GPUWattch[†]: Enabling Energy Optimizations in GPGPUs

Jingwen Leng[1], Tayler Hetherington[2], Ahmed ElTantawy[2], Syed Gilani[3],
Nam Sung Kim[3], Tor M. Aamodt[2], Vijay Janapa Reddi[1]

[1] The University of Texas at Austin, [2] University of British Columbia, [3] University of Wisconsin-Madison

ABSTRACT

General-purpose GPUs (GPGPUs) are becoming prevalent in mainstream computing, and performance per watt has emerged as a more crucial evaluation metric than peak performance. As such, GPU architects require robust tools that will enable them to quickly explore new ways to optimize GPGPUs for energy efficiency. We propose a new GPGPU power model that is configurable, capable of cycle-level calculations, and carefully validated against real hardware measurements. To achieve configurability, we use a bottom-up methodology and abstract parameters from the microarchitectural components as the model's inputs. We developed a rigorous suite of 80 microbenchmarks that we use to bound any modeling uncertainties and inaccuracies. The power model is comprehensively validated against measurements of two commercially available GPUs, and the measured error is within 9.9% and 13.4% for the two target GPUs (GTX 480 and Quadro FX5600). The model also accurately tracks the power consumption trend over time. We integrated the power model with the cycle-level simulator GPGPU-Sim and demonstrate the energy savings by utilizing dynamic voltage and frequency scaling (DVFS) and clock gating. Traditional DVFS reduces GPU energy consumption by 14.4% by leveraging within-kernel runtime variations. More finer-grained SM cluster-level DVFS improves the energy savings from 6.6% to 13.6% for those benchmarks that show clustered execution behavior. We also show that clock gating inactive lanes during divergence reduces dynamic power by 11.2%.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures; C.4 [**Performance of Systems**]: Modeling techniques

General Terms

Experimentation, Measurement, Power, Performance

Keywords

Energy, CUDA, GPU architecture, Power estimation

[†]GPUWattch is named after the original CPU power modeling framework, Wattch [6], which enabled widespread architecture-level power analysis and optimizations. However, our approach is independent of Wattch.

ISCA '13 Tel-Aviv, Israel

1. INTRODUCTION

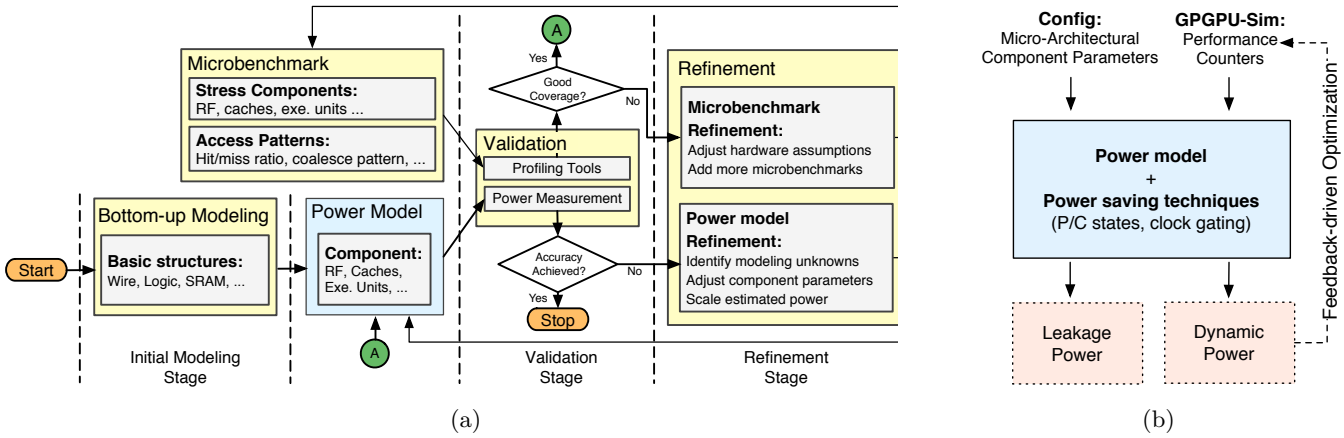
From datacenters to power-constrained mobile devices, performance per watt has emerged as an indispensable metric for evaluating the efficiency of a GPU architecture [9, 18]. Although GPU performance models, such as GPGPU-Sim [4], Multi2Sim [34] and MacSim [1], have enabled performance oriented research on branch divergence [11, 12, 27], memory bandwidth pressure [5, 32], and so forth, similar efforts to investigate and optimize GPU energy-efficiency problems have been difficult owing to the lack of a suitable power modeling infrastructure. Researchers using these tools may inadvertently be optimizing for performance while penalizing performance per watt. To avoid such pitfalls and develop energy-efficient GPU architectures, we require a robust power model.

A robust power model must satisfy three requirements to be useful for computer architecture research. It must be (1) configurable, (2) cycle level, and (3) strongly validated against existing processor architectures using a rigorous methodology. As shown in Table 1, Wattch [6] and McPAT [22] are robust CPU power models that satisfy all three requirements, and as such have enabled new research areas in energy-efficient CPU design. No such power model exists for GPU architecture research. Hong and Kim [13] were the first to propose an integrated power and performance model for GPUs. However, their power model is not configurable to different architectural parameters. Moreover, it is incapable of providing cycle-level power estimates to evaluate fine-grained power saving techniques such as clock gating.

In this paper, we introduce *GPUWattch*, a new power model that addresses all of the aforementioned requirements. We follow the rigorous process shown in Figure 1(a) to develop the robust power model. We use a bottom-up methodology to build the initial model. Then we compare our simulated power with the measured hardware power to identify any modeling inaccuracies. We resolve these inaccuracies using a special suite of 80 microbenchmarks that are designed to create a system of linear equations that correspond to the total power consumption. By solving for the unknowns in the system, we progressively eliminate the inaccuracies. We

Work	GPU	Configurable?	Cycle-level?	Validated?
Wattch/McPAT [6, 22]	No	Yes	Yes	Yes
Hong and Kim [13]	Yes	No	No	Yes
GPUWattch	**Yes**	**Yes**	**Yes**	**Yes**

Table 1: Robust power modeling requirements for a GPU.



(a) (b)

Figure 1: (a) Steps to build a robust power model. The various stages indicate our systematic and rigorous methodology to iteratively identify and refine inaccuracies in the model. (b) Our integrated power and performance modeling framework.

validated the simulated power model's average and runtime estimates against measured results using a comprehensive set of 25 real-world kernels that were not used to originally develop the power model. The power model achieves an average accuracy that is within 9.9% error of the measured results for the GTX 480 and 13.4% for the Quadro FX5600. Moreover, it accurately tracks the trace of relative power consumption over time.

A salient feature of our power model is its longevity. Power consumption is often sensitive to the details of the specific architecture, implementation choices, and technology parameters, all of which continuously evolve from one generation to another. Therefore, the power model must be continuously adapted, refined, and validated for evolving GPU architectures and their manufacturing technology. A key distinguishing feature of our power model, as compared to other architecture-level power models, is that it is equipped with the suite of microbenchmarks and a refinement and validation methodology that can support future GPUs.

We integrate the power model with GPGPU-Sim [4] for cycle-level power calculation, as shown in Figure 1(b), to establish a complete framework for exploring energy-efficient GPU architectures and power-management techniques. On the basis of this framework, we demonstrate that traditional processor-level dynamic voltage and frequency scaling (DVFS) achieves 14.4% energy savings in GPUs by leveraging the phase behavior within kernels. Average performance loss is within 3%. Furthermore, we identify new opportunities for cluster-level DVFS (i.e., grouping streaming multiprocessors [SMs] in the GPU to perform DVFS separately). Cluster-level DVFS can achieve an additional 7% energy savings over conventional DVFS for benchmarks that exhibit such grouped behavior. We also evaluate fine-grained lane-level clock-gating in order to reduce processor power consumption when SIMD lanes go inactive due to branch divergence. On average, *lane gating* reduces dynamic power consumption by approximately 11.2%.

In summary, our work makes the following contributions:

1. We propose a power model based on the bottom-up methodology for GPU architecture research that can enable performance per watt energy-efficiency studies.
2. We describe a systematic and rigorous methodology to develop and validate the power model using a large variety of microbenchmarks that stress test the microarchitecture.
3. We demonstrate the opportunities for improving the energy efficiency of GPUs using both traditional techniques (DVFS) and new techniques (lane gating).

