AccelWattch: A Power Modeling Framework for Modern GPUs

Vijay Kandiah
vijayk@u.northwestern.edu
CS & ECE, Northwestern University
USA

Scott Peverelle
scott.peverelle@intel.com
Intel
Canada

Mahmoud Khairy
abdallm@purdue.edu
ECE, Purdue University
USA

Junrui Pan
pan251@purdue.edu
ECE, Purdue University
USA

Amogh Manjunath
manjuna@purdue.edu
ECE, Purdue University
USA

Timothy G. Rogers
timrogers@purdue.edu
ECE, Purdue University
USA

Tor M. Aamodt
aamodt@ece.ubc.ca
ECE, University of British Columbia
Canada

Nikos Hardavellas
nikos@northwestern.edu
CS & ECE, Northwestern University
USA

ABSTRACT

Graphics Processing Units (GPUs) are rapidly dominating the accelerator space, as illustrated by their wide-spread adoption in the data analytics and machine learning markets. At the same time, performance per watt has emerged as a crucial evaluation metric together with peak performance. As such, GPU architects require robust tools that will enable them to model both the performance and the power consumption of modern GPUs. However, while GPU performance modeling has progressed in great strides, power modeling has lagged behind. To mitigate this problem we propose *AccelWattch*, a configurable GPU power model that resolves two long-standing needs: the lack of a detailed and accurate cycle-level power model for modern GPU architectures, and the inability to capture their constant and static power with existing tools. AccelWattch can be driven by emulation and trace-driven environments, hardware counters, or a mix of the two, models both PTX and SASS ISAs, accounts for power gating and control-flow divergence, and supports DVFS. We integrate AccelWattch with GPGPU-Sim and Accel-Sim to facilitate its widespread use. We validate AccelWattch on a NVIDIA Volta GPU, and show that it achieves strong correlation against hardware power measurements. Finally, we demonstrate that AccelWattch can enable reliable design space exploration: by directly applying AccelWattch tuned for Volta on GPU configurations resembling NVIDIA Pascal and Turing GPUs, we obtain accurate power models for these architectures.

CCS CONCEPTS

• **Hardware → Power estimation and optimization**; • **Computing methodologies → Modeling methodologies**; *Simulation tools.*

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480063

KEYWORDS

GPGPU/GPU Computing, Power Modeling and Simulation

ACM Reference Format:
Vijay Kandiah, Scott Peverelle, Mahmoud Khairy, Junrui Pan, Amogh Manjunath, Timothy G. Rogers, Tor M. Aamodt, and Nikos Hardavellas. 2021. AccelWattch: A Power Modeling Framework for Modern GPUs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3466752.3480063

1 INTRODUCTION

Graphics Processing Units (GPUs) are becoming increasingly popular for accelerating both general-purpose and High-Performance Computing (HPC) applications. There are 147 GPU-accelerated systems in the most recent TOP500 HPC list [43], and 70% of the top-50 HPC applications are GPU-accelerated [40]. Similarly, GPUs have become one of the dominant forces in machine learning and AI acceleration [11]. As the proliferation of GPUs grows to satisfy the demand for higher performance, they are fast becoming a major consumer of power. Thus, it is not surprising that performance per watt, together with peak performance, have emerged as indispensable metrics for evaluating the efficiency of GPU architectures. As such, GPU architects require robust tools that will enable them to quickly and accurately model both the performance and the power consumption of modern GPUs.

However, while GPU performance modeling has progressed in great strides [20], GPU power modeling has lagged. GPUWattch [21] has been an indispensable tool for modeling the power consumption of new innovations in GPU architectures, but it was designed to model (and validated against) older architectures with fewer energy efficiency optimizations.

Attempting to model recent GPUs such as Pascal [29], Volta [30] and Turing [31] using the methodology employed by GPUWattch produces significant inaccuracies, both in terms of absolute numbers and in terms of the relative power consumption of individual hardware components. This can lead to inadvertently optimizing components that may not be as important for energy efficiency in real systems as the model may allude. We find that a key source of errors is the lack of a model for Dynamic Voltage and Frequency

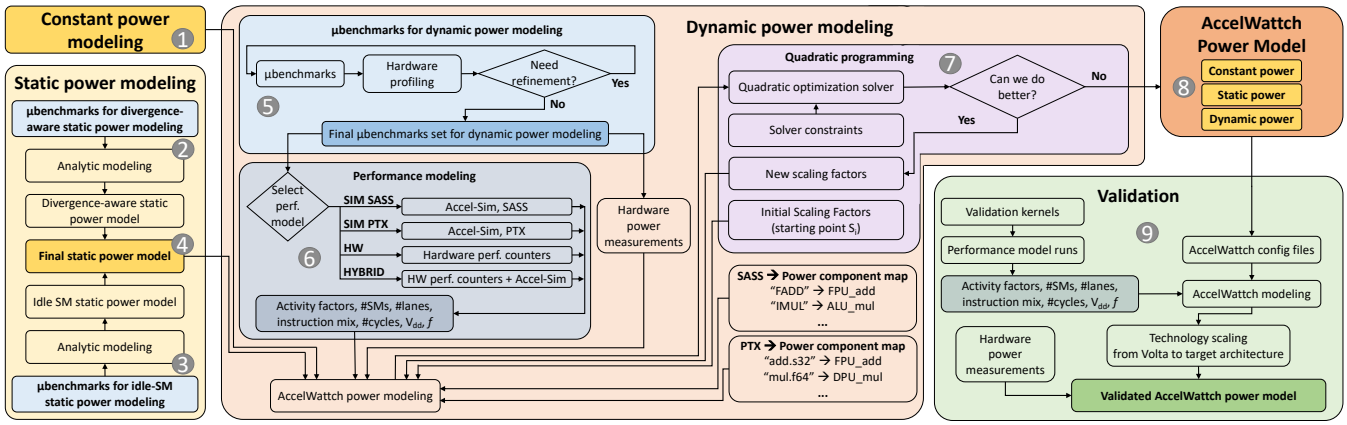


Figure 1: AccelWattch power modeling flowchart.

Scaling (DVFS). Lacking a DVFS model results in recent GPUs being reported to have a negative constant power term. Our insight, that power under V-F scaling is better modeled by a 3rd-degree polynomial missing a quadratic term (Section 4.2), allows AccelWattch to accurately estimate constant power.

Another key source of error is the lack of a model for capturing the power-down of hardware components, and their contribution to static power when powered-up but inactive. In the absence of such a model, static power is lumped into a single constant, an oversimplification for modern chips with aggressive power gating. We infer, for the first time to our knowledge, how modern GPUs power-gate chip-wide hardware components (e.g., L2 cache), Streaming Multiprocessor (SM)-wide components (e.g., L1 caches) and lane-specific components (e.g., FPUs). AccelWattch accurately models the effect of reactivating power-gated structures. It does so by capitalizing on our insights (Section 4.3) that: (1) activating the first SM powers up global chip components which leak when not switching; (2) activating the first lane of an SM powers up SM-wide components, which leak when inactive; and (3) activating subsequent lanes additionally powers up only those lanes' execution units. Our insight, that the simultaneous execution of operations within a warp (now supported by modern GPUs [30]) presents a counter-intuitive sawtooth pattern of power consumption (Section 4.4), allows AccelWattch to accurately model thread divergence in the presence of Instruction-Level Parallelism (ILP) (Section 4.5).

Some recently-proposed power models [2, 13, 47] target modern GPUs, but they, as well as earlier works [16, 50] are provided only as analytic models over average behavior, which hinders research that requires cycle-level accuracy (e.g., research on DVFS). Moreover, analytic models are hard to extend to describe novel architectural components; often it is easier to build a cycle-level model that emulates the component's behavior and use it for evaluation. To support cycle-level research, the computer architecture community needs a robust and configurable power modeling tool, capable of supporting cycle-accurate simulation.

We address the lack of cycle-level power modeling tools for modern GPUs by introducing *AccelWattch*, a new GPU power model that is configurable, capable of cycle-level calculations in emulation and trace-driven environments, and supports DVFS. To the best of our knowledge, AccelWattch is the only power model capable of modeling both PTX (virtual ISA) and SASS (native machine ISA) instructions, and the only open-source tool capable of modeling closed-source workloads with hand-tuned SASS instructions—it only needs a binary. In addition, AccelWattch is the only GPU power model that can be driven by either pure software performance models (e.g., Accel-Sim [20]), or hardware performance counters commonly found in modern GPUs (thereby capturing execution on real silicon), or a combination of the two. These AccelWattch variants allow researchers to balance the trade-off between power model accuracy and performance modeling effort.

We validate AccelWattch against hardware power measurements on an NVIDIA Volta GV100 [30] GPU running a suite of 26 kernels from NVIDIA CUDA Samples [35], Rodinia 3.1 [7], Parboil [41], and CUTLASS 1.3 [32] suites. AccelWattch yields a mean absolute percentage error (MAPE [9]) between 7.5–9.2±2.1–3.1%, depending on the AccelWattch variant, achieving a Pearson r coefficient of 0.83–0.91. These errors are a factor of 22–24× lower than GPUWattch's when targeting the same architecture. As a case study, we apply AccelWattch on kernels from DeepBench [25] workloads, and find that it obtains 12.79% MAPE over hardware power measurements despite the significant limitations of existing performance models. We demonstrate the reliability of AccelWattch for design space exploration by applying our validated AccelWattch Volta model (i.e., without retraining or needing new hardware measurements) to model the power of two GPU architectures: a Pascal TITAN X [29], and a Turing RTX 2060S [31]. AccelWattch accurately predicts the power consumption of these new architectures, achieving 11 ± 3.8% and 13 ± 4.7% MAPE, respectively.

