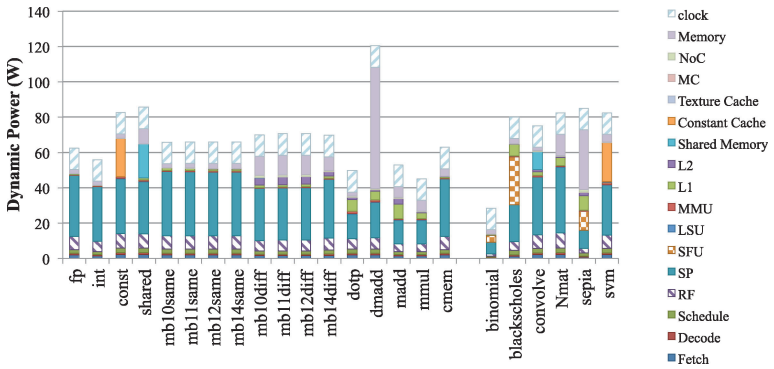

Fig. 11. Breakdown of dynamic power.

interconnection among caches and inside SMs. The reason for the errors also can be attributed to the extra power that is not modeled but is included in the empirical data, for example, the fan power and the power variations caused by the temperature increases. The cold-state temperature of the typical GPU is measured as 57°C [Hong and Kim 2010], and we use 340K (67°C) by considering the temperature increase during the operation. However, different benchmarks have different saturating temperatures; thus the leakage power delta due to this temperature difference can be an additional error.

5.2. Power Analysis Details

This section reports a set of experimental results aimed at demonstrating the usage of the model and understanding the nature of GPU power consumption. In general, there is little information about detailed power data such as power breakdown between components and the absolute numbers even for CPUs. Therefore, we believe that the revealed data in this work will be useful for research on Fermi and other GPU systems. Figure 10 shows the distribution between the dynamic and the leakage power in the total power consumption. Most benchmarks consume more leakage power than dynamic power, but `dmadd` has more dynamic power consumption than leakage power due to the high power consumption at memory. Figure 11 shows the dynamic power distribution between the components. SPs generally consume the most power, as they are the main execution units. On the other hand, for the memory-intensive benchmarks, the power