We begin with an overview of our power modeling approach in Section 2. We explain how we model the various important microarchitectural structures. To assess and fix any discrepancies in the initial model, we describe our extensive microbenchmarking methodology in Section 3. Next, we validate the power model comprehensively using hardware measurements in Section 4. We then demonstrate the benefits of two hardware optimizations to save GPU power in Section 5. We compare our results with related work and conclude the paper in Section 6 and Section 7, respectively.

2. GPU POWER MODELING

We model GPU architectures similar to NVIDIA's GPUs. The main components shown in Table 2 include streaming multiprocessors (SMs), memory controllers, the interconnection network, and DRAM. Figure 2 shows the architecture of an SM in a GPU. The SM's major difference from a traditional CPU core is that it has a SIMD execution unit and also contains a texture cache, constant cache, and shared memory. GPUs also adopt GDDR instead of DDR memory for higher bandwidth [28]. Note that our initial modeling may differ from the real hardware's implementation. Therefore, during the refinement stage of the modeling methodology we update the initial model estimates to be more precise, as shown in Figure 1(a).

Equation (1) captures at a very high level all aspects of GPU power that we model, which consists of the leakage, idle SM, and all components' (N in total) dynamic power. Each component's dynamic power is calculated as the activity factor (α_i) multiplied by the component's peak power (P_{max_i}).

$$Power = \sum_1^N (\alpha_i * P_{max_i}) + Idle\ SM\ power + Leakage \quad (1)$$

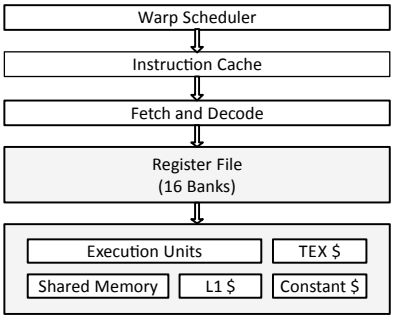


Figure 2: Streaming multiprocessor (SM) overview.

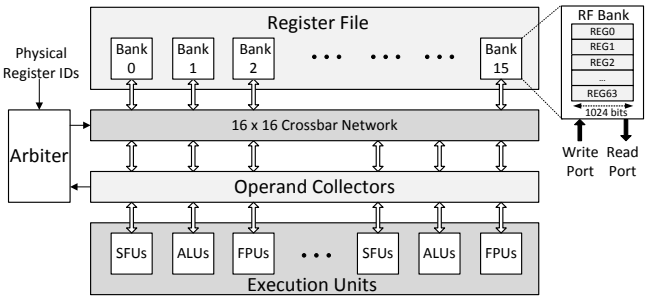


Figure 3: Register file structure in the Fermi architecture.

2.1 Infrastructure

We use publicly available resources that describe the GPU microarchitecture and rely on McPAT [22] to model most microarchitectural blocks (refer to Table 2). Throughout, we stay consistent with the process of abstracting the microarchitectural parameters of each component and using them to model each component's circuit implementation in order to ensure configurability.

Although McPAT includes detailed models for several microarchitectural blocks, many components are either not present or are considerably different for GPUs as compared to general-purpose CPUs. Therefore, we added or adapted several important blocks in McPAT to more accurately represent the underlying GPU microarchitecture. We model SRAM array structures using CACTI [33].

2.2 Register File

Owing to the lack of resources for the NVIDIA GPU register file architecture, we adopt the architecture used in GPGPU-Sim version 3.2.1 (see Figure 3), which is inspired by NVIDIA patents [24,25]. Each SM has a large and unified register file that is shared across warps executing in the same SM. GPUs adopt a multibanked register file architecture to avoid the area and power overhead of a multiported register file. Crossbar networks and operand collectors are used to transport operands from different banks to the appropriate SIMD execution lane.

The register file's bandwidth and size determines the number and size of banks. Fermi GPUs can perform a fused multiply-add (FMA) operation per cycle [28], which needs three input operands and one output operand from the register file. Each SM contains 32,768 32-bit registers, which are shared by all threads executed on the same SM. We model the register file as 16 dual-ported (one read and one write port) banks, each of which provides 2,048 logical 32-bit registers. Physically, these registers are accessed as 64 1,024-bit registers. Thus, each operand read from a register file bank provides operands for 32 SIMT threads. We estimate the power consumption for these memory arrays using CACTI.

A crossbar interconnection network is used to transfer operands from register file banks to operand collectors. Furthermore, in the case of bank conflicts, the arbiter shown in Figure 3 is responsible for serializing the register file access, and the operand collectors are used to buffer the data already read from the register file.

We include the crossbar interconnect model in our register-file power model. The crossbar network's parameters are determined by the number of register-file banks, as well as the number of operand collector units. For the Fermi architecture, we model a 16-input × 16-output crossbar network, as shown in Figure 3. Each input and output of the crossbar is 1,024 bits wide. Consequently, the operands read from a register file bank are routed to the proper operand collector. We model each operand collector as an SRAM array bank.

Description	Microarchitectural Components	Basic Structures
SM Pipeline	Pipeline	In-order multithreaded pipeline with SIMD datapath
Register Files	Register-file banks	1024-bit wide, 64-entry SRAM
	Operand collectors	1024-bit wide SRAM
	Collection network	1024-bit wide 16 x 16 crossbar
Shared Memory	Shared memory banks	Architecture dependent size
	Shared memory network	32 x 32 crossbar network
Caches	L1, L2, texture, and constant caches	Architecture dependent size
	Memory-coalescing logic	Synthesis-based power model
Execution Unit	Integer ALUs	Synthesis-based power model
	Floating-point units	
	Special-function units	
Main Memory	GDDR5/GDDR3	Empirical model
	Memory Controller Network-on-chip	McPAT model

Table 2: Main GPU components in the Fermi that we model as best we can due to limited public information.

2.3 Shared Memory

NVIDIA GPUs contain shared memory per SM. It can be used for interthread communication for threads in a thread block. Because SIMD lanes can access any address concurrently, shared memory is multibanked and contains a crossbar interconnect to improve performance. That is, shared memory has a very similar structure to the register files (Section 2.2). In the case of the Fermi architecture, the number of banks is 32 and the crossbar is 32-input×32-output.

2.4 Execution Units

We model the FP pipeline in the Fermi architecture using floating-point FMA units. Two lanes of FMA units can combine to execute a double-precision (DP) operation. The SFU units are also modeled as DP FMA units. Because most instructions executed on the SFU units are transcendental operations that employ iterative algorithms to compute the result, these units' power consumption depends on the latency and throughput of instructions.

Area and power modeling of execution units are initially estimated based upon synthesis of their Verilog descriptions. The execution units are synthesized using a 45-nm standard cell library. The synthesized netlists are annotated with

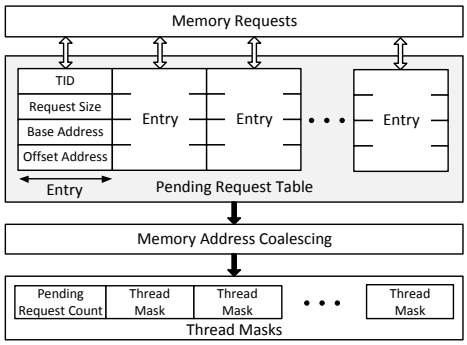


Figure 4: Main components of memory address coalescing.

switching activity for random inputs. We use the Synopsys Power Compiler® [3] to estimate the power consumption of the designs after the annotation of switching activity. All designs assume an operating nominal V_{DD} of 1 V. We perform technology and voltage scaling according to *ITRS* projections to estimate the dynamic power consumption at the given voltage and technology node.

2.5 Memory Coalescing Logic (MCL)

GPUs' load/store units are responsible for coalescing memory accesses to reduce the bandwidth usage. Figure 4 depicts MCL's main structures. Each MCL contains a pending request table (PRT) with multiple entries. Each PRT entry stores the thread index, the base and offset addresses for the memory access, and the request sizes for all of the concurrently issued memory requests by a warp. These entries are written to the PRT whenever a memory request is issued.