In summary, we make the following contributions:

- For the first time to our knowledge, we infer and introduce an analytic model that explains and accurately captures constant, static, and dynamic power consumption in the presence of DVFS, thread divergence, intra-warp functional unit overlap, variable SM occupancy, and power gating.
- We introduce AccelWattch, a cycle-level constant, static and dynamic power model for the NVIDIA Volta GPU architecture. AccelWattch resolves long-standing needs for modern GPU architectures: the lack of a cycle-level power model, and

the inability to capture the constant and static power with existing methodologies. We validate AccelWattch and show it achieves high correlation to hardware measurements.
- To the best of our knowledge, AccelWattch is the only GPU power model that can be directed by emulation (PTX) or trace-driven (SASS) software performance models, or by hardware performance counters, or by a combination of the above. This allows for the study of discrete hardware components without the need to develop performance models of the entire architecture.
- We demonstrate that AccelWattch can enable reliable design space exploration. Directly applying the Volta power model on a GPU configuration resembling the Pascal and Turing architectures results in accurate power models for these architectures without tuning specifically for them.

2 ACCELWATTCH MODELING WORKFLOW

We follow the process shown in Figure 1 to develop a model that accurately estimates: (a) constant power, for example by board fans and peripheral circuitry, in the presence of DVFS ①; (b) static power in the presence of execution divergence, the simultaneous execution of operations within the same warp [30], variability in SM occupancy, and the power gating of lanes, SMs, and global chip hardware components ②–④; and (c) dynamic power consumption for each individual hardware component ⑤–⑧. To model dynamic power, we develop a suite of 102 microbenchmarks that isolate and stress the various components of a modern GPU ⑤. We use them, together with hardware power measurements and execution statistics ⑥, to bound any modeling inaccuracies using quadratic programming ⑦. AccelWattch is driven by a performance model, which provides AccelWattch with statistics on hardware component activity, active SMs and lanes, voltage-frequency parameters, and cycle count ⑥. We integrate AccelWattch with GPGPU-Sim [3] and Accel-Sim [20] to facilitate its use for both PTX [37] and SASS [36] simulations, producing the *AccelWattch PTX SIM* and *AccelWattch SASS SIM* variants, respectively.

AccelWattch can also be driven by hardware performance counters collected during execution on real silicon, either entirely (*AccelWattch HW*) or in combination with software-modelled ones (*AccelWattch HYBRID*). This allows for the study of discrete hardware components without the need to develop accurate software performance models for the entire architecture. Building comprehensive and accurate performance models for GPUs is a painstaking and time-consuming process. In the absence of sophisticated software performance models like Accel-Sim [20] for future GPUs, one can use hardware performance counters and execution on real silicon to model the power of a GPU architecture, and replace the hardware performance counters of the component targeted by the research with counters obtained from a model of only that component.

The AccelWattch framework can be used to estimate the power consumption of a kernel running on a new architecture by first initializing it with the AccelWattch model ⑧. Then, a performance model provides AccelWattch with the kernel's execution statistics (e.g., through simulation or hardware counters from execution on real silicon) ⑨. If needed, the resulting power estimates are scaled to a new technology node.

A salient feature of AccelWattch is its longevity. As architectures and technology continue to evolve, power modeling tools must adapt to match their target systems. A key feature of AccelWattch is that it is software-only; it makes use of integrated power monitoring tools in modern GPUs, and requires no external equipment beyond a GPU card. Our power modeling framework is equipped with a suite of microbenchmarks, analytical models, an optimization solver, and a validation methodology that can support future GPUs.

3 THE ARCHITECTURE OF NVIDIA VOLTA

GPUs execute programs known as "kernels", which comprise several threads, often thousands. Threads do not execute instructions independently; rather, sets of 32 threads (warps) execute the same instruction on different data by using their thread ID to select the data items to work on.

A Volta GV100 GPU chip consists of 80 Streaming Multiprocessors (SMs). Each SM is partitioned into four processing blocks [30], each with 16 INT32 cores for integer arithmetic, 16 FP32 and 8 FP64 cores for 32- and 64-bit floating-point, two tensor cores for matrix arithmetic, one special function unit (SFU) for complex operations (e.g., log), one warp scheduler, one dispatch unit, and a 64KB register file [30]. The INT32, FP32 and FP64 cores have adders, multipliers, and fused-multiply-add (FMA) units. The warp scheduler and dispatch unit can issue one instruction per clock to 32 execution lanes. Each of the 4 processing blocks per SM has 8 LD/ST units and a 12KB L0 instruction cache [19]. Each SM features a 128KB L1 data cache/shared memory, 2KB L1 and 64KB L1.5 constant caches, and a 128KB L1 instruction cache. The GPU has a 6144KB unified L2 cache at the chip level, and a 32GB GPU DRAM off chip [19].

4 CONSTANT, STATIC AND IDLE POWER MODELING

4.1 Hardware Experimentation Methodology

We use NVIDIA Management Library (NVML) [33] and the higher-level API, NVIDIA System Management Interface (nvidia-smi) [28] interchangeably for collecting power measurements on silicon for all of our experiments. If needed, we vary the processor frequency with nvidia-smi (e.g., for the constant power modeling experiments described in Section 4.2). When possible, we lock the processor frequency to the default applications clock frequency while collecting power measurements for microbenchmark and validation suite kernels. We ensure that the microbenchmarks present the desired behavior by profiling them using hardware performance counters provided by NVIDIA Nsight Compute [39].

To protect our experiments from the impact of temperature variations, we bring the GPU chip to 65^{o}C before taking power measurements for a target kernel. Temperature variability affects static power exponentially. Keeping a constant temperature during hardware measurements eliminates this noise. After AccelWattch learns a model assuming constant temperature, one can model temperature variations by multiplying the modeled static power with an experimentally-derived temperature-dependent factor.

4.2 DVFS-Aware Constant Power Modeling

At a high level, GPU power can be described by Eq. (1):

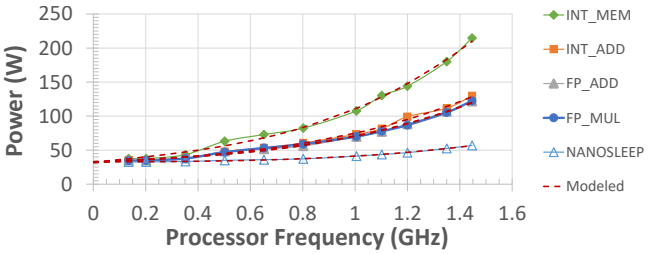


Figure 2: Measured and curve-fitted total power with varying processor frequency on GV100.

$$P_{total} = P_{proc,dyn} + P_{mem,dyn} + P_{proc,static} + P_{mem,static} + P_{const} \quad (1)$$

The terms $P_{proc,dyn}$ and $P_{mem,dyn}$ comprise the dynamic power consumption of the GPU chip and memory, respectively, and depend on the respective component's frequency, voltage and technology parameters (e.g., capacitance, gate length). The $P_{proc,static}$ and $P_{mem,static}$ terms comprise the static power consumption of the GPU chip and memory, respectively, and depend on voltage and technology parameters only. GPUs consume power not only by activating microarchitectural components (e.g., ALUs, caches) or through leakage currents at inactive components (static power), but also by peripheral components such as GPU board fans and other auxiliary support circuitry. We capture the power consumption of these components in the constant power term P_{const}. We rewrite Eq. (1) to reflect these dependencies and obtain Eq. (2), in which C refers to the gate capacitance, V the supply voltage, and f the clock frequency. The terms a, a', b, b', m, and n are constants that abstract away environmental, technology and design factors (a, b for GPU chip dynamic and static power; a', b' for memory):

$$P_{total} = aCV^2f + a'CV^2f + bV + b'V + P_{const} = mCV^2f + nV + P_{const} \quad (2)$$

The methodology employed by prior cycle-level models like GPUWattch [21] to estimate constant power is based on Eq. (2) and **does not work on recent GPUs** [13]. This methodology relies on scaling down the frequency f, which linearly reduces the first term (dynamic power). By running kernels at varying frequencies and measuring the GPU's power consumption each time, one could estimate this linear relationship. Extrapolating this line to $f = 0$ eliminates the mCV^2f term and leaves only the $nV + P_{const}$ term, providing an estimate of the static and constant power.

Modern GPUs employ DVFS to scale voltage with frequency and this invalidates the underlying assumptions of the above methodology. Fitting the experimental results from a GPU employing DVFS (like Volta) to a linear model (as in GPUWattch) results in a negative constant and static power estimate, which is clearly incorrect. Recently-published data for fully-realized processors show a near-linear relationship in the frequency-voltage curve [18, 51]. Hence, we can approximate the voltage scaling as a linear function of frequency, $V \approx kf$, and rewrite Eq. (2) as:

$$P_{total} = \beta Cf^3 + \tau f + P_{const} \quad (3)$$

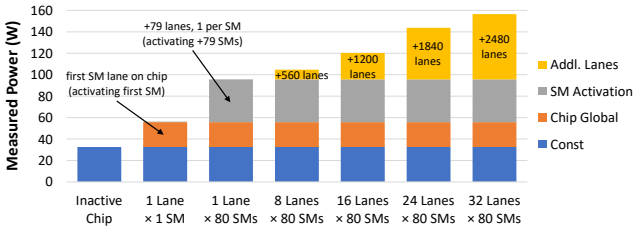


Figure 3: Inferring the power consumption of activating power-gated chip-wide and SM-wide components.

Thus, total power can be approximated by a cubic polynomial with a missing quadratic term (β, τ are constants). Armed with Eq. (3), we perform hardware power measurements at varying clock frequencies of kernels running on a Volta GV100 following the methodology at Section 4.1. We then curve-fit the experimentally-measured data on curves of the form of Eq. (3). Figure 2 shows the experimental results, along with the fitted cubic polynomials. We evaluate a mixture of high-power microbenchmarks (e.g., INT_MEM, which executes a mix of integer and memory operations and exceeds 200 W), light workloads (e.g., NANOSLEEP, which executes only nanosleep instructions), and moderate workloads (e.g., INT_ADD, FP_ADD, and FP_MUL which execute integer adds, FP adds, and FP muls, respectively).

The fitted polynomials of the form of Eq. (3) show strong correlation to the hardware power measurements (0.998 Pearson r coefficient). By extrapolating the fitted curves all the way to the y-axis intercept point, we can estimate the total power of the GPU when $f = 0$. That is, the y-intercept corresponds to P_{const}. Following this methodology, we estimate that the constant power P_{const} for a Volta GV100 is 32.5 W.