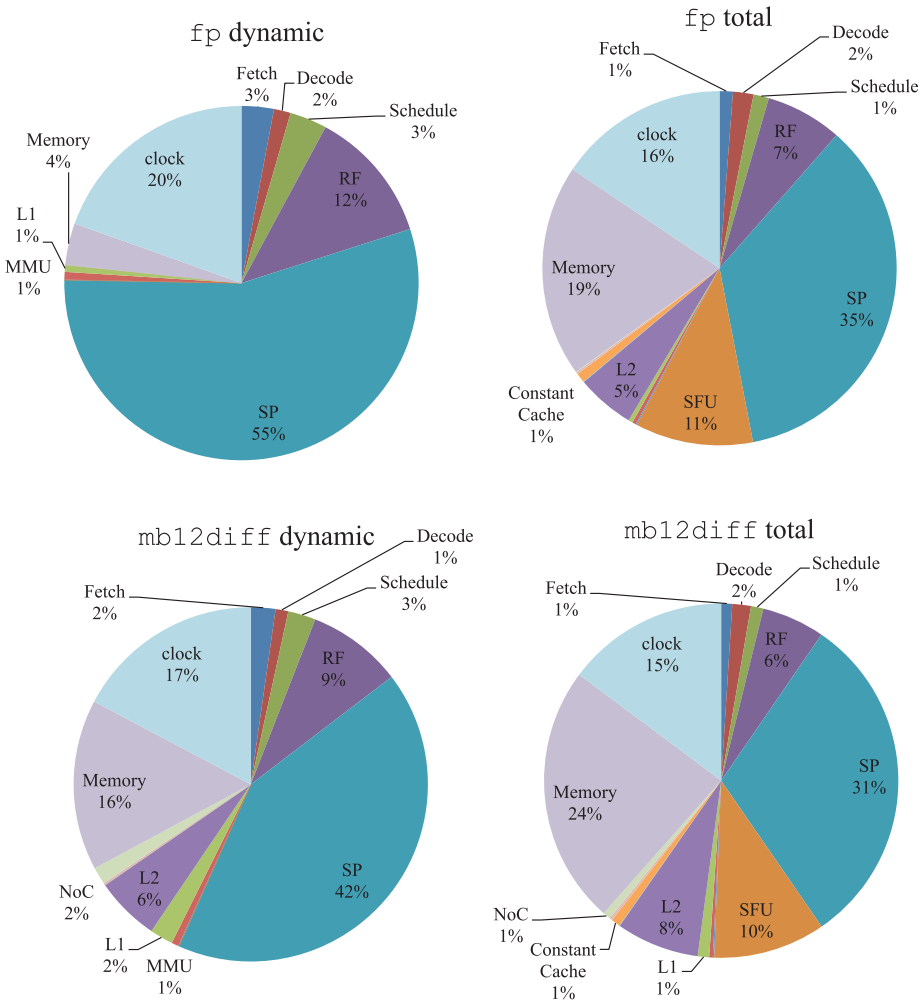

Fig. 12. Breakdown of dynamic and total power for `fp` and `mb12diff` benchmarks.

consumed by the modules related to the memory system, such as the memory controller, caches, and memory, is remarkably increased when compared to power consumption of the same modules for computer-intensive benchmarks, while the core power is low because of the idle time waiting for the memory transfers. According to these results, the big contributors to the dynamic power are SPs, register file, memory, and clock network.

To compare one compute-intensive benchmark (`FP`) and one memory-intensive benchmark (`mb12diff`), Figure 12 presents the breakdown of the dynamic and the total power, and Table XI summarizes the number of accesses to major modules for the workloads. Obviously, for the compute-intensive benchmark, most dynamic power comes from SPs and the register file, but for the memory-intensive benchmark, SPs also consume a significant amount of dynamic power (42%) because arithmetic instructions were used to process the results of the memory instructions and prevent the memory operations from being removed by the compiler.

Figure 13 shows the effect of temperature on leakage power consumption. As the temperature increases, the leakage power increases exponentially. As shown in this

Table XI. The Number of Accesses to the Main Modules for Figure 12, per SM

Statistics	fp		mb12diff	
	Count	Percentage	Count	Percentage
Instruction count	368,509	100%	1,072,883	100%
ALUs in SP	32,407	8.79%	672,665	62.7%
FPUs in SP	336,052	91.2%	272,041	25.4%
Register File	368,459	100%	944,706	88.1%
L1 Cache	19	0.01%	128,162	12.0%
L2 Cache	66	0.02%	2,114	0.20%
Memory	751	0.20%	2,570	0.24%

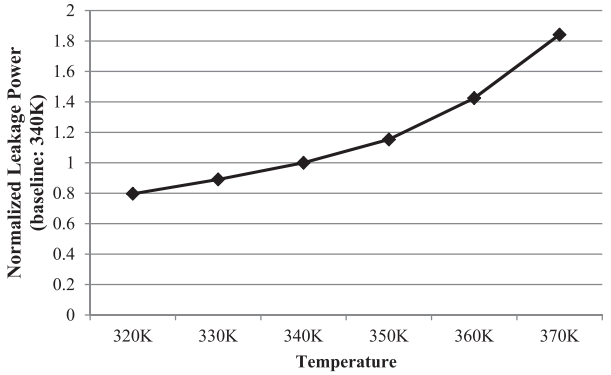

Fig. 13. Leakage power vs. temperature.

figure, the leakage power at the maximum operating temperature (97°C, 370K) is theoretically twice that at 340K. In order to prevent this unnecessary power dissipation and to guarantee the normal operation of the processor, GPUs usually have their own cooling system.

5.3. Case Study: Varying Register File Size

To demonstrate how our model can be used for the design and analysis of a microarchitecture, we evaluate the effect of the register file size on the system, as shown in Figure 14. Since current GPUs manage a heavy register file even larger than the L1, it is important to select an appropriate size to achieve the best performance. Although we refer to these variations as register file size changes, we vary the maximum number of active blocks that can be simultaneously executed on one SM because the number of active blocks determines the effective register size used. In addition, since directly varying the register file size affects the whole microarchitecture because of the interdependence between components, we vary the maximum number of active blocks per SM. To reflect the reduced effective register size, we change the register file size in the power simulation at the same time. In this experiment, we use `blackscholes`, for which the maximum number of active blocks per SM is calculated to be eight using the CUDA occupancy calculator. In Figure 14, a register file size of 32k corresponds to the baseline. From the IPC and power curves, it can be seen that there is a large performance gain and power increase at first by increasing the register file size, but the curves remain fairly flat for sizes greater than 24k, showing diminishing returns.