To determine the number of coalesced memory requests that a warp must issue, MCL compares the base address of the first thread's request with the base addresses of all the remaining requests in the PRT entry. The memory request mask of all the SIMT threads with the same base address is set to zero. The process is repeated until masks have been generated for all requests in the PRT entry. These masks are stored in a separate thread mask array. Memory requests are generated for all addresses whose mask bit are set. For each PRT entry, a pending request count (PRC) is maintained, and is incremented when a request is sent to memory and decremented when a response is received. When the PRC becomes zero, requests for the warp are satisfied. We model the PRT as an SRAM array structure using CACTI.

2.6 Idle SMs

Due to load imbalance, it is possible to have SMs that are idling during execution. In our experiments, we observe that an idle SM still consumes a noticeable amount of power that affects the accuracy of the power model. Some benchmarks, such as HRTWL and MGST, suffer from this load imbalance issue, where nearly a quarter of their SMs becomes idle after finishing their work. For the idle SM, activity factor α_i in (1) is zero but the SM still consumes power. Therefore, it is important and necessary to model the idle power of individual SMs correctly.

We determine the dynamic power consumption of an idle SM using a microbenchmarking methodology. First, we use a microbenchmark in which each SM can run at most one thread block concurrently. Then we vary the number of active SMs to perform a linear regression fit to estimate the power used when all SMs are idle. We subtract this power from the constant power component of our GPU card (discussed in Section 4.2) to determine the dynamic power component of an idle SM.

2.7 Main Memory

Our paper focuses on modeling GPU chip power. We found no public information on GDDR3/5 power modeling and while flexible DRAM power models [35] could be incorporated into our effort, they require additional reverse engineering of the GDDR implementation, which is beyond the scope of our processor microarchitecture-centric work. Hence, we use an empirical approach based on prior work [13] to compute the effect of DRAM on our studies. The approach considers only DRAM dynamic power modeling.

Equation (2) states the empirical DRAM dynamic power model we used. We consider its dynamic power to be composed of precharge power, row buffer activation power, read power, and write power. The precharge power is calculated as the per-operation energy (E_{pre}) times the number of precharge operations (c_{pre}) divided by the execution time. Read, write, and activation powers are calculated similarly.

$$P_{DRAM} = \frac{E_{pre} * c_{pre} + E_{act} * c_{act} + E_{wr} * c_{wr} + E_{rd} * c_{rd}}{Execution\ Time} \quad (2)$$

The above DRAM power model can be lumped into Equation (1), resulting in a complete linear system with all GPU components' access rates. We treat the access energy for each DRAM operation as an unknown variable. In the next section, we explain the process of solving for the unknowns.

3. MICROBENCHMARKING THE UNCERTAINTIES

Microbenchmarking plays a critical role in the refinement and validation of our initial power model. We rely on microbenchmarks to address the power modeling uncertainties that arise for various reasons, such as misguided assumptions about undocumented features. The microbenchmarks also help us isolate the power consumption of key components in the GPU so that we can validate and refine the power model's component-level power breakdown. They also help us achieve good test coverage of the various components in the processor. Real benchmarks do not stress all aspects of the processor microarchitecture as the microbenchmarks.

3.1 LSE Problem Formation

A major source of inaccuracies in the initial power model arises from uncertainties due to undocumented design decisions in the target architecture. Undocumented aspects of the GPUs we model include unknown sizes or configurations of components. Our first step is to ascertain these inaccuracies.

We use an iterative process to continuously refine the power model based on inaccuracies that we observe between the power model and the actual hardware power measurements. As Equation (3) shows, we model the dynamic power consumption as a linear combination of access rate α_i (access counts divided by time) of each microarchitectural component multiplied by its peak power P_{max_i}. We consider the modeling inaccuracy for a particular microarchitectural component i as an unknown variable x_i in Equation (4).

```
void kernel (unsigned *A, unsigned *C){
   unsigned tid, offset, sum=0;
   tid = get_tid(); //thread id
   offset = (tid%M + tid/K)*OFFSET;
   sum += A[offset]; //manually unroll
   sum += A[offset];
   ...//operator(+=) creates dependency
}
```

Figure 5: A generic microbenchmark stressing the L1 cache.

Thus, if there are N access rates for the different microarchitectural components, each microbenchmark will yield a microarchitectural component access-rate vector that constitutes one linear equation. With an arbitrary number of microbenchmarks, say M, this will result in a $M \times N$ linear estimation problem, as shown in Equation (5). With a sufficiently large number of equations (i.e., microbenchmarks) and hardware power measurements (P_M), we can effectively solve the modeling inaccuracies using the least-squares estimation (LSE).

$$P_{dynamic} = \sum_1^N (\alpha_i * P_{max_i}) \quad (3)$$

$$P_{max_i} = P_{modeled\_max_i} * x_i \quad (4)$$

$$\mathbf{A}_{M\times N} \times \mathbf{x}_{N\times 1} = \mathbf{P}_{M\times 1} \quad (5)$$

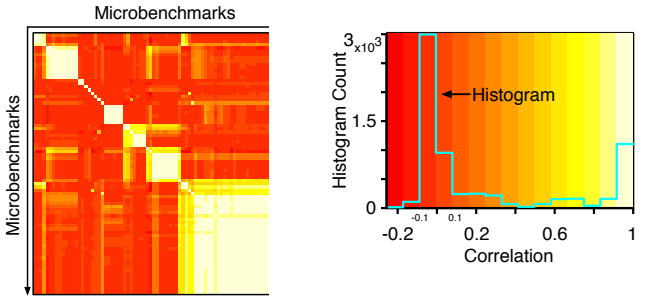
We iteratively refine the power model on the basis of the sources of the various inaccuracies that LSE identifies. For instance, in our infrastructure (i.e., McPAT) the power estimation for certain components is biased toward CPU implementations. We narrow the resulting inaccuracy gap for the GPU power model by fixing our initial assumptions about the implementation and then applying the scaling factors that are obtained from LSE.

3.2 Microbenchmarking Design Methodology

To aid the LSE solving process, we use a systematic microbenchmarking methodology. We describe the three important microbenchmark characteristics that we identified as requirements. Then we show that our microbenchmarks satisfy these requirements.

Component Stress To solve the LSE problem formed by all microbenchmarks effectively, the performance counter vector of each microbenchmark should have as low a correlation as possible with the others [16]. We achieve this by designing some microbenchmarks to stress individual microarchitectural components. For example, we limit the number of variables in the execution unit stressing microbenchmark so that all necessary values reside in the register files with minimal access to caches or DRAM. Similarly, when designing L2 cache microbenchmarks, we disable the L1 cache both in the simulator and hardware for power isolation. We follow a similarly strict design methodology for all of the other microbenchmarks. This step is also important for enabling component-level model validation (discussed in Section 4).

Access Patterns It is important to design microbenchmarks to exercise the same microarchitectural component in different ways, because switching activity impacts dynamic power consumption. Thus, we created additional microbenchmarks with different access patterns to create more equations for LSE estimation. For example, cache microbenchmarks exercise different memory spaces, hit ratios, and coalescing patterns, using reads or writes. The kernel shown in Figure 5 is a parametrized microbenchmark designed for the L1 cache that can be configured to achieve different hit ratios, with different coalescing patterns. The cross-thread instruction locality within each thread is highly amortized by massive multithreading; hence, the hit ratio is mostly determined by the interthread locality, which can be engineered based on the K, M and OFFSET parameters.



(a) Heat map. (b) Color key and histogram.

Figure 6: Correlation heat map of all 80 microbenchmarks.

Test Coverage Real benchmarks only exercise a small portion of the processor. For example, few benchmarks use the texture or constant caches. To ensure that our refinement and validation tests all components of the processor, we rely on our microbenchmarks to achieve good test coverage. As shown in Table 3, we designed four sets of microbenchmarks. Functional microbenchmarks are designed to exercise executional units such as integer and floating point. Memory microbenchmarks access the various caches and shared memory intensively. DRAM exercises the main memory. In addition, we also create the MIX microbenchmarks that have more complex access patterns such as accessing multiple components simultaneously. These efforts result in 80 unique microbenchmarks.

Name	Exercised Components	Counts	Name	Exercised Components	Counts
Func.	Integer, floating point and special function unit	11	Mem.	L1 & L2 & texture & constant caches, and shared memory	25
DRAM	DRAM	22	Mix	Mix of functional, DRAM and memory	22

Table 3: 80 refinement and validation microbenchmarks.