4.3 Power-Gating-Aware Static Power Model

After estimating constant power, i.e., P_{const} in Eq. (3), we next consider the second term, τf, which models static power. First, we turn our attention to modeling static power in the presence of power gating. We infer how modern GPUs are gating chip-wide, SM-wide and lane-specific hardware components. We measure the impact on power consumption of re-activating these components on real hardware, and introduce an analytic model that explains the power-gating behavior of GPUs and accurately captures their power consumption in the presence of power gating. To the best of our knowledge, this is the first time that the power-gating behavior of GPUs is inferred, measured and modeled analytically.

When lanes or SMs are inactive, modern GPUs gate them to conserve power. There are some components that all SMs share (e.g., L2 cache). These global chip components are powered up even when there is *only one* SM active on the GPU. When they are activated, these components leak power when they are not switching. Similarly, there are components that all lanes share in an SM (e.g., L1 caches, shared memory. These SM-wide components are active even when there is *only one* thread active in an SM, while the remaining lanes are power gated. These components also leak power when they are not switching. As additional lanes become active, they power up their own lane-specific functional units (e.g., INT32 and FP32 cores) which may also leak power when inactive.

Figure 3 shows the hardware-measured power of a microbenchmark that issues integer operations to a varying number of SMs and lanes per SM. When no SM is active on the chip, it consumes only constant power (estimated in Section 4.2). When the microbenchmark runs on only one lane on one SM (*1 Lane* × *1 SM*), that first SM activation powers up that SM's structures, but also activates global chip structures shared by all SMs. When the microbenchmark utilizes additional SMs (*1 Lane* × *80 SMs*), it additionally powers up only the SM-wide components of the additional SMs. As a result, the first activated SM on a GPU consumes 47× more power than each one of the subsequently-activated SMs. For example, *1 Lane* × *80 SMs* consumes 70% more power than *1 Lane* × *1 SM*, even though it utilizes 79× more SMs.

Similarly, the first lane activation in an SM also activates SM-wide structures that are shared by all lanes. In contrast, activating additional lanes also powers up only those lanes' functional units. As a result, the first activated lane in an SM demands 31× more power than lanes activated after it. For example, *8 Lanes* × *80 SMs* consumes 10% more power than *1 Lane* × *80 SMs*, even though it utilizes 7× more lanes.

The increased power consumption after re-activating a component is both due to higher dynamic power as the component is utilized (captured by our dynamic power model), as well as higher static power. We capture the latter in the analytic model we introduce in the next section, as the effects of power gating are inherently linked to the effects of execution divergence.

4.4 Divergence-Aware Static Power Modeling

When a warp executes in an SM, it may leave some lanes inactive due to execution divergence, which may be gated to conserve power. The active lanes, however, still leak power as not all of their components are continuously utilized. We capture this behavior of a warp with y active lanes in Eq. (4).

$$\begin{aligned} P_{static,addLane} &= (P_{static,32Lanes} - P_{static,firstLane}) \;/\; 31 \\ P_{static,yLanes} &= P_{static,firstLane} + P_{static,addLane} \cdot (y-1) \end{aligned} \quad (4)$$

The $P_{static,firstLane}$ term captures the static power of the first active lane, to which we attribute the static power of all the SM-wide components that lanes share. Each additional active lane is only responsible for its own functional units' static power, $P_{static,addLane}$. The $P_{static,32Lanes}$ term refers to the static power when 32 lanes are active. We refer to Eq. (4) as the *Linear static power model*, as it distributes equally the static power among all lanes of a warp except the first lane.

However, the linear model of Eq. (4) does not always match real-world observations. An SM in Volta comprises 4 processing blocks [30], each with 16 CUDA cores. A warp executes by running two 16-thread half-warps, one after the other. If a warp has $y \leq 16$ threads active, the processing block executes the active half-warp but forgoes the execution of the empty one. Thus, the same fraction of lanes is active on every cycle, and the power consumption rises as y grows. At $y = 16$, all 16 cores on all processing blocks are always active, consuming maximum power.

If a warp has $16 < y < 32$ active threads, then "full half-warps" with 16 active threads (maximum power consumption) alternate with "partial half-warps" with the remaining active threads (lower power consumption). Thus, the power consumption for $16 < y < 32$ will be lower than the power consumption for $y = 16$ (note that the energy consumption will still be higher). When $y = 32$, all processing clusters once again execute "full half-warps" and reach maximum power. We capture this counter-intuitive behavior in Eq. (5), to which we refer as the *Half-warp static power model*.

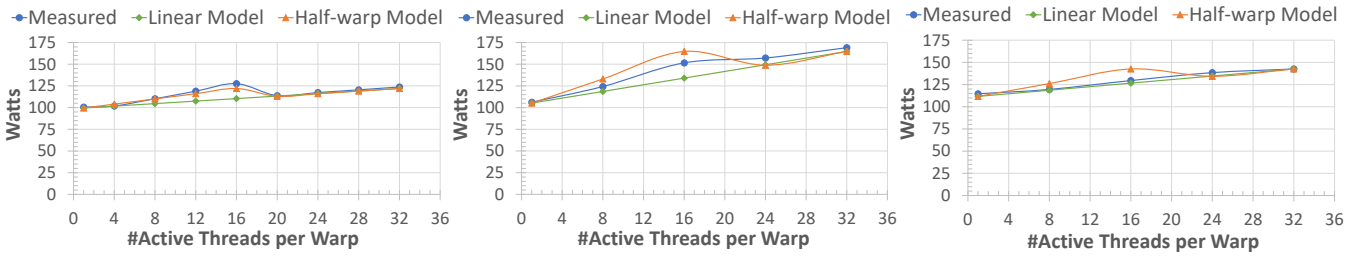
$$P_{static,yLanes} = \begin{cases} P_{static,firstLane} \\ \quad + P_{static,addLane} \cdot (y-1), & \text{if } y \leq 16 \\ \\ P_{static,firstLane} \\ \quad + \frac{1}{2} P_{static,addLane} \cdot 15 \\ \quad + \frac{1}{2} P_{static,addLane} \cdot (y-17), & \text{if } y > 16 \end{cases} \quad (5)$$

To study this behavior experimentally, we follow the process shown in Figure 1-② and the methodology described in Section 4.1. We develop microbenchmarks that utilize all SMs but with configurable thread divergence. We run each microbenchmark at varying clock frequencies and thread divergence, collect hardware power measurements, and curve-fit them to Eq. (3) (the fitted curve has 1% MAPE). From the fitted Eq. (3) for each microbenchmark, we estimate its static power (fitted τf term) when only one lane is active per warp ($P_{static,firstLane}$), and when 32 lanes are active per warp ($P_{static,32Lanes}$). Then, we replace Eq. (3)'s τf term with the linear and the half-warp models from Eqs. (4) and (5) to obtain an analytic model for total power that is divergent-aware.

We validate the half-warp power model in Figure 4a, which compares the power estimated by AccelWattch with hardware power measurements for a microbenchmark that issues INT_MUL instructions. The power for this microbenchmark strongly follows the half-warp model. We emphasize that the blue line in Figure 4a represents hardware measurements. It is indeed the case that 16-lane and 32-lane warps consume the maximum power on real hardware, and all other configurations consume less, giving rise to a sawtooth pattern. However, other cases (e.g., Figure 4c) follow the linear model instead. The reasons behind this behavior are discussed next.

4.5 ILP and Execution Divergence

When a kernel uses only one functional unit, power strongly follows the half-warp model (Figure 4a). When exercising two units, (e.g., when the kernel issues both INT32 and FP32 instructions—Figure 4b), the half-warp behavior is less pronounced. This happens because Volta can simultaneously execute operations in the same warp by running multiple functional units concurrently [30]. A kernel with ILP can exploit this behavior to execute faster. As different operations usually have different latencies, their executions become interleaved in time. Hence, on every cycle we observe a statistical mix of full and partial half-warps: if we take a snapshot of the GPU, we will observe processing blocks executing "full half-warps" (Section 4.4) of one instruction concurrently with "partial half-warps" of the other. This statistical mix smooths out the sawtooth-like pattern of power consumption. When more units are employed (Figure 4c) the behavior becomes almost purely linear. Thus, static power for active SMs gradually drifts from the half-warp to the linear model, depending on the instruction mix. To the best of our knowledge, this is the first time this behavior is inferred and modeled.



(a) INT_MUL (b) INT_FP (c) INT_FP_SFU

Figure 4: Hardware measurements and modeled power with varying number of active threads in each warp.

Capitalizing on this observation, we assess the typical instruction patterns in GPU kernels and develop microbenchmarks that selectively stress them. We identified a total of 9 instruction mix categories, ranging from homogeneous categories with only integer ADD or only integer MUL instructions, to categories comprising a mix of instructions: int, int/FP, int/FP/DP, int/FP/SFU, int/FP/TEX, int/FP/tensor, and a category of only light instructions (e.g., nanosleep). We create the appropriate half-warp or linear models for each instruction mix and integrate them in AccelWattch. During an AccelWattch run, the performance model (simulator or hardware counters) reports the lane occupancy and instruction mix to AccelWattch, which then picks the appropriate power model.