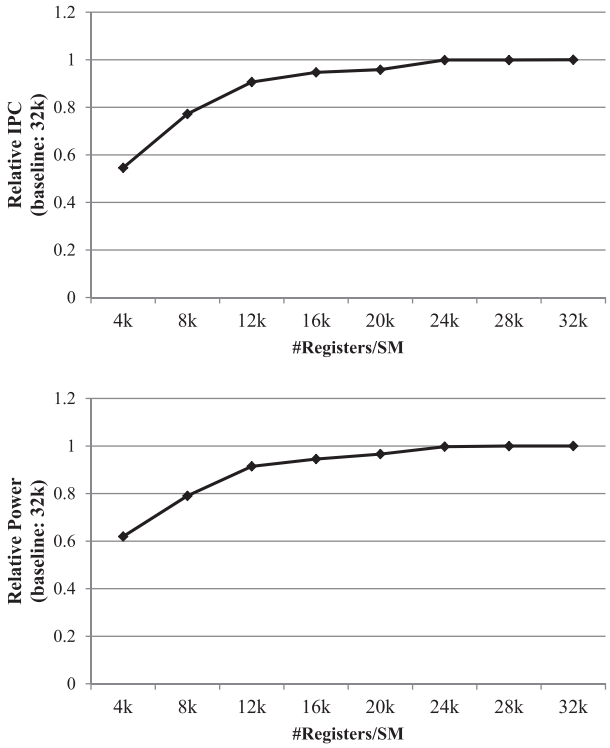

Fig. 14. Effect of varying register file size.

6. RELATED WORK

Modeling power consumption of CPUs has been widely studied. Wattch and McPAT are well-known systematic frameworks to predict the per-module power based on analytical and empirical models. Except for these two tools, other works utilize performance counters or empirical data to build a linear regression model. Joseph and Martonosi [2001] and Isci and Martonosi [2003] proposed using performance counters to examine power-relevant events and to present per-unit power estimates, respectively. Bellosa et al. [2003] similarly used processor events to determine the energy consumption; moreover, the temperature characteristics are estimated for power and thermal management of the system. Wu et al. [2006] search the component unit power of the Pentium 4 by using a K-means-based method to correct inaccuracy resulting from manual tuning using empirical data. Peddersen and Parameswaran [2007] modified a processor for self-prediction at runtime by adding counters capturing power-related events. CAMP [Powell et al. 2009] presented simple equations to provide insight into the relationship between processor parameters and per-structure power as well as estimated power based on microprocessor utilization statistics. Jacobson et al. [2011] built various levels of abstract models and proposed a systematic way to find a utilization metric for estimating power numbers and a scaling method to evaluate new microarchitectures.

Power estimation for other systems such as CMPs and mobile architectures was also conducted. Flores et al. [2007] built an architecture-level power-performance simulator for CMP architectures and validated their work with existing simulators such as Wattch, HotLeakage [Zhang et al. 2003], and Orion [Kahng et al. 2009]. Kanev et al.

[2012] developed a simulation framework to model the power and performance of mobile x86 cores by integrating Zesto [Loh et al. 2009], an x86 simulator, with McPAT. A few existing studies [Gurumurthi et al. 2002; Bircher and John 2012] conduct the estimation and the profiling of the complete system power including CPU, memory hierarchy, and disk subsystem.

While the study of architecture and performance improvement with GPUs has been explored widely, power modeling of GPUs has received little attention. Wang [2010] extended GPGPUSim [Bakhoda et al. 2009] with Wattch and Orion to analyze the GPU power consumption but did not validate his results. PowerRed [Ramani et al. 2011], a modular architectural power estimation framework, combined both analytical and empirical models, and the authors also simulated interconnect power dissipation by employing an area cost. On the other hand, they did not provide the absolute values and validation results. A few GPU power modeling works used a statistical linear regression method rather than an analytical model. Ma et al. [2009] dynamically predicted the runtime power of NVIDIA GeForce 8800 GT using recorded power data and a trained statistical model. Nagasaka et al. [2010] used the linear regression method by collecting the information about the application from performance counters. Some statistical-method-based works used the random forest model to provide insight into understanding the correlation between metrics. Chen et al. [2011] built a high-level power consumption model using a tree-based random forest method to achieve better accuracy than regression-based methods and to study the correlation between individual performance metrics. Zhang et al. [2011] also utilized random forest methods for ATI GPUs and analyzed the power consumption along with performance. Pool et al. [2010] adopted a different approach that built the energy and power model from the unit energy consumed by each instruction. Although past studies employing the regression method have a very small error because they built the model from empirical data obtained from existing hardware, these methods are not applicable for power estimation at the early-design stage. Although our work also utilizes empirical data, our work can be extended to brand new architectures as long as they maintain characteristics of the current GPUs such as the memory hierarchy and three types of GPU-specific execution units. As a concurrent study, GPUWattch has been recently presented [Leng et al. 2013]. Their work also models GPU power using McPAT but the main difference is that our work focuses more on the methodology of developing power models, whereas their work focuses on the GPU power model itself. GPU-PowerSim was also released recently [Goswami et al. 2012], which also uses McPAT and GPGPU-Sim. And GPU-PowerSim is used to evalute GPU regislter files [Goswami et al. 2013].