Microbenchmark Correlation Figure 6a shows a matrix for the correlation between any two microbenchmarks' performance counter vectors for our 80 microbenchmark. Figure 6b shows its color key (the hotter the color is, the higher correlation two microbenchmarks have), and the histogram for correlation among all microbenchmarks. Figure 6b indicates that most microbenchmarks have a low correlation around ± 0.1, which is good because it shows that these microbenchmarks are independent and stress components differently. The only microbenchmarks that have a high correlation are the DRAM microbenchmarks that are used to develop DRAM power model. We expect this high correlation because the activity differences among these microbenchmarks are limited to DRAM reads and writes.

4. POWER MODEL VALIDATION

We begin this section with a detailed description of our power measurement setup for real GPU cards. We select two NVIDIA GPU cards with different architectures to show our

power model's configurability for validation. Table 4 summarizes the differences between the architectures. We focus on comprehensive validation of the leakage power, average dynamic power, and dynamic power trace of kernels against measurement results for both the GPU card configurations.

Category	GTX 480	Quadro FX5600
Architecture	Fermi	G80
CUDA cores	480	120
Frequency	1.4 GHz	1.2 GHz
Register file size	131 KB	32 KB
Shared memory	48 KB	16 KB
L1 cache	16 KB	N/A
L2 cache	768 KB	192 KB
Texture cache	12 KB	5 KB
Constant cache	8 KB	8 KB
Technology node	40 nm	65 nm
Memory type	GDDR5	GDDR3
Memory bandwidth	177.4 GB/s	76.8 GB/s
Memory controllers	6	6
Modeled Leakage Power	41.9W	21.3W

Table 4: Used two GPU cards configurations.

4.1 Experimental Setup

We validate our power model with a reference hardware platform containing both the cards. We provide details here.

4.1.1 Power Measurement Setup

We use both NVIDIA GTX 480 and Quadro FX5600 for validation. Both the cards are connected to the PCIe slot through a PCIe riser card and an ATX power supply. The PCIe riser card and the ATX power supply have power pins that deliver power to the GPU. For each power supply source, we measure the instantaneous current and voltage to compute power. We sense the current draw by measuring the voltage drop across a current sensing resistor. We use a NI DAQ to sample the voltage drop at a rate of 2 MS/s.

Figure 7 shows a detailed schematic of our setup. The diagram illustrates the peripheral components that we are measuring as part of the total GPU power. These peripherals include the GPU processor, DRAM modules, voltage regulator module (VRM), and other auxiliary support circuitry, as shown in Equation (6). Our power model is built to model both static leakage and dynamic power, and therefore in our experiments we separate these two parts and validate them individually.

$$P_{measured} = \underbrace{P_{proc\_leak} + P_{mem\_leak} + P_{VRM} + P_{peripherals}}_{P_{const}\ -\ independent\ of\ frequency} + \underbrace{P_{proc\_dynamic} + P_{mem\_dynamic}}_{P_{dyn}\ -\ scales\ with\ frequency} \quad (6)$$

4.1.2 Simulator Setup

The software power model builds on McPAT 0.8. It is integrated with GPGPU-Sim version 3.2.1. We configure the simulator to match the two GPU cards separately. We use GPGPU-Sim's PTXPlus mode to simulate the native instruction set (ISA) on the Quadro GPU (SASS). We also use the NVIDIA compute profiler [29] to ensure that the microbenchmarks' performance matches the target hardware.

4.2 Constant Power Component

The measurement setup captures both dynamic and constant power. Equation (6) shows that each power source (P_{const} and P_{dyn}) consists of different components. The first part, P_{const}, includes processor leakage power, main memory leakage power, VRM power, and all peripheral circuits' power. P_{const} is independent of processor/memory frequency. The other part, P_{dyn}, comprises the dynamic power of both the processor and the main memory. P_{dyn} scales linearly with processor/memory frequency. Therefore, if memory frequency scales the same as the processor's frequency, we can rewrite the measured power in Equation (6) as a linear function of only the processor frequency f. This is shown below in Equation (7):

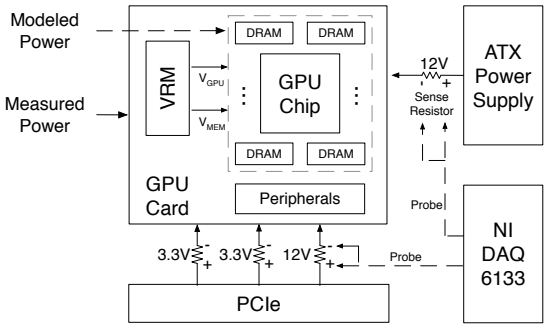


Figure 7: GPU power measurement schematic.

$$P_{measured} = k * f + P_{const} \quad (7)$$

k – constant

f – processor frequency

Using Equation (7) with varying frequencies f, we can determine the constant power component. With the aid of overclocking tools for the NVIDIA GPU card, we scale the processor and DRAM frequencies separately and measure power. With the simplified linear model in the equation, we performed a linear fit to get the constant power component.

Subtracting the constant power numbers from the measured total power gives us the *dynamic* power component of the GPU processor and its main memory. We perform the above procedure on both the test cards. Figure 8 shows the experiment results for GTX 480. We use two different workloads. The first workload exercises the integer unit heavily, while the second workload exercises the floating-point unit. We choose these two computation-intensive workloads to minimize interaction with the DRAM memory subsystem. Figure 8 shows the constant power component for the GTX 480 card, which is approximately 59 W. The constant power component for the Quadro FX5600 card is 38 W.

Another reason for determining the constant power component is to get the static leakage power of both GPU cards. Our model reports 41.9 W and 21.3 W leakage power for the GTX 480 and Quadro FX5600, respectively. However, we cannot validate these numbers owing to the lack of publicly available resources on GPU leakage power numbers. Nevertheless, according to Equation (6), the linearly fitted constant power serves as an upper bound, which the results reported by our power model do not violate. We do not have leakage power estimates for the GDDR5 memory, because we adopt an empirical approach to estimate its power.

4.3 Dynamic Power Validation

We validate the power model using microbenchmarks and real programs from public benchmark suites. We compare the average power and the dynamic power behavior of several kernels against the measured hardware results. The measurement process has inherent limitations that affect the

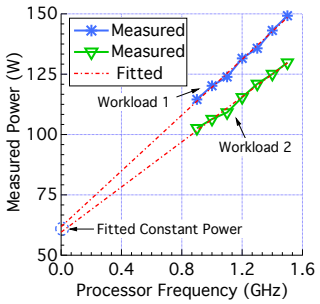


Figure 8: Linear regression fit to get constant power.

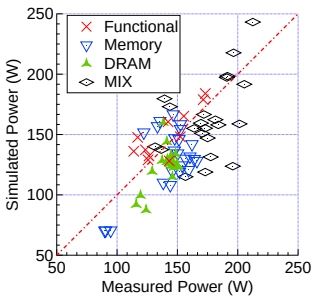
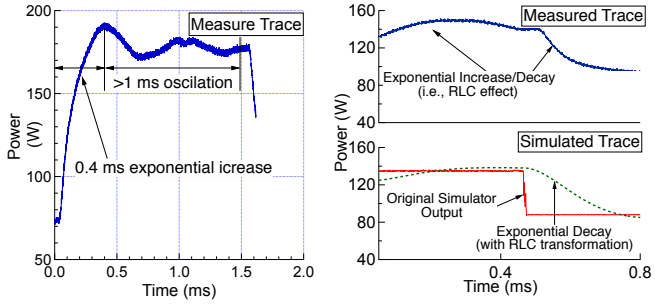


Figure 9: Microbenchmark power comparison.



(a) (b)

Figure 10: (a) RLC effect in measured trace. (b) Simulated trace with derived RLC filter to match measured trace.

benchmark selection and dynamic power trace comparisons, and this is not to be confused with simulation inaccuracies.

4.3.1 Hardware Measurement Issues

In our measurements, we find that any large changes in power draw are accompanied by an exponential increasing or decaying curve (i.e., *RLC effect*), as well as an oscillation, as shown in Figure 10(a). The behavior is related to the power-delivery network (PDN) component characteristics and interactions (i.e., voltage regulator module and decoupling capacitance), which effectively form an RLC circuit. These effects are impossible to isolate from our power measurements because they are board-level components. We verified that the exponential power trace is not caused by workload or architectural policies, such as thread scheduling, by constructing microbenchmarks.