4.6 Power Modeling for Idle SMs

Following a similar methodology, shown in Figure 1-③, we develop a model that captures the power consumption of SMs that are idle. We follow the methodology in Section 4.1 to develop microbenchmarks that vary the number of active SMs but use all 32 lanes of each warp (so thread divergence does not perturb our results). For simplicity, we assume that all SMs contribute equally to power consumption when they are occupied with the same microbenchmark. Thus, we estimate the dynamic plus static power per active SM when running microbenchmark i, $P_{dyn+static,perActiveSM,i}$, through Eq. (6), where $P_{total,80SMs,i}$ is the hardware power measurement of microbenchmark i with all SMs active (GV100 has 80) and P_{const} is the constant power estimated in Section 4.2.

$$P_{dyn+static,perActiveSM,i} = (P_{total,80SMs,i} - P_{const}) \;/\; 80 \quad (6)$$

When the same microbenchmark i is configured to occupy fewer SMs, $N_{activeSMs}$, we still expect each active SM to expend the power shown by Eq. (6). With that in mind, Eq. (7) models the power of all idle SMs (combined), where P_{total} is the hardware power measurement of that experiment.

$$P_{idleSMs,i} = P_{total,i} - P_{const} - P_{dyn+static,perActiveSM,i} \cdot N_{activeSMs} \quad (7)$$

We model the static power consumption per idle SM for microbenchmark i as a linear model in which each idle SM contributes equally: $P_{perIdleSM,i} = P_{idleSMs,i}/N_{idleSMs}$. We repeat this process for all n microbenchmarks, and use the geomean in Eq. (8) as the final estimate of idle SM power.

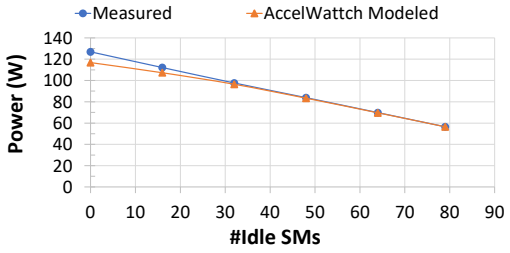


Figure 5: Validation of Idle SM static power model.

$$P_{perIdleSM} = \sqrt[n]{\prod_{i=1}^{n} P_{perIdleSM,i}} \quad (8)$$

Figure 5 shows that AccelWattch exhibits strong correlation with hardware measurements of the total power when running the INT_MUL microbenchmark, validating our model.

4.7 Putting It All Together

We combine Eq. (4), Eq. (5), and Eq. (6) and model the static power per active SM with y active lanes per warp in Eq. (9). The term $P_{static,yLanes,80SMs}$ is identical to $P_{static,yLanes}$ in Eqs. (4) and (5); we just make it explicit that the term is for 80 SMs.

$$P_{static,yLanes,perActiveSM} = P_{static,yLanes,80SMs} \;/\; 80 \quad (9)$$

Taking all of the above into account, AccelWattch's overall power model is shown in Eq. (10).

$$P_{total,yLanes,kSMs} = P_{dyn} + P_{static,yLanes,perActiveSM} \cdot k + P_{perIdleSM} \cdot (80 - k) + P_{const} \quad (10)$$

The two middle terms of Eq. (10) comprise AccelWattch's final static power model (Figure 1-④) for y active lanes and k active SMs. The term P_{dyn} corresponds to the dynamic power, and is the subject of the next section.

5 DYNAMIC POWER MODELING

5.1 Dynamic Power Model Formulation

AccelWattch employs an iterative approach to tune its parameters for dynamic power modelling, similar to GPUWattch, but it uses quadratic programming [4] instead of a least-squares solver. Given N microarchitectural components, the dynamic power consumed

by a kernel can be described by Eq. (11) as a function of each component's i energy per access E_i, its activity factor a_i (i.e., the number of accesses to it during execution), and the run time $T_{elapsedTime}$.

$$P_{dyn} = \sum_{i=1}^{N} \frac{a_i \cdot E_i}{T_{elapsedTime}} \tag{11}$$

The initial estimate $\hat{E_i}$ of component i's energy consumption per access is likely to be inaccurate. We consider this inaccuracy as an unknown variable x_i in equation $E_i = \hat{E_i} \cdot x_i$ and rewrite Eq. (10) for y active lanes and k SMs as:

$$P_{est.} = \sum_{i=1}^{N} \frac{a_i \cdot \hat{E_i}}{T_{elapsedTime}} \cdot x_i + P_{static,yLanes,perActiveSM} \cdot k + P_{perIdleSM} \cdot (80 - k) + P_{const} \tag{12}$$

One can view Eq. (12) as a dot product between a vector of power coefficients $\hat{P_i} \cdot x_i$, ..., $P_{static,activeSM} \cdot 1$, $P_{idleSM} \cdot 1$, $P_{const} \cdot 1$ (which remain constant for a given GPU) and activity factors, a_i, y and k, which vary across kernels. These vectors have $N + 3$ dimensions. N components of the vectors relate to estimated dynamic power, where each component contains an unknown factor, x_i. The three remaining components (modeled in Section 4) relate to constant, static, and idle SM power (these components effectively have $x_i = 1$). A single kernel will produce a vector with $N + 3$ activity factors, which can be used to estimate that workload's power consumption on the GPU. A collection of M kernels will provide $M \times (N + 3)$ set of equations, shown in Eq. (13). In this equation, the initially-inaccurate power estimates for the power components in $P_{est.}$ are corrected by the parameter vector X, to obtain a power for each kernel that approximates its hardware power measurement, $P_{meas.}$.

$$P_{est.}^{M \times (N+3)} \times X^{(N+3) \times 1} = P_{meas.}^{M \times 1} \tag{13}$$

Given enough workloads M, we can solve the system of Eq. (13) and obtain the best estimates X^*. We model dynamic power as an equation system with 22 parameters (Table 1). We tune these parameters using 102 microbenchmarks, each stressing specific microarchitectural components (Table 2), producing a 102×22 set of linear equations. Eq. (14) formulates the complete power model as an optimization problem, which AccelWattch solves using quadratic programming [4].

$$\begin{aligned} X^* = \arg\min_X \left(X^T \times P_{est.}^T \times P_{est.} \times X - (P_{est.}^T \times P_{meas.})^T \times X \right) \\ s.t.\ \forall i : 0.001 \le X_i \le 1000 \ \wedge\ X_{static} = X_{idleSM} = X_{const} = 1 \\ X_{ALU} \le X_{FPU} \le X_{DPU} \ \wedge\ X_{ALU} \le X_{imul} \\ X_{fpmul} \le \{X_{imul}, X_{dpmul}, X_{sqrt}, X_{log}, X_{sin}, X_{exp}, X_{tensor}, X_{tex}\} \end{aligned} \tag{14}$$

Table 1 lists the dynamic power components (corresponding components of X) modeled by AccelWattch. Each INT32, FP32, and FP64 unit in Volta can perform additions, multiplications, FMAs, and a few other operations. Each of these operations activates different parts of the functional unit's circuitry, which results in a different power consumption per operation. Thus, AccelWattch tracks these operations separately. Similarly, AccelWattch tracks SFU operations separately (e.g., log, sqrt). The L2 Cache and NoC components cannot be distinguished from each other, hence we model them together (similarly for DRAM and Memory Controller).

The AccelWattch HW model collects the information shown in Table 1 from hardware counters, except for the shaded components. There are no hardware counters for L1i and register file activity, and while DRAM read and write counters exist, there is no DRAM precharge counter. AccelWattch HYBRID is built similarly, but with the L2 Cache and NoC counters derived from Accel-Sim simulations.

5.2 Performance Modeling Framework

We developed AccelWattch by extensively modifying the dormant McPAT-based [22] GPUWattch [21] power model that comes packaged with the underlying GPGPU-Sim v4.0.1 [3] integrated in Accel-Sim v1.1.0 [20]. AccelWattch is driven by a performance model, which provides AccelWattch with statistics on hardware component activity, active SMs and lanes, voltage-frequency parameters, and cycle count (Figure 1-⑥).

For the performance model of AccelWattch variants driven fully or partially by software simulations (SASS SIM, PTX SIM, HYBRID) we use the latest publicly-available version of Accel-Sim v1.1.0 [20]. Accel-Sim has been extensively validated against Volta, showing strong performance correlation with > 0.97 Pearson r coefficient. For SASS simulations we feed Accel-Sim with SASS traces generated by the NVIDIA Binary Instrumentation Tool (NVBit) [45]. For PTX simulations, Accel-Sim invokes the underlying GPGPU-Sim.

At each sampling period (500 cycles) Accel-Sim provides execution statistics to AccelWattch, which uses them to estimate the workload's power for each sampling period. As the performance model provides AccelWattch with frequency and voltage settings at each sampling interval, AccelWattch can scale the estimated power for that interval following Eq. (2). Thus, if the performance model is DVFS-capable, AccelWattch will calculate all power transitions.

Similarly, for the AccelWattch variants driven by hardware (HW and HYBRID), AccelWattch collects hardware activity and execution statistics from kernel runs on real silicon using hardware counters provided by NVIDIA Nsight Compute [39]. For these models, AccelWattch collects dynamic instruction information and lane activity from the SASS traces (which are also obtained from execution on real silicon). For AccelWattch HYBRID, we follow the same methodology as for AccelWattch HW for all available hardware counters, but we utilize the methodology for AccelWattch SASS SIM to obtain the activity counters for the L2 Cache and NoC.

5.3 Microbenchmarking for Dynamic Power

Following the process shown in Figure 1-⑤ and the methodology in Section 4.1, we build a suite of 102 microbenchmarks that stress target hardware components to estimate their power consumption. We use a mixture of compiler options, inline-assembly (PTX), and pointer-chasing (for microbenchmarks that stress the memory hierarchy) to work around default compiler optimizations. We place the Region of Interest (ROI) of these kernels inside an unrolled loop, and run them on silicon with a high loop iteration count.

Table 1 lists the 22 microarchitectural components that AccelWattch tracks. Table 2 combines them into coarser-grain categories and lists the number or microbenchmarks that target components

Table 1: Dynamic power components in AccelWattch.

AccelWattch Dynamic Power Component	Hardware Unit on Volta	AccelWattch Dynamic Power Component	Hardware Unit on Volta
Instruction Buffer	L0 Inst. Cache	sqrt	SFU
Instruction Cache	L1i	log	
Constant Cache	Constant Cache	sin/cos	
L1d Cache	L1d Cache/ Shared Memory	exp	
Shared Memory		Tensor Core	Tensor Core
Register File	Register File	Texture Unit	Texture Unit
ALU	INT32 core	Scheduler	Sched. & Dispatch
int mul/mad		SM Pipeline	SM Pipeline
FPU	FP32 core	L2 Cache	L2 Cache
fp mul/mad		NoC	NoC
DPU	FP64 core	Dram	DRAM
dp mul/mad		Memory Controller	Memory Controller

Table 2: AccelWattch tuning μBenchmarks.