7. CONCLUSIONS

In this article, we developed a GPU power model using McPAT, a CPU power simulation tool, and also presented steps to develop the model. Our model estimates the power consumption of different GPU components by using configuration parameters that are obtained from published papers in some cases and determined experimentally by exploring all the possibilities provided by McPAT in other cases. Our work exposes undocumented information that is related to the performance and power consumption of the GPU architecture. We focused on identifying appropriate internal McPAT parameters to model GPUs accurately. Through experiments we demonstrate that our model achieves results that are comparable with those measured empirically, with an average error of 12.8% for Merge benchmarks, and that our model tracks the shape of the variation trend relatively well. In other words, these validation results indicate that our power model is an effective solution for both relative and absolute accuracy. Notwithstanding its limitations, this model may offer some insight during

microarchitecture trade-off studies by allowing users to understand the effect of different microarchitecture design options on power consumption.

REFERENCES

A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. 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).* 163–174.

F. Bellosa, S. Kellner, M. Waitz, and A. Weissel. 2003. Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low-Power*.

W. Bircher and L. John. 2012. Complete system power estimation using processor performance events. *IEEE Trans. Comput.* 61, 4, 563–577.

D. Brooks, V. Tiwari, and M. Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00).* ACM Press, New York, 83–94.

J. Chen, B. Li, Y. Zhang, L. Peng, and J. Kwon Peir. 2011. Tree structured analysis on GPU power study. In *Proceedings of the 29th IEEE International Conference on Computer Design (ICCD'11).* 57–64.

J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc. 2013. A roofline model of energy. In *Proceedings of the 27th IEEE International Symposium on Parallel Distributed Processing (IPDPS'13).* 661–672.

Extech. 2014. http://www.extech.com/instrument/products/310_ 399/380801.html.

A. Flores, J. Aragon, and M. Acacio. 2007. Sim-powercmp: A detailed simulator for energy consumption analysis in future embedded CMP architectures. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07),* vol. 1. 752–757.

M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron. 2011. Energy-efficient mechanisms for managing thread context in throughput processors. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11).* ACM Press, New York, 235–246.

N. Goswami, B. Cao, and T. Li. 2013. Power-performance co-optimization of throughput core architecture using resistive memory. In *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA'13).* 342–353.

N. Goswami, A. Verma, and T. Li. 2012. Gpu-powersim. http://www.ideal.ece.ufl.edu/main.php?action=gpu-powersim.

S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John. 2002. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA'02).* IEEE Computer Society, 141.

S. Hong and H. Kim. 2010. An integrated GPU power and performance model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA'10).* ACM Press, New York, 280–289.

Hynix. 2006. 512M (16mx32) GDDR3 SDRAM hy5rs123235fp. http://www.hynix.com/datasheet/pdf/dram/HY5RS123235FP(Rev1.3).pdf.

C. Isci and M. 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 (MICRO'03).* IEEE Computer Society, 93.

H. Jacobson, A. Buyuktosunoglu, P. Bose, E. Acar, and R. Eickemeyer. 2011. Abstraction and microarchitecture scaling in early-stage power modeling. In *Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA'11).* 394–405.

JEDEC. 2014. JEDEC standard GDDR5 SGRAM. http://www.jedec.org/sites/default/files/docs/JESD212.pdf.

R. Joseph and M. Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISPLED'01).* 135–140.

A. Kahng, B. Li, L.-S. Peh, and K. Samadi. 2009. Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of the Design, Automation, and Test in Europe Conference and Exhibition (DATE'09).* 423–428.

S. Kanev, G.-Y. Wei, and D. Brooks. 2012. Xiosim: Power-performance modeling of mobile x86 cores. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'12).* ACM Press, New York, 267–272.