Figure 10(b) shows a snapshot of the measured and simulated power trace for MGST_K1 in which we can mimic the RLC behavior in the measured trace by transforming our simulated power trace with a derived RLC filter. This RLC filter is derived based on Figure 10(a), which is effectively the filter's step response. Analyzing Figure 10(a), the filter can be approximated as a second-order system with rise time 0.18 μs and maximum overshoot 15%.

In our experiment, we must filter out GPU kernels with execution less than 500 μs because the measured power trace of this type of kernel is always in the exponential increase stage due to the RLC effect. This type of kernel could have a much higher power consumption than the measured value. Thus we only trust kernels with long enough execution times ($> 500\ \mu s$) for validation purposes. For each benchmark, we use the largest available input sizes and modify parameters to increase the execution time without changing functionality. However, this execution time limitation still filters out 56 unique kernels in the ISPASS and RODINIA benchmarks suites to the 21 kernels discussed in Section 4.3.3. In addition, we study another four kernels from prior work by Hong and Kim [13]. So in all, we have 25 long-running kernels. To avoid the RLC effect, our microbenchmarks were designed with long enough (typically several seconds) execution time.

4.3.2 Microbenchmark based validation

Following the rigorous design methodology discussed in Section 3, we designed 80 microbenchmarks that are used to iteratively refine our power model. We present the final power model accuracy for these microbenchmarks and validate the component-level power isolation microbenchmarks by showing their simulation power distribution.

Average Power Test coverage is one purpose of our microbenchmarks. Our microbenchmark design effort results in an even distribution of measured steady-state power. Figure 9 shows the comparison between our power model and the hardware for microbenchmarks running on the GTX 480. The measured results on the x-axis are spread evenly from 80 W to 210 W, and simulated power on the y-axis tracks the corresponding power difference across different microbenchmarks. As we will show later, real benchmarks' power has a much smaller variation range (120 W $\sim$ 190 W).

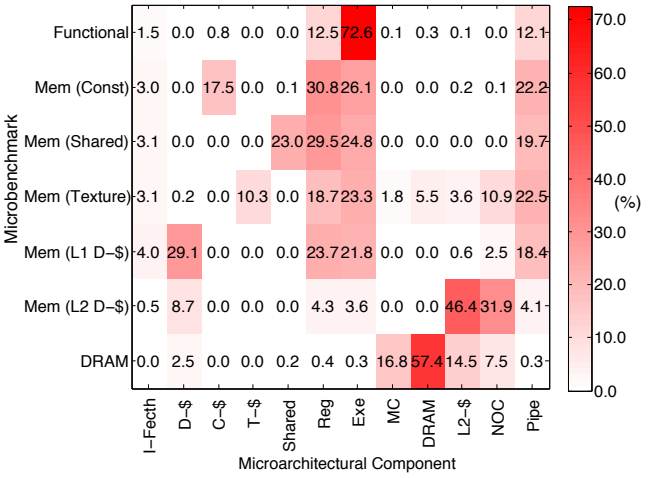
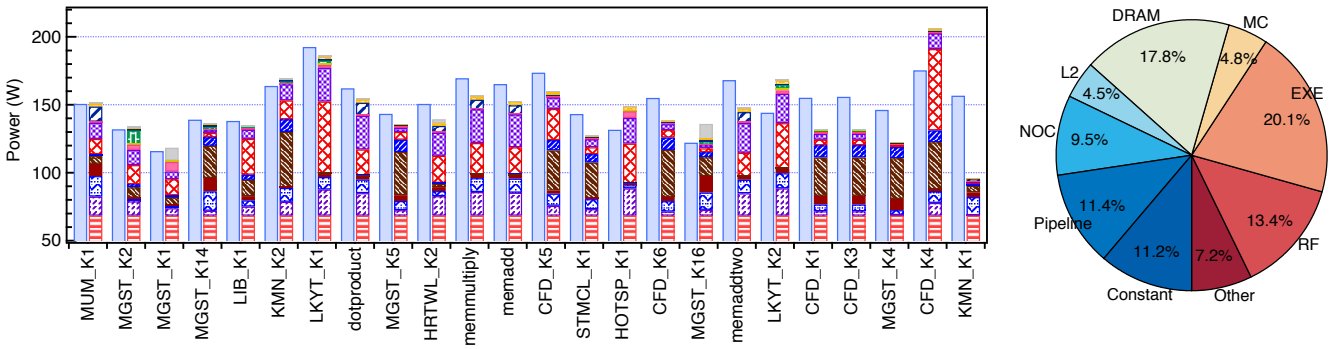


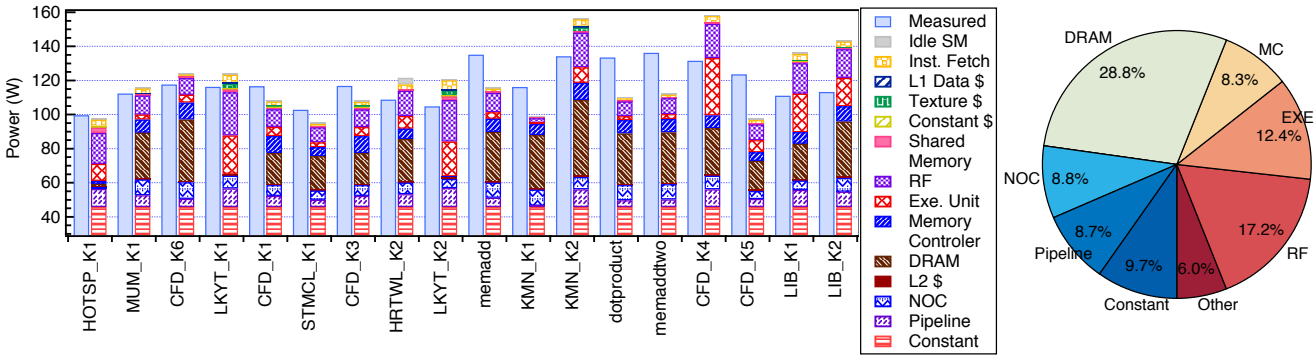
Figure 11: Simulated per-component dynamic power distribution. The values represent the percentage of power contributed by a given component for a microbenchmark.

The average absolute error for the microbenchmarks on the GTX 480 card is 15%. Because we designed each microbenchmark to isolate a specific component power consumption, the error in total power for the microbenchmarks serves as a good indicator of the exercised component's modeling error. A certain amount of inaccuracies for the Memory, DRAM, and Mix microbenchmarks come from the performance mismatches in the modeling of the memory hierarchy in GPGPU-Sim versus the targeted hardware. For example, previous work [10] indicates that the hardware adopts a hash function for memory address mapping, whereas the GPGPU-Sim simulation uses a linear address mapping. Such types of modeling errors are significantly amplified in the microbenchmarks because they are repeatedly performing a single or set of operations.

The averaged modeling error for the Quadro FX5600 card



(a) GTX 480. Bar chart (left) is sorted in order of accuracy, and pie chart (right) is derived by averaging data in the bar chart.



(b) Quadro FX5600. Bar chart (left) is sorted in order of accuracy.

Figure 12: Estimated average power comparison and breakdown for real benchmarks.

under the microbenchmarking tests is 16.2%. This GPU has a different implementation of the memory coalescing logic [30] from the simulator, which causes our memory microbenchmarks that were designed for the Fermi to perform poorly, thus resulting in the slightly larger modeling error.

Component-level Validation It is impossible for us to measure the component-level power consumption. We achieve the component-level modeling validation by isolating the component power when designing the microbenchmarks as explained in Section 3.2. If the targeted component consumes most of the total power, the error in the total power consumption is a good indicator of that component's modeling error. In the ideal case that the component consumes 100% of the total power, this component's modeling error is equivalent to the total power error.