Hardware Comp. Category	μBench Count	Hardware Comp. Category	μBench Count
Active/Idle SMs	12	Register File	1
INT32 core	9	dCaches + Sh.Mem. + NoC	11
FP32 core	8	DRAM + MC	2
FP64 core	8	Tensor core	6
SFU	9	Mix	29
Texture Unit	7	Other (L0, L1i, Pipeline, Scheduler)	102

within each category. The *mix* microbenchmarks target combinations of these categories. The *Other* category includes components such as the SM pipeline, scheduler, and L0 and L1i instruction caches. All microbenchmarks stress this category, so all are included.

Figure 6 shows the dynamic power heat-map of microbenchmarks based on the hardware component categories they target. Each cell's color encodes the fraction of dynamic power a microbenchmark spends on the corresponding GPU component, as estimated by AccelWattch SASS SIM. It is important to note that even if a hardware unit is stressed by continuously issuing instructions to it, other units may also be accessed with a high frequency (e.g., register file, L1i), and the power consumed by these other units may be higher than the targeted one's. In such cases, the heat-map cell corresponding to the targeted unit may not appear "hot". For example, this behavior appears with units that do not consume much power, e.g., the Texture Unit, or with microbenchmarks that stress the INT32 Core with highly-divergent code. Overall, the heat-map demonstrates that each microbenchmark exercises the corresponding GPU component it targets, and our suite adequately exercises all components.

To build the power model of an AccelWattch variant (Section 2), we collect each hardware component's activity when running a microbenchmark from the respective performance model, and generate a set of per-component power estimates. These estimates, together with hardware measurements of total power consumption, are used in the quadratic programming optimization of Eq. (14).

5.4 Quadratic Programming Optimization

We perform quadratic programming optimization to minimize the relative error between the modeled system power and the measured hardware power (Eq. (14)). At each step of the regression, we obtain new per-component scaling factors that we supply to AccelWattch and re-iterate, until the solver can no longer reduce the relative errors (Figure 1-⑦). We enforce per-scaling-factor constraints on

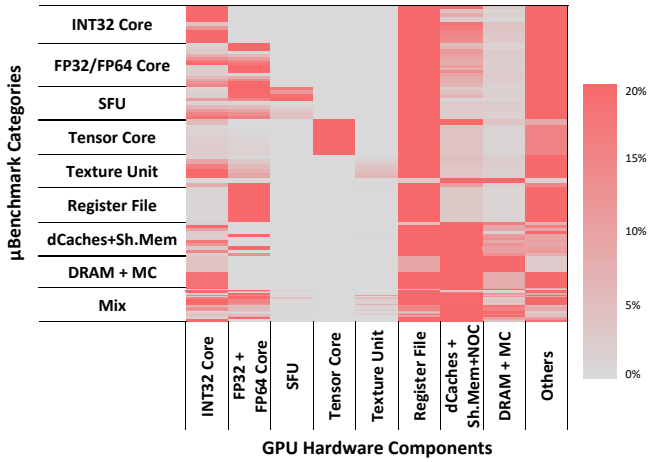


Figure 6: Dynamic power heat-map of GPU hardware component categories exercised by microbenchmarks.

the quadratic solver to guard against unrealistic component power estimates. In particular, we ensure all scaling factors are positive, and we constrain the energy cost of execution units to guard against unrealistic estimates (see constraints in Eq. (14)).

We use two different starting points for the scaling factors used in the initial iteration of this process. For one of the starting points, all initial scaling factors are set to one, thus there is no scaling taking place initially. The other starting point is obtained from the GPUWattch model for NVIDIA Fermi GTX 480 [27] which has been independently validated [21]. Due to having two starting points, we end up with two AccelWattch models. The model obtained from the Fermi starting point achieves higher accuracy (9.2% vs. 14.8% MAPE on the validation set for SASS SIM). Thus, for each AccelWattch variant, we adopt the model obtained from the Fermi starting point as the final AccelWattch model (Figure 1-⑧).

6 VALIDATION

6.1 Target Architecture and Workloads

We validate AccelWattch against a Volta GV100 GPU (Table 3) by applying the AccelWattch power models on a suite of validation kernels that are not part of the training set.

Our validation suite consists of a wide range of kernels selected from NVIDIA CUDA Samples (SDK) [35], Rodinia 3.1 [7], Parboil [41], and CUTLASS 1.3 [32]. Including NVIDIA SDK is important for the unbiased evaluation of AccelWattch. It does not skew results to AccelWattch's favor: AccelWattch SASS SIM achieves 9.91% MAPE for NVIDIA SDK, compared to 4.9% (Parboil), 9.55% (Rodinia), and 9.98% (CUTLASS). On the contrary, AccelWattch experiences its largest error in one of the SDK workloads (dct_k2). All workloads are compiled with NVCC V11.0.167 [34] with compute-capability support for Volta (`code=sm_70`).

To collect hardware power measurements, we launch the target validation kernel to the default CUDA stream repeatedly in a loop so that it runs sufficiently long for accurate hardware measurements using NVML. We ensure that the kernel runs for NVML's entire sampling period through nvidia-smi. We wait until the chip reaches 65°C, then collect several power measurements. We let

Table 3: Target GPUs for validation and case studies.

GPU	Tech. Node	Clock Frequency for HW Power Measurement	Power Limit	Case Study?
Quadro GV100 (Volta)	12 nm	1417 MHz	250 W	N
TITAN X (Pascal)	16 nm	1470 MHz	250 W	Y
RTX 2060S (Turing)	12 nm	1905 MHz	175 W	Y

Table 4: List of kernels in validation suite.

Kernel	Run-time Coverage	Benchmark	Kernel	Run-time Coverage	Benchmark
CUDA Samples 11.0					
tensor_K1	100%	cudaTensorCoreGemm	dct_K1	19.6%	dct8x8
binOpt_K1	100%	BinomialOptions	dct_K2	72.3%	dct8x8
walsh_K1	47.8%	fastWalshTransform	histo_K1	52.9%	histogram
walsh_K2	49.4%	fastWalshTransform	msort_K1	71.8%	mergesort
qrng_K1	66.4%	quasirandomGenerator	msort_K2	26.3%	mergesort
qrng_K2	33.6%	quasirandomGenerator	sobol_K1	100%	SobolQRNG
Rodinia 3.1					
kmeans_K1	91.6%	kmeans	sradv1_K1	53.9%	sradv1
bprop_K1	75.7%	backprop	hspot_K1	100%	hotspot
bprop_K2	24.3%	backprop	b+tree_K1	48.5%	b+tree
pfind_K1	100%	pathfinder	b+tree_K2	51.5%	b+tree
CUTLASS 1.3 (cutlass-wmma)			Parboil		
cutlass_K1	100%	input: 2560x16x2560	sgemm_K1	100%	sgemm
cutlass_K2	100%	input: 4096x128x4096	mri-q_K1	100%	mri-q
cutlass_K3	100%	input: 2560x512x2560	sad_K1	95.9%	sad

the chip cool down back to its idle-state temperature, repeat at least 5 times, and report the average across all measurements. We observe that measurements at the NVML resolution frequency are stable throughout kernel execution (0.0018–1.9% variance across all measurements and repetitions). If a target kernel cannot reach 65°C, we use a power-hungry kernel first to heat up the chip to much higher than 65°C, and then switch to the target kernel and collect power measurements at 65°C as the chip cools down.

NVML has a low sampling frequency of 50-100 Hz. Thus, we are unable to collect accurate hardware power measurements of short-running kernels ($< 2\mu s$ run time) because their measurements are perturbed by other events (e.g., invocation/setup overheads to prepare the next kernel iteration, host synchronization, PCI transfers). We exclude such kernels from our suite. We also exclude kernels that are impractical to simulate (> 2 days per run) due to exceedingly-long simulation times and multi-TB instruction trace storage requirements. All kernels are run to completion.

Table 4 lists our full evaluation suite of 26 kernels from 18 workloads along with their run-time coverage of the respective workloads they are from. We use the largest available input configuration for all workloads except CUTLASS, for which, we use three different input matrix sizes. We exclude CUTLASS, *hotspot*, and *pathfinder* from the PTX SIM validation suite because they do not compile for Accel-Sim's PTX mode We exclude *pathfinder* from the HW and HYBRID validation suites because NVIDIA Nsight Compute fails to provide hardware performance counters for this workload.

To validate AccelWattch, we follow the flow in Figure 1-⑨ and discussed in Section 2, with a NVIDIA Volta GV100 GPU as the target architecture. We compare AccelWattch's power estimates with hardware measurements obtained for the validation suite kernels running on a GV100 GPU.

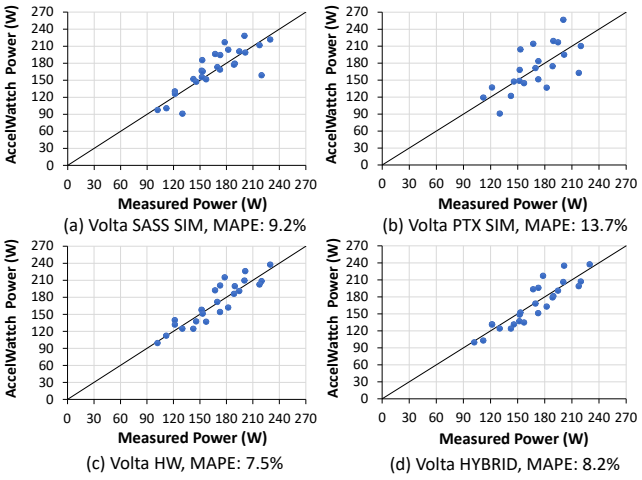


Figure 7: Correlation plots for AccelWattch validation.