J. Leng, T. Hetherington, A. Eltantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. 2013. GPUWattch: Enabling energy optimizations in GPGPUs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13).* 487–498.

A. Leon, J. Shin, K. Tam, W. Bryg, F. Schumacher, P. Kongetira, D. Weisner, and A. Strong. 2006. A power-efficient high-throughput 32-thread sparc processor. In *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'06).* 295–304.

S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. 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*.

M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng. 2008. Merge: A programming model for heterogeneous multi-core systems. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08)*. ACM Press, New York.

E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. 2008. Nvidia Tesla: A unified graphics and computing architecture. *IEEE Micro* 28, 2, 39–55.

G. Loh, S. Subramaniam, and Y. Xie. 2009. Zesto: A cycle-level simulator for highly detailed microarchitecture exploration. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'09).* 53–64.

X. Ma, M. Dong, L. Zhong, and Z. Deng. 2009. Statistical power consumption analysis and modeling for GPU-based computing. In *Proceedings of the ACM SOSP Workshop Power Aware Computing and Systems (HotPower'09).*

MacSim Simulator. 2012. http://code.google.com/p/macsim/.

S. Mathew, M. Anders, B. Bloechel, T. Nguyen, R. Krishnamurthy, and S. Borkar. 2005. A 4-GHz 300-mW 64-bit integer execution ALU with dual supply voltages in 90-nm CMOS. *IEEE J. Solid-State Circ.* 40, 1, 44–51.

N. Muralimanohart, R. Balasubramonian, and N. Jouppi. 2007. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'07).* 3–14.

H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka. 2010. Statistical power modeling of GPU kernels using performance counters. In *Proceedings of the International Green Computing Conference*. 115–122.

NVIDIA. 2009. Fermi: Nvidia's next generation CUDA compute architecture. White paper. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.

NVIDIA. 2014a. Geforce GTX280 specification. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-280.

NVIDIA. 2014b. Geforce GTX580 specification. http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580/specifications.

NVIDIA. 2014c. Nvidia GF100. http://www.hardwarebg.com/b4k/files/nvidiagf100whitepaper.pdf.

J. Peddersen and S. Parameswaran. 2007. Clipper: Counter-based low impact processor power estimation at runtime. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'07).* 890–895.

J. Pool, A. Lastra, and M. Singh. 2010. An energy model for graphics processing units. In *Proceedings of the IEEE International Conference on Computer Design (ICCD'10).* 409–416.

M. Powell, A. Biswas, J. Emer, S. Mukherjee, B. Sheikh, and S. Yardi. 2009. Camp: A technique to estimate per-structure power at run-time using a few simple parameters. In *Proceedings of the 15th IEEE International Symposium on High Performance Computer Architecture (HPCA'09).* 289–300.

K. Ramani, A. Ibrahim, and D. Shimizu. 2011. Powerred: A flexible power modeling framework for power efficiency exploration in GPUs. In *Proceedings of the Workshop on General Purpose Processing on Graphics Processing Units (GPGPU'11).*

P. Rosenfeld, E. Cooper-Balis, and B. Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *Comput. Archit. Lett.* 10, 1, 16–19.

W. Song, S. Yalamanchili, S. Mukhopadhyay, and A. Rodrigues. 2012. *Energy Introspector User Manual*. Georgia Tech Research Corporation.

A. Stepin and Y. Lyssenko. 2014. Natural born winner: Nvidia Geforce GTX580 review. page 5. http://www.xbitlabs.com/articles/graphics/display/geforce-gtx-580_5.html.

G. Wang. 2010. Power analysis and optimizations for GPU architecture using a power simulator. In *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE'10),* vol. 1. V1–619–V1–623.

W. Wu, L. Jin, J. Yang, P. Liu, and S.-D. Tan. 2006. A systematic method for functional unit power estimation in microprocessors. In *Proceedings of the 43rd ACM/IEEE Design Automation Conference (DAC'06).* 554–557.

Y. Zhang, Y. Hu, B. Li, and L. Peng. 2011. Performance and power analysis of ATI GPU: A statistical approach. In *Proceedings of the 6th IEEE International Conference on Networking, Architecture and Storage (NAS'11)*. 149–158.

Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. 2003. Hotleakage: A temperatureaware model of subthreshold and gate leakage for architects. Tech. rep. University of Virginia, VA.

Received January 2013; revised March 2014; accepted March 2014