We present our microbenchmarks' power isolation effect in heat-map (Figure 11), which shows estimated per-component *dynamic* power distribution for a representative set of the component-level power isolation microbenchmarks (by stressing a certain component). Each colored grid corresponds to the percentage of estimated dynamic power for that particular microarchitectural component (x-axis) in the designed microbenchmark (y-axis). For example, the execution unit for the functional microbenchmark in Figure 11 consumes 72.6% of the total dynamic power, with significantly less power consumed by the register file (Reg) and pipeline (Pipe).

Achieving component power isolation for the execution units and DRAM is easier. These two components consume 72.6% and 57.4% of the total power separately in their stress testing microbenchmarks. It is almost impossible to isolate L2 cache (L2-$) power from NOC power because each L2 cache access involves an NOC access. We also found that L1 caches (data, constant and texture) are hard to isolate. For instance, the L1 data cache (D-$) consumes 29.1% of the total power, whereas register file and execution units still consume 23.7% and 21.8% power separately. This extra power "overhead" is caused to some extent by the data dependencies (Figure 5) that we intentionally insert to prevent the compiler from optimizing away the microbenchmarking code. I In summary, the functional units and DRAM microbenchmarks achieve better power isolation on targeted components, thus total power error better indicates component modeling error. However, this does not mean other components have higher modeling errors. The reason is two fold. Firstly, we designed more complex patterns to exercise these components. Secondly, these components are majorly SRAM based arrays, which have less modeling uncertainty as compared to functional units and DRAM.

4.3.3 *Benchmark Based Validation*

We evaluate the power model on a set of benchmarks that are not used in the refinement process. We use 25 kernels from the 13 different benchmarks presented in Figure 12. There are 18 kernels from the RODINIA suite [7] and 3 from the ISPASS suite [4]. We also include four microbenchmarks presented in prior work [13]. MGST kernels are not shown for the Quadro FX5600 because they fail to run on the card.

Average Power Figure 12 shows the our power model accuracy compared to measured power along with component-level breakdown of power consumption for both the GPU cards. The results are sorted in order of modeling accuracy. For GTX 480, the power model achieves an average

error within 9.9%. Memory-intensive kernels, such as the first kernel in KMN (KMN_K1) and CFD_K4, show more error due to the mismatch in memory hierarchies between our performance model and the real hardware.

For the Quadro FX5600 card, the model obtains an average error of 13.4%. The results are weaker because GPGPU-Sim's performance correlation, in terms of execution time, is 90% for this card and 93% for the GTX 480. Moreover, the number of evaluated benchmarks for Quadro FX5600 is less than that used for GTX 480 because this card does not support the computation ability required by MGST.

Comparing the performance per watt, we deduce that the GTX 480 is 3× better than Quadro FX5600. GTX 480 consumes 1.33× the total power of the latter, but it is ∼4× faster in performance. The contributing factors include several changes in the GTX 480 for energy efficiency and performance, such as the presence of more functional units, increased SIMD width and the introduction of data caches. The only kernels that show a decrease in the overall performance per watt are KMN_K1, MUM_K1 and LIB_K1. They do not have enough data locality to utilize the caches effectively, and thus retrieve data more often from DRAM.

The pie charts in Figure 12 show the power breakdown of the modeled *dynamic* power consumption in the two cards. The chart is derived by averaging and normalizing all kernels' power breakdowns. Shared memory, texture, constant and data caches contribute much less than other components to the average dynamic power and are thus grouped into the "Other" label in the pie chart. The lower power consumption for these components are caused by few kernels using these structures. The increased number of functional units and SIMD width cause execution units to increase from 12.4% in Quadro FX5600 to 20.1% in GTX 480. Pipeline power increases from 8.7% to 11.4% because of the increased pipeline depth in GTX 480. GTX 480 has added both L1/L2 data caches, which reduces the DRAM and memory controller (MC) power consumption from 28.8% and 8.3% to 17.8% and 4.8% separately. Note that although GTX 480 has a larger register file (RF) size, the percentage of RF's power consumption actually increases because the total power consumption of GTX 480 increases more than the RF does.

Runtime Trace We also validate the dynamic behavior of our power model against the hardware measurements. This is an important step because a steady power trace can have the same average power as a trace that is oscillating around the steady state power. In Figure 13, we show the trace of five kernels from benchmarks LIB, CFD, and MGST. These kernels are run separately, and thus the time in the horizontal axis is not continuous. Moreover, we only show 1 ms of kernel execution time due to space constraints. The start time for all kernels but one (MGST_K1, discussed later) is 0.4 ms because data before this time includes the exponential increase due to the RLC effect (see Section 4.3.1).

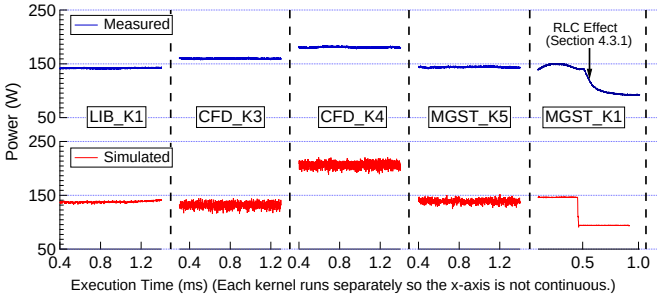


Figure 13: Measured and modeled power trace comparison.

The dynamic power profile in GPGPU applications can be categorized into two types. The first type has a steady power consumption, as seen in the first four kernels (from LIB_K1 to MGST_K5) in Figure 13. The second type is more like MGST_K1, which has phases in the power trace.

The power model tracks the steady state power consumption trend accurately, both across kernels from *within* a benchmark, as well as *across* kernels from different benchmarks. For example, the CFD_K4 kernel from the CFD benchmark has higher measured power (∼180 W) than the CFD_K3 kernel (∼160 W). LIB_K1 and MGST_K1 have the lowest power consumption (both below 150 W). Another thing to notice is that the simulator typically appears more "noisy" compared to the measured trace because the hardware RLC circuit forms a low-pass filter that smooths out the measured power trace. The simulator is in fact effectively capturing subtle variations in workload activity.

In MGST_K1, the dynamic power consumption drops past 0.5 ms due to load imbalance. Cores are idling in the second half of the kernel's execution. As shown in Figure 12, MGST_K1 has a relatively large portion of idle SM power, and the model accurately tracks this behavior in the temporal perspective. Note that this kernel's starting time is not 0.4 ms, as for other kernels, because it has only 1 ms execution, so the whole execution time is shown here. The exponential decay pointed out by the arrow is again caused by the RLC effect rather than any inaccuracies in our model.

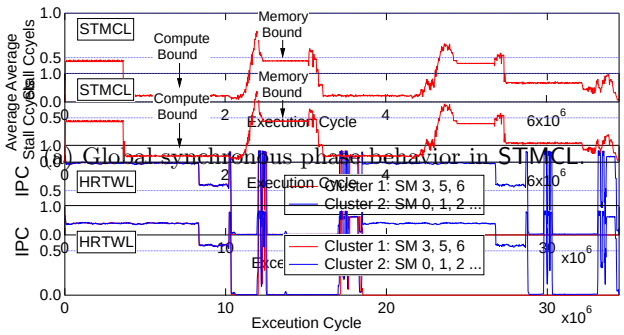
4.4 Power Model Extensions

On the basis of the methodology in Figure 1(a), the power model we have described thus far can adapt to newer architectures. Depending on the extent of the differences in the target architecture from the existing power model, full or partial iterations of the refinement and validation loop may be required. In this process, components may need to be added or removed from the power and performance simulators. Activity counters may need to be added that capture the behavior of the modified architecture, and additional microbenchmarks may also be necessary to stress and refine the new components.

Consider the example of adapting the power model from the Quadro FX5600 to the GTX 480 architecture. This extension requires the addition of new components, such as L1 and L2 data caches. Adding these components requires designers to systematically go through the process of initial modeling, microbenchmarking and iteratively refining and validating the power model. However, differences in the implementation of the execution units between the Quadro FX5600 and GTX 480 [23, 28] can easily be captured by the refinement and validation stages using the microbenchmarks targeted at the functional units.

5. ENERGY OPTIMIZATIONS

We find that a number of kernels have strong phase behavior. Thus, we demonstrate benefits using DVFS exploiting runtime memory and compute boundedness. Furthermore, we quantify the potential for fine-grained lane-level clock gating during branch divergence. Both techniques are based on the GTX 480 configuration in the GPGPU-Sim.