6.2 Validation Results

Figure 7 shows the correlation plots of the estimated vs. measured power consumption and the corresponding MAPE for AccelWatch SASS SIM and PTX SIM for Volta. Overall, AccelWattch exhibits stronger correlation for SASS than PTX. We attribute this to the fact that PTX instructions do not map 1:1 to SASS instructions; prior work [14] demonstrates that simulating a virtual ISA (PTX) introduces inaccuracies compared to simulating the native ISA (SASS), which directly corresponds to execution on hardware units.

AccelWattch SASS SIM modeling a Volta GV100 compared to hardware measurements attains a MAPE of 9.2% ± 3.12% (95% confidence interval) with a maximum relative error of 30%. Two thirds of our validation suite kernels (17 out of 26) have < 10% absolute relative error, while only 4 kernels have an absolute relative error of > 20%. In comparison, when modeling a Volta architecture, GPUWattch reaches a MAPE of 219% with maximum error 447% (Section 7.3). For the two GPUs it is trained for (GTX 480 and Quadro FX5600), GPUWattch achieves average error of 9.9% and 13.4%, respectively, but with a maximum relative error of 57.8%. Overall, AccelWattch successfully tracks the high variability in measured power across our validation suite (from 90 W up to 230 W).

Among all variants, AccelWattch HW achieves the lowest MAPE (7.5%). This is expected, as it is driven by performance counters collected from execution on real silicon, and does not suffer from the inevitable inaccuracies of software performance models (e.g., Accel-Sim). However, AccelWattch HW is the most restrictive model, as it does not lend the same modeling flexibility as software simulators.

AccelWattch HYBRID is introduced as a way to alleviate this problem. The HYBRID variant utilizes hardware performance counters for all components except the ones that the user decides to replace with their own models. Thus, users may partially avoid the inordinate effort required to build, tune and validate highly sophisticated software models of entire GPU architectures, and instead focus their attention on only the few components that are relevant to their research target. As an example of a HYBRID model in this paper, we target the L2 and NoC hardware components and replace their statistics with ones obtained from Accel-Sim. AccelWattch

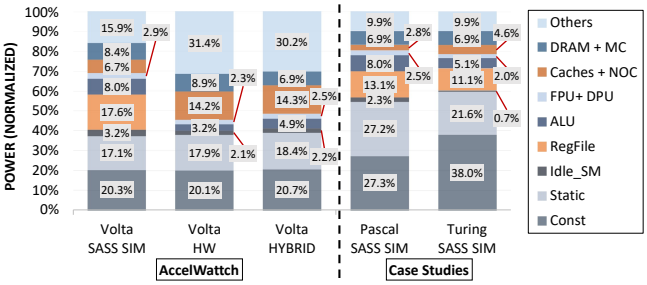


Figure 8: Normalized per-component power breakdown.

HYBRID achieves a MAPE of 8.2%, showing that it can successfully trade-off accuracy for modeling flexibility.

Figure 8 (left) shows the normalized power breakdown estimated by AccelWattch for Volta averaged across all kernels in the validation set. The register file, static power, and constant power are the most significant power contributors, together consuming 55% of the total system power on average. Volta also has a measurable *Idle_SM* power component, owing to having a high number of SMs which our validation suite kernels do not always fully utilize. The *Others* category comprises of the instruction buffer, scheduler, SM pipeline, texture units, and tensor cores (exercised by only 4 out of 26 kernels). As Volta does not have hardware performance counters for register file and L1i, the AccelWattch HW solver minimizes error by lumping their power to other commonly-accessed ones: instruction buffer, scheduler, and SM pipeline. Hence, the Others category grows proportionally to accommodate this reassignment. AccelWattch HYBRID shows a similar trend. Figure 8 also shows that replacing the hardware counters for L2 and NoC hardly changes the power breakdown compared to HW, suggesting that HYBRID is likely to work well when the software model of the targeted component closely approximates its behavior on real silicon.

Figure 9 shows the power breakdown for each kernel. Tensor cores consume a significant portion of total system power (geomean 28.7%) for the kernels that use them. Note that tensor cores are not part of the *Others* category in Figure 9. Also, several kernels (backprop_k1, hotspot_k1, sgemm_k1) consume over 90% of the peak power. We postulate this happens because these kernels keep resources busier than the rest due to high thread IPCs (5690, 7157, and 4668, respectively) and have a nearly even split of ALU and FPU (DPU) instructions, which can execute concurrently on Volta.

7 CASE STUDIES

7.1 Modeling Pascal and Turing Architectures

An important goal of an architecture power model is to enable accurate design space exploration. For the use-case scenarios, we envision an architect who starts with the Volta architecture and uses AccelWattch to estimate the power consumption for a new architecture with different parameters. While GPUWattch is configurable, its accuracy when varying parameters has been identified as a weakness [26] (see also Section 7.3). Thus, as our first case study, we apply AccelWattch to estimate the power consumption of two new architectures for which it has **not** been tuned. We emphasize that if we directly tuned models for these GPUs they would likely result in more accurate models.

Ideally, we would validate AccelWattch predictions for these new architectures against Volta chips that employ these different configurations. However, such chips do not exist. As a proxy, we select configuration parameters similar to the NVIDIA Pascal and Turing architectures, and compare against real Pascal and Turing chips. The Pascal and Turing architectures are the nearest to Volta. Hence, they are the most likely to have similar hardware implementations. Differences in the implementation of hardware units and the ISA between Volta and these architectures will manifest as modeling error. Table 3 lists the parameters of the target GPUs.

The evaluation closely follows the validation flow shown in Figure 1-⑨, with small modifications. First, the workloads are compiled with compute-capability support for Pascal and Turing (`code=sm_61` and `code=sm_75` compiler options, respectively). Second, Volta, Pascal and Turing implement different ISAs. Thus, comparing hardware runs on Pascal (or Turing) with Volta-derived traces in Accel-Sim would introduce spurious inaccuracies; we only attempt to model different architectural parameters with AccelWattch in these use cases after all, not different ISAs. To avoid such spurious inaccuracies, we re-extract traces for Pascal and Turing GPUs to use in the validation of the corresponding use cases. AccelWattch is still using the Volta-trained model—the traces are used only for validation purposes. Differences in hardware implementations still manifest as errors, as we do not attempt to model different functional unit implementations. We exclude all workloads that use tensor cores (CUTLASS and *cudaTensorCoreGemm*) from our validation suite for Pascal, since Pascal does not have tensor cores.

AccelWattch is based on the Volta architecture which is implemented at 12 nm, while Pascal is at 16 nm. Thus, following the flow in Figure 1-⑨, after we collect power estimates for Pascal from AccelWattch, we apply technology scaling based on published IRDS [17] parameters. Technology scaling reduces MAPE by 1.22%

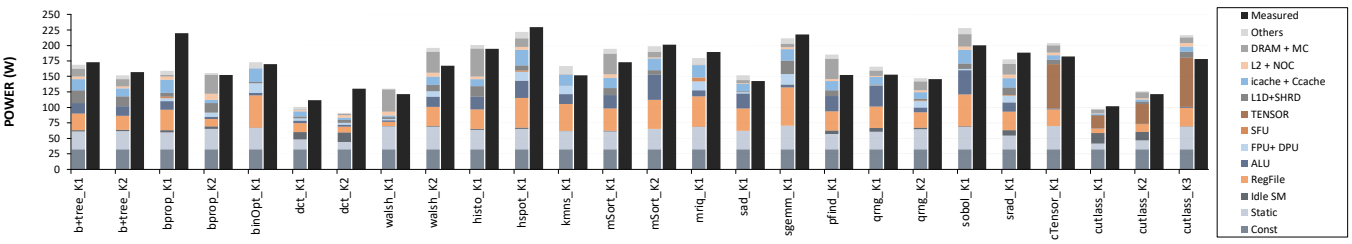


Figure 9: AccelWattch validation: AccelWattch SASS SIM modeling a Volta GV100.

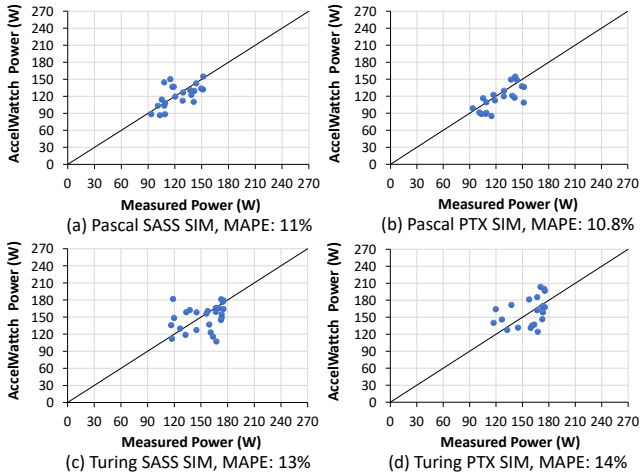


Figure 10: Correlation plots for case studies.

for PTX and 1.85% for SASS compared to the non-scaled models. Turing is also at 12 nm so it does not need technology scaling. We also set the constant power for Turing at 1.7× higher than Volta's to approximate the new board's fans and peripheral circuitry. We only make this change so we can compare against a real-world chip without perturbations from unrelated components; architecture research does not typically modify fans and peripherals.

Figure 10 shows the correlation plots of the estimated vs. measured power consumption for the two case studies, and Figure 11 shows the per-component breakdowns for all kernels. Overall, AccelWattch shows strong correlation to hardware power measurements for both Pascal and Turing. Figure 8 (right) shows the normalized average power breakdown. Similarly to Volta, the register file, static power and constant power are the three most significant components, together consuming 67.7% and 70.7% of the average total system power on Pascal and Turing, respectively.

Computer architects typically evaluate their designs by comparing a figure of merit (e.g., power) relative to a baseline design across workloads. As all simulators occasionally exhibit high errors, architects typically also consider averages across many workloads, rather than only a few worst-case applications. Following this common pattern, Figure 12a shows the estimated power of Pascal relative to the estimated power of Volta across workloads, and on average (red bar). The figure also shows the hardware-measured power of Pascal relative to Volta chips running the same workloads for comparison, along with an average of the relative hardware measurements. Similarly, Figures 12b and 12c show the estimated and hardware-measured power of Turing relative to Volta and Turing relative to Pascal. While AccelWattch's error varies by workload, the aggregate estimate closely tracks real hardware measurements for all architectures. Across all workloads (26 for Turing, 22 for Pascal), the average relative power as estimated by AccelWattch differs from the average relative power measured in hardware by 1% for Pascal vs. Volta, 3% for Turing vs. Volta, and 1% for Turing vs. Pascal. High errors are rare in AccelWattch: only 9 out of 26 workloads exhibit error over 10%.