(a) Global synchronous phase behavior in STMCL.

(b) Clustered phase behavior in HRTWL.

Figure 14: Two types of phase behavior.

5.1 Exploiting Phase Behavior

We studied the opportunity for processor-level DVFS (i.e., single voltage and frequency value for the entire processor) within a kernel by tracking the average stall cycles in every SM caused by memory operations. We conducted the analysis at the individual SM level, which indicates that in most programs all the SMs are in global synchronous behavior. This is to be expected because programs suitable for GPU architectures are inherently optimized for throughput behavior. However, an interesting aspect of the global synchronous behavior is that the GPU kernels exhibit phase behavior. We also investigated the opportunity for cluster-level DVFS optimization, in which groups of SMs that behave similarly are clustered together so that a few voltage domains can allow for fine-grained control.

Figure 14(a) shows the averaged stall cycles during the execution of STMCL. Although the stall cycles shown in the plot are averaged across all SMs, each SM's behavior is almost identical due to the global synchronous behavior. The benchmark repeatedly shifts between compute- and memory-bounded phases every 150,000 cycles. Processor-level DVFS can leverage these memory-bounded phases in this case. However, in some GPGPU programs, SMs are not in global synchronous behavior. For example, benchmark HRTWL suffers from load imbalance. In this benchmark, all the SMs are assigned thread blocks to execute at the start of program execution. After the SMs finish executing the assigned thread blocks, there are no more blocks for further assignment to SMs 3, 5 and 6 (Figure 14(b)). These SMs remain idle but they consume non-negligible power.

We implemented a DVFS algorithm by monitoring the average stall cycles caused by memory operations, which is used to decide whether the kernel is memory bounded. In the case of global synchronized behavior, it is sufficient to monitor either a single SM, or averaged metric across all SMs. We assume that the GPGPU has 7 P-states, ranging from a peak of 700 MHz to a minimum of 100 MHz, with step of 100 MHz. These settings align with GTX 480's existing DVFS settings (see Section 4). We use the 45 nm predictive technology models [2] to scale the voltage with frequency (from 1 V to 0.55 V). The baseline architecture is always running at the highest frequency. We quantify DVFS benefits under two scenarios. First, we assume a fast responding on-chip regulator that can make P-state transitions quickly within 500 cycles, which are similar to those used by Kim *et al.* in prior work for CPUs [19]. Second, we consider a conventional off-chip regulator with coarser granularity of 10,000 cycles transition time, or roughly 10 μs.

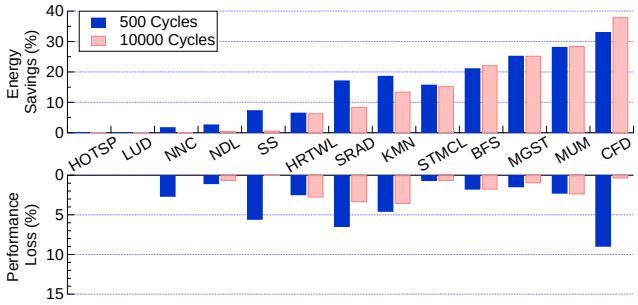


Figure 15: Energy savings and performance loss for fine-grained and coarse-grained processor-level DVFS.

Figure 15 shows the results of the processor-level DVFS algorithm for both settings. Benchmark STMCL from Figure 14a achieves 15.8% energy savings with only 0.7% loss in performance, assuming fast on-chip regulators. Even in the case of slow off-chip regulation, the energy savings is 15.3%, because the duration of each computation/memory bound phase is typically long enough that the advantage of fast-responding diminishes. The fine-grained DVFS scheme saves about 10% more energy than coarse-grained DVFS in benchmarks KMN, SS and SRAD. These programs have short phases that only on-chip regulation can effectively leverage. For benchmarks that are memory-bounded during the entire execution, such as BFS and MUM, both the DVFS schemes achieve more than 20% energy savings. Purely compute-bound benchmarks, such as LUD and HOTSP, do not benefit from DVFS. The worst-case performance loss is CFD in the fine-grained DVFS scheme because it has some rapid phase changes, which our simple prediction technique fails to capture. Overall, the algorithm achieves 14.4% and 13.2% savings in energy under both fine-grained and coarse-grained DVFS, with 2.9% and 1.3% performance loss, respectively.

We also evaluate the benefit of cluster-level DVFS for benchmark HRTWL in Figure 14(b). With the cluster-level DVFS, we set the frequency of the idle SMs to the lowest P-state point (i.e. 100 MHz and 0.55 V). The idle SM leakage power reduces to approximately 10% [2]. Cluster-level DVFS achieves 13.6% energy savings while processor-level DVFS achieves only 6.6%. To put the benefits of cluster-level DVFS in context, we compare it against ideal power gating, which achieves a 14% energy reduction. The cluster-level DVFS scheme does nearly as well as ideal power gating. However, there are several factors that designers must consider, such as modeling all of the additional overheads. Careful trade-off analysis for this optimization is beyond the scope of this paper.

5.2 Harnessing Branch Divergence

GPGPU programs suffer from branch divergence [12]. Figure 16 shows the amount of branch-divergence-related lane inactivity. The data is constructed by calculating the fraction of time when none (that is, idle), one to two, or three to four out of 32 total SIMD lanes are active, similar to [12]. For example, benchmark NN has only one or two lanes active during execution. Whenever branch divergence occurs, inactive execution lanes are idle but they still consume clocking and toggling logic dynamic power.

We propose to minimize the unnecessary toggling dynamic

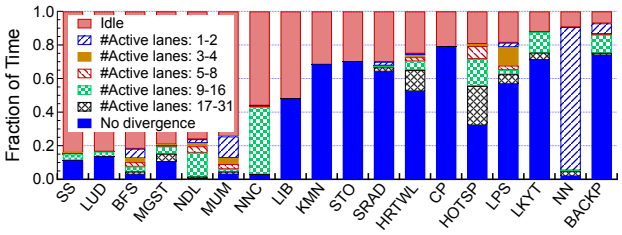


Figure 16: Time distribution of active lanes.

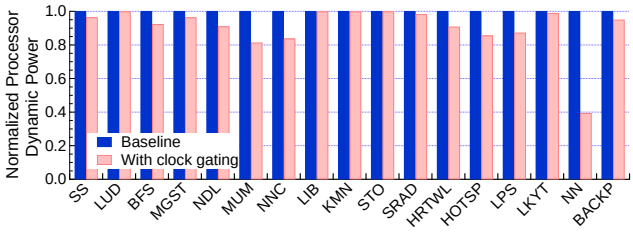


Figure 17: Dynamic power saving after clock gating.

power by applying clock gating to the unused, idle execution lanes. This technique is well-suited to exploit the short durations of branch divergence. We design microbenchmarks with different numbers of active threads to verify that neither card implements clock gating. To the best of our knowledge, there is no published literature that explores power reduction using fine-grained, lane-level clock gating for branch divergence in GPGPU architectures.

We implemented fine-grained clock gating in the power model to ensure that only SIMD lanes with active threads are active. In addition, RF reads/writes and operand collectors are also clock-gated at a fine-grained granularity. It is important to note that clock gating incurs power overhead owing to additional logic in the clock tree. We model these overheads using the approach proposed by Li *et al.* [21]. We assume an overall pipeline depth of 24 for the GPU architecture. Execution units, operand collectors and the crossbar network are clock gated in each SIMD lane level to prevent unnecessary switching activity, while the register files are gated in the register file bank level. For each of the 32 SIMD lanes and the 24 pipeline stages per lane, fine-grained clock gating requires an additional AND gate (to gate the clock) and an additional flip-flop to store the clock-gating control bit. The control bit ensures that the decision to clock gate a stage is available before the start of the cycle. The GPU maintains active lane masks for all warps, which can be used to determine which lanes should be clock gated.

We utilize the 45 nm TSMC technology library to estimate the power consumption of the additional flip-flops and logic gates. By our estimation, lane-level clock gating increases the GPU power consumption by 0.3 W. This power is consumed regardless of the lane-level activity. But as we will see next, this overhead is negligible relative to the net benefits.

In our experiment, we found the power saving of idle lane clock gating is proportional to the time of control divergence. Figure 17 shows the dynamic power saving of applying this technique. Note that we only count the dynamic processor (i.e., without leakage and DRAM power). Benchmark NN benefits the most (60.6% savings) from the clock gating, because it has only 1 or 2 lanes active for more than 90% execution time, as shown in Figure 16. Although benchmarks HOTSP and LPS both suffer severely from divergence, they only show 14.4% and 12.6% dynamic power savings, respectively, because they have more active lanes during divergence. Other benchmarks such as KMN, STO and SRAD have little or no divergence behavior, and thus show no improvement. Overall, the fine-grained technique achieves a (geometric) averaged 11.2% dynamic power savings.

6. RELATED WORK

There have been prior works regarding building a GPGPU power model [8, 13, 17, 26, 36, 38]. We compare our model against these works from the perspective of the power modeling methodology and the necessary validation effort.

Modeling Methodology Zhang *et al.* [38] showed that GPU power consumption increases linearly with the program's computation intensity, and they use this to predict the power consumption, similar to [36]. Maruyama *et al.* [26] use hardware performance counters to predict power. Hong and Kim [13] propose a power and performance prediction model for GPUs that predicts the GPU's execution time and power consumption using kernel and architecture characteristics. These modeling methodologies are good for program-level power prediction, but they lack our framework's *configurability* for modeling the power consumption of different GPU architectures. Moreover, although these previous works can only predict the GPU's average power consumption, our proposed model can provide *cycle-level* power trace of GPU programs, which lets us capture different execution phases within a kernel.

Validation Effort Hong and Kim [13] measure the power of the whole system for validation, whereas we isolate and measure the power that is consumed only by the processor. Additionally, by conducting the voltage and frequency scaling experiment mentioned in Section 4, we isolate the constant power component from our measured power to validate the static leakage power and dynamic power separately. We validated the dynamic power with a suite of microbenchmarks that contains 80 kernels that exercise different microarchitectural components. Moreover, with the 2 M/s sampling rate, we measured the power consumption of real benchmarks. The execution time for these benchmarks is typically around several milliseconds. We show the measured traces of real benchmarks and use those traces to validate the runtime power trace of our power model.

Prior work [14, 37] leveraged memory/compute phase behavior for CPU energy optimizations. We adopt a similar methodology to conduct phase analysis at the individual SM-level, and identify both global and clustered synchronous behavior across SMs, and use that to demonstrate the benefits of processor-level DVFS. Lee *et al.* [20] performed the oracle study for processor-level DVFS in GPUs, but they tried to maximize throughput given a power constraint, while we try to minimize the energy consumption. Moreover, we study DVFS opportunity within kernel execution, while their study emphasizes behavior across kernels.

The impact of fine-grained clock-gating on power consumption has been studied for CPUs [15] and some SIMD processors [31]. We exploit this technique to solve a GPU-specific problem (i.e., branch divergence).

7. CONCLUSION

The community requires a robust GPU power model. We demonstrate a configurable, cycle-level and validated power model for GPGPUs that can be used for architecture and software energy-efficiency studies. The robustness of the

power model is proven against the measured power of two commercial GPUs using a complete suite of both microbenchmarks and real benchmarks. The power model achieves the averaged absolute error within 9.9% for GTX 480 card, with 13.4% for Quadro FX5600, respectively, for our evaluated benchmark suite from RODINIA and ISPASS. We developed this model on the basis of a strong power-modeling methodology that will enable us to extend the existing power model systematically to support future GPU architectures.

Using *GPUWattch*, we show that fine-grained and coarse-grained DVFS are useful for reducing dynamic power consumption in GPGPU workloads, because they exhibit phase behavior. On average, coarse-grained DVFS achieves 13.2% energy savings, while fine-grained DVFS achieves 14.4% energy savings, both with less than 3% performance loss. For kernels with short-lived phases, fine-grained DVFS achieves 7% more energy savings than coarse-grained DVFS. In addition, we explore the opportunity for SM cluster-level DVFS to address the load imbalance problem in some GPGPU workloads. This technique reduces energy consumption further by 7% for benchmark HRTWL. Additionally, we evaluated lane gating. By clock gating individual SIMD lanes during branch divergence, we show that in some programs with high branch divergence, such as NN, this technique reduces dynamic power consumption by more than 60%.

8. ACKNOWLEDGEMENTS

We are extremely grateful to several people for helping us with this research undertaking, including the reviewers. We thank Steve Keckler and John Edmondson for helpful discussions on the challenges of power modeling of GPUs.

This work is supported by gifts from Intel Corporation and AMD, in addition to the support provided by the National Science Foundation grants CCF-1218474, CCF-0953603 and CNS-1217102. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

9. REFERENCES

[1] MacSim, http://code.google.com/p/macsim.
[2] Predictive technology model, http://ptm.asu.edu.
[3] Synopsys Inc., Power Compiler, www.synopsys.com.
[4] A. Bakhoda et al. Analyzing CUDA workloads using a detailed GPU simulator. In *ISPASS*, 2009.
[5] M. Bauer et al. CudaDMA: optimizing GPU memory bandwidth via warp specialization. In *SC*, 2011.
[6] D. Brooks et al. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*, 2000.
[7] S. Che et al. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, 2009.
[8] S. Collange et al. Power consumption of GPUs from a software perspective. In *ICCS*, 2009.
[9] W. J. Dally. Moving the needle, computer architecture research in academe and industry. In *ISCA*, 2010.
[10] J. M. V. Dyke et al. Graphics system with virtual memory pages and non-power of two number of memory elements, 2011.
[11] W. Fung and T. Aamodt. Thread block compaction for efficient SIMT control flow. In *HPCA*, 2011.
[12] W. Fung et al. Dynamic warp formation and scheduling for efficient GPU control flow. In *MICRO*, 2007.
[13] S. Hong and H. Kim. An integrated GPU power and performance model. In *ISCA*, 2010.
[14] C. Isci et al. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *MICRO*, 2006.
[15] H. Jacobson et al. Stretching the limits of clock-gating efficiency in server-class processors. In *HPCA*, 2005.
[16] T. Kailath, A. Sayed, and B. Hassibi. Linear Estimation. *Prentice Hall*, 2000.
[17] K. Kasichayanula et al. Power aware computing on GPUs. *SAAHPC*, 2012.
[18] S. Keckler. Life After Dennard and How I Learned to Love the Picojoule. In *MICRO*, 2012.
[19] W. Kim et al. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, 2008.
[20] J. Lee et al. Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling. In *PACT*, 2011.
[21] H. Li et al. Deterministic clock gating for microprocessor power reduction. In *HPCA*, 2003.
[22] S. Li et al. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.
[23] E. Lindholm et al. NVIDIA Tesla: A unified graphics and computing architecture. *Micro, IEEE*, 2008.
[24] J. E. Lindholm et al. Simulating multiported memories using lower port count memories, 2008.
[25] S. Liu et al. Operand collector architecture, 2010.
[26] H. Nagasaka et al. Statistical power modeling of GPU kernels using performance counters. In *Green Computing Conference*, 2010.
[27] V. Narasiman et al. Improving GPU performance via large warps and two-level warp scheduling. In *MICRO*, 2011.
[28] NVIDIA. *Fermi Compute Architecture Whitepaper*, 2009.
[29] NVIDIA. *Compute Visual Profiler - User Guide, Version 4*, 2011.
[30] NVIDIA. *NVIDIA CUDA C Programming Guide*, 2012.
[31] H.-J. Oh et al. A fully pipelined single-precision floating-point unit in the synergistic processor element of a CELL processor. *JSSC*, 2006.
[32] V. Sathish et al. Lossless and lossy memory-link compression techniques for improving performance of memory-bound GPGPU workloads. In *PACT*, 2012.
[33] S. Thoziyoor et al. A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. In *ISCA*, 2008.
[34] R. Ubal et al. Multi2Sim: A simulation framework for CPU-GPU computing. In *PACT*, 2012.
[35] T. Vogelsang. Understanding the energy consumption of dynamic random access memories. In *MICRO*, 2010.
[36] H. Wang and Q. Chen. Power estimating model and analysis of general programming on GPU. *Journal of Software*, 2012.
[37] Q. Wu et al. A dynamic compilation framework for controlling microprocessor energy and performance. In *MICRO*, 2005.
[38] Y. Zhang et al. Performance and power analysis of ATI GPU: A statistical approach. In *NSA*, 2011.