The error is higher for Turing over Volta (Figure 12b) largely due to inaccuracies in Accel-Sim (Accel-Sim is not part of this work). For example, the L1d miss rate for *kmeans_K1* on a Turing RTX 2060S is 10× higher than the one estimated by Accel-Sim, leading to a 1.7× error in the run time estimate. As AccelWattch depends on the run time estimate to convert energy to power (Eq. (12)), inaccuracies in the performance model can adversely affect the power estimates. In addition, some errors are artifacts of ISA and hardware changes, which we did not intend to capture in these use cases, and are therefore spurious (Section 7.1). The Turing power relative to Volta is also concentrated in relatively small changes around zero (Figure 12b), which makes it easy even for small inaccuracies to result in estimates pointing in the opposite direction (*kmns_k1, sad_k1, pfind_k1*). Even with the adverse impact of performance model inaccuracies and treading around zero, only in 4 out of 26 workloads (i.e., 15% of the time) AccelWattch's prediction points in the opposite direction than the hardware measurement; in 85% of the time the predictions are tracking the measurements on real

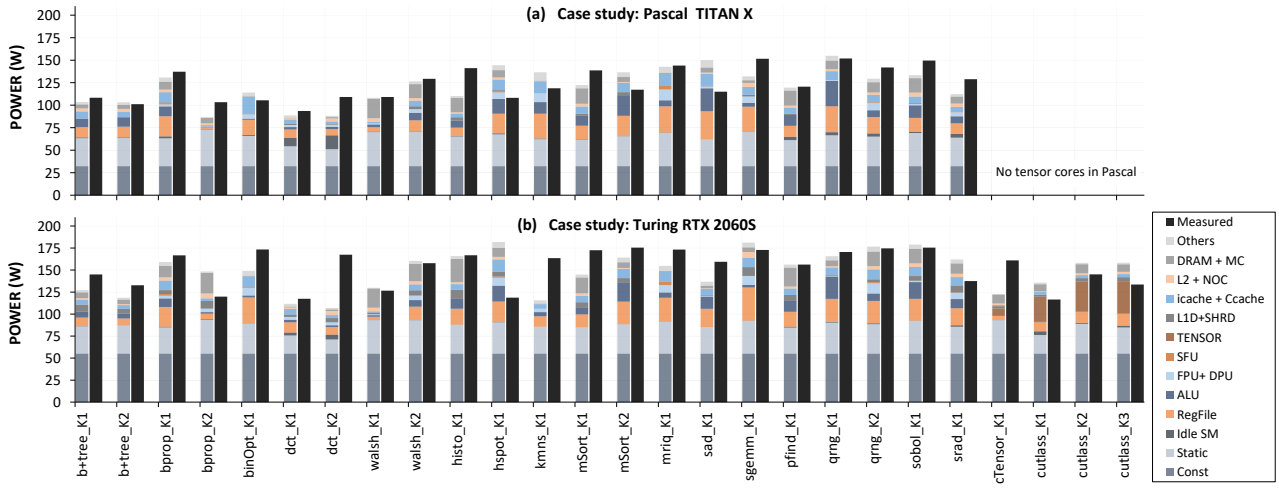
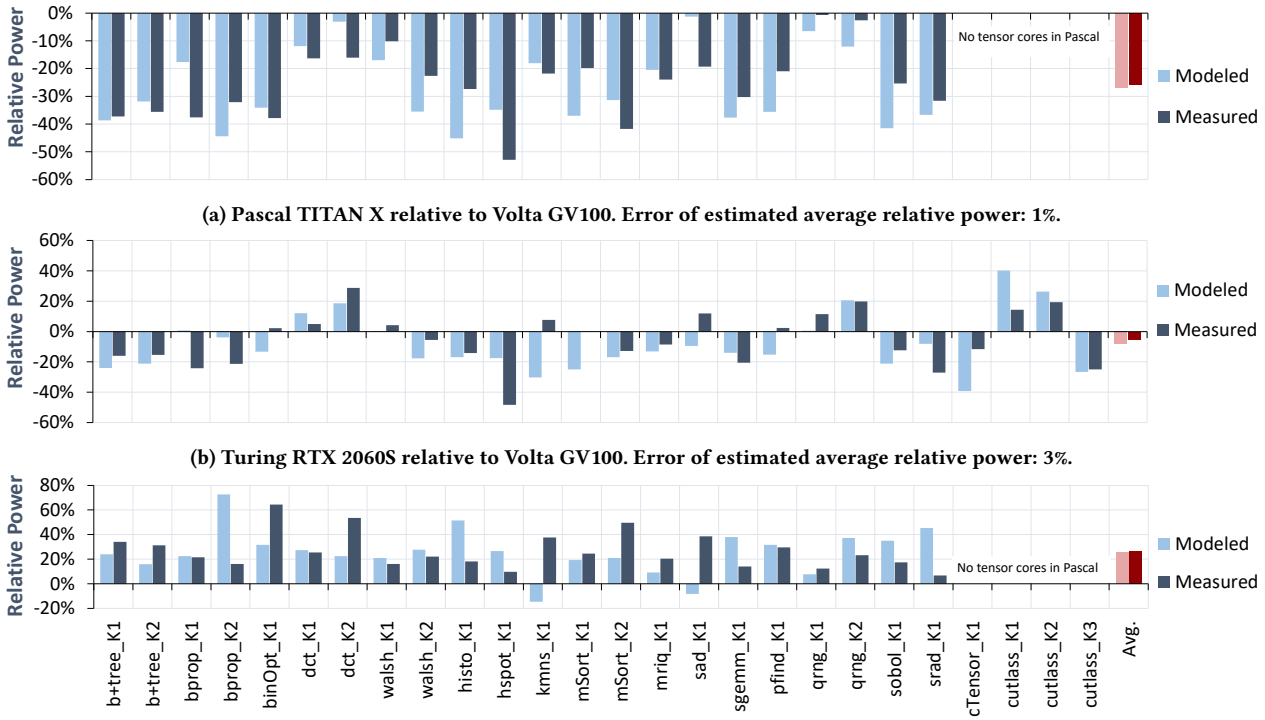


Figure 11: Case studies: AccelWattch SASS SIM (tuned for Volta), applied to model Pascal and Turing architectures.



(a) Pascal TITAN X relative to Volta GV100. Error of estimated average relative power: 1%.

(b) Turing RTX 2060S relative to Volta GV100. Error of estimated average relative power: 3%.

(c) Turing RTX 2060S relative to Pascal TITAN X. Error of estimated average relative power: 1%.

Figure 12: Relative Modeled and Measured Power across three architectures for AccelWattch SASS SIM.

silicon. This fraction grows for Turing relative to Pascal to 91%, and becomes 100% for Pascal relative to Volta.

7.2 AccelWattch for Deep Learning Workloads

GPUs are one of the dominant forces in Machine Learning (ML) acceleration [11]. To assess AccelWattch's accuracy in the ML space, our validation suite already includes workloads from CUTLASS, which primarily consists of general matrix multiply (GEMM) operations. Many operations in modern deep neural networks are either defined as GEMMs or can be cast as such, thus CUTLASS is representative of many ML workloads. In this section we delve deeper into the ML space and evaluate the accuracy of AccelWattch not only in GEMMs, but also in benchmarks implementing Convolutional (CNN) and Recurrent Long-Short Term Memory Neural Networks (RNN-LSTM). For this purpose we use DeepBench [25], a widely-used deep learning benchmark suite.

DeepBench utilizes closed-source, hand-tuned SASS kernels from the cuDNN [8] and cuBLAS [38] libraries, for which there are no PTX representations. While AccelWattch can execute closed-source kernels and estimate their power consumption, validating AccelWattch predictions on DeepBench is challenging and error-prone. Each DeepBench workload issues 10–130 kernels (geomean 33), and each kernel only uses about 12 SMs. The GPU hardware executes several kernels concurrently. However, Accel-Sim executes kernels only sequentially, leaving most of the simulated GPU idle and misleading AccelWattch to report significantly lower power. This is a limitation of Accel-Sim and not of AccelWattch. To mitigate this problem, we hand-construct a possible concurrent kernel

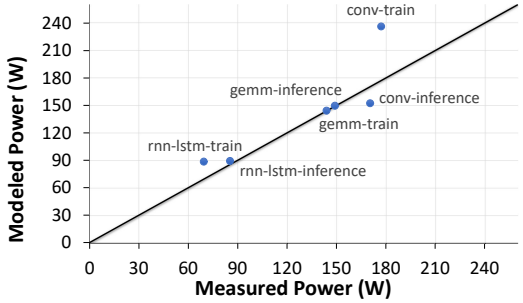


Figure 13: Correlation plot for DeepBench benchmarks.

execution schedule for each DeepBench benchmark, and then use AccelWattch to estimate its power consumption.

Even then, there is no guarantee the schedule is viable or that it matches the hardware-executed schedule. Kernel dependencies are unknown (cuDNN and cuBLAS are closed-source) and are thus not considered in the schedule. Also, cuDNN probes the hardware to decide which propagation algorithm to use, and may pick different algorithms for Accel-Sim and hardware execution, leading to hard-to-compare divergent behaviors. Similarly, AccelWattch HW reads hardware counters from Nsight, which is also constrained to serial kernel execution. Since we cannot design a validation process as precise as for the other benchmark suites, we exclude DeepBench from the validation suite and only present it as a case study.

We emphasize that these are not constraints imposed by AccelWattch, but by the existing performance simulators and profiling

tools, and only apply to validating AccelWattch's estimates. AccelWattch can predict the power of individual cuDNN and cuBLAS kernels just fine. With these restrictions in mind, we perform a best-effort experiment in which we evaluate the application of AccelWattch on 6 DeepBench benchmarks: train and inference for CONV, RNN-LSTM, and GEMM. Figure 13 presents the results. Overall, AccelWattch SASS SIM obtains 12.79% MAPE over Quadro GV100 hardware measurements for the DeepBench benchmarks.

7.3 Comparison to GPUWattch

To compare AccelWattch with GPUWattch, we apply GPUWattch's Fermi (NVIDIA GTX480) configuration to model Volta. As GPUWattch does not model tensor cores, we enhance it with AccelWattch's estimates for them. In fact, this is our Fermi starting point described in Section 5.4, with updates for the components that GPUWattch does not model (i.e., tensor cores). Running SASS and PTX simulations with this configuration for Volta architectures results in a MAPE of 219% and 225%, respectively, on the same validation suite of kernels. GPUWattch calculates unrealistically high power consumption for all kernels. The average power consumption it estimates is 530 W, with all but three of the kernels scoring above 300 W and a maximum of 926 W.

Moreover, in some cases GPUWattch reports unrealistic power consumption for particular components. For example, GPUWattch reports that constant and static power together account for 10.45 W on all validation kernels, which corresponds to 2.4% of the total system power on average. This contradicts our hardware power measurements on Volta, where even the lightest workload possible at the lowest frequency setting consumes >30 W. In addition, GPUWattch estimates that an average 14% of the total system power (including DRAM) is spent on INT_MUL units, compared to 1.4–1.8% in all AccelWattch variants. We believe the high power consumption that GPUWattch attributes to multipliers is unrealistic, as they would consume more power than GPUWattch's estimate for the register file (9.1%), pointing to a GPUWattch inaccuracy. Another notable difference includes GPUWattch's estimate of 27% of the system power spent on DRAM, compared to 8.4–9% in AccelWattch.

8 RELATED WORK

GPU architecture research has been largely enabled by event-driven cycle-accurate simulators such as GPGPU-Sim [3], Multi2Sim [44], and MGPUSim [42]. The lack of a fast, SASS-capable simulator was only recently resolved by Accel-Sim [20], into which AccelWattch integrates. Validating such tools against real hardware has always been a crucial component of performance modeling research. Similar to our work, prior studies [1, 6, 10, 12, 15, 46] present validation methodologies integrated into contemporary CPU simulators.

Wattch [5], McPAT [22], and SimplePower [49] are robust cycle-accurate CPU power modeling frameworks that have enabled a wide range of architecture-level research. Xi *et al.* [48] provided insightful guidelines for creating accurate power models with McPAT, including the use of analytical modeling for power gating and targeted microbenchmarking, which are followed by AccelWattch.

Cycle-accurate GPU power models based on McPAT (Lim *et al.* [23], GPUWattch [21], GPUSimPow [24]) model a decade-old Fermi architecture [27] at the PTX ISA level only. GPUWattch and GPUSimPow estimate constant power based on Eq. (2), a methodology no longer applicable to modern GPUs. AccelWattch rectifies this problem and is substantially more accurate than GPUWattch for modern architectures.

The IPP [16] analytic power model achieves high accuracy, but requires source-level PTX analysis that is infeasible for large or closed-source workloads, and PTX may not accurately correspond to real hardware activity [14]. Guerreiro *et al.* [13] present an analytical model that accurately predicts the power consumption of a GPU given a voltage-frequency setting. However, it provides a fixed power component encompassing static, constant and idle SM power, does not account for component power gating, and can only model architectures with a silicon implementation. AccelWattch is not similarly constrained, and also models 25 microarchitectural power components compared to 8 in Guerreiro *et al.*

In general, analytical models (GPUJoule [2], IPP [16], Guerreiro *et al.* [13]) can only capture program-level average power consumption, which hinders research that requires cycle-accurate simulation. Meanwhile, AccelWattch can provide a power trace at cycle-level granularity. AccelWattch is the first GPU power model, to the best of our knowledge, to have power components mapped to the SASS machine ISA instructions. Owing to this salient feature, AccelWattch can estimate the power consumption of closed-source GPU workloads that contain hand-tuned assembly.

9 CONCLUSIONS

There is a need for robust tools that will enable GPU architects to quickly model both the performance and the power consumption of modern GPUs. In this paper, we introduce AccelWattch, a configurable cycle-level power model for modern GPUs that can be directed by emulation and trace-driven simulation environments, hardware performance counters, or a combination of the two, striking a balance between model accuracy and performance modeling effort. AccelWattch is the only open-source tool capable of modeling closed-source workloads with hand-tuned SASS instructions.

We infer, for the first time to our knowledge, how modern GPUs power-gate chip-wide, SM-wide and lane-specific hardware components, and introduce an analytic power model that accurately captures the combined effects of power gating, thread divergence, intra-warp functional unit overlap, and variable SM occupancy. We also introduce a DVFS-aware methodology for modeling constant power. We integrate AccelWattch with Accel-Sim and GPGPU-Sim to facilitate its widespread use and release it in the public domain, along with its microbenchmarks and support infrastructure, as an open-source research tool for the computer architecture community. We extensively validate AccelWattch and find that it is within 7.5–9.2 ± 2.1–3.1% of hardware power measurements on a NVIDIA Volta Quandro GV100 GPU. Finally, we demonstrate that AccelWattch can enable reliable design space exploration.

ACKNOWLEDGMENTS

This work was partially funded by NSF awards CCF-1453853, CNS-1763743, CCF-1910924, and by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada. Tor M. Aamodt provides consulting services to Huawei Canada.

A ARTIFACT APPENDIX

A.1 Abstract

The artifact comprises the source code for the AccelWattch power modeling framework, a model tuned for an NVIDIA Quadro Volta GV100 GPU, the source code for the microbenchmarks used for tuning the model, the source code of support scripts and ancillary software, pre-compiled binaries of the validation suite, input datasets for the validation suite kernels, and Accel-Sim traces for the power microbenchmarks and validation suite benchmarks. The artifact is available publicly through an archived repository.

The artifact also describes the requirements and contains instructions for using the AccelWattch power modeling framework. These requirements and instructions are detailed in this appendix as well. Users of the artifact can reproduce the key AccelWattch and case studies results shown in Figures 7 through 12.

A.2 Artifact check-list

- **Algorithm:** A cycle-level constant, static and dynamic power model for the NVIDIA Volta Quadro GV100 GPU architecture tuned using quadratic programming optimization.
- **Program:** CUDA microbenchmarks for tuning AccelWattch and NVIDIA CUDA Samples, Rodinia 3.1, Parboil, and CUTLASS 1.3 benchmark suites. The artifact includes sources and pre-compiled binaries for all programs in the AccelWattch repository.
- **Compilation:** GCC 7.5.0 and NVCC V11.0.167.
- **Binary:** The artifact repository contains pre-compiled CUDA executables with dynamically linked libraries for all microbenchmarks and validation benchmarks.
- **Data set:** All datasets used in the experiments are publicly available and also included in the artifact repository.
- **Runtime environment:** Red Hat Enterprise Linux 8.4 with CUDA Toolkit 11.0; nvidia-smi from NVIDIA driver 465.19.01.
- **Hardware:** NVIDIA Volta Quadro GV100 or other similar V100 GPU to collect hardware power and performance measurements and Accel-Sim traces for the key results; NVIDIA Pascal TITAN X and NVIDIA Turing RTX2060S for the case studies; Intel Xeon CPU E5-2695 server or other similar system to run Accel-Sim performance and AccelWattch power simulations.
- **Execution:** There should be no other application running on the target GPU during the power and performance profiling stage. Power profiling for all validation benchmarks running serially on a single GPU takes 75 minutes to complete (average across 5 iterations; 15 minutes per iteration). Performance profiling for all validation benchmarks on a single GPU takes 10 minutes in total. AccelWattch runs take 16 hours to complete if all jobs are running concurrently. 8 AccelWattch configurations validated; 20 AccelWattch jobs per configuration.
- **Metrics:** The artifact reports a per-component power breakdown from AccelWattch runs for each validation kernel. The artifact also reports hardware power measurements for the same set of validation kernels collected from each target GPU.
- **Output:** Validation graphs, including case studies (Figures 7 – 12). The artifact also provides an excel sheet that contains the raw data used to generate these graphs. The expected results are pre-filled into this sheet.
- **Experiments:** The artifact includes a set of scripts and a guide to start from cloning the AccelWattch repository, following the experimental methodology used in this paper, and finally generating the Figures 7 through 12.
- **How much disk space required (approximately)?:** 600 GB
- **How much time is needed to prepare workflow (approximately)?:** Extracting traces and datasets, building the simulator, and compiling benchmarks can take 6 hours in total.
- **How much time is needed to complete experiments (approximately)?:** Performing hardware performance and power profiling for all 3 target GPUs can take 6 hours. AccelWattch power estimates for all 8 validated configurations presented in Figures 7 through 12 can take 2 days when running concurrently on a x86 system with 56 cores.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** BSD 2.0.
- **Workflow framework used?:** The Accel-Sim simulation framework is used extensively for managing simulator runs and collecting hardware performance counters. The build frameworks in individual benchmark suites are used for building the validation benchmarks.
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.5398781

A.3 Description

A.3.1 How to access. The artifact, as described in this paper, is archived on Zenodo at https://doi.org/10.5281/zenodo.5398781 under a permissive BSD 2.0 license. To ease the adoption of the tool, we are also in the process of integrating the AccelWattch sources with the latest version of Accel-Sim in the official Accel-Sim GitHub repository at https://github.com/accel-sim/accel-sim-framework, and the sources of the AccelWattch microbenchmarks and validation benchmarks at the official Accel-Sim GPU application collection repository at https://github.com/accel-sim/gpu-app-collection. Interested users should refer to these GitHub links for the latest version of AccelWattch and its support infrastructure.

A.3.2 Hardware dependencies. NVIDIA Volta Quadro GV100 or other similar V100 GPU to collect hardware power and performance measurements and Accel-Sim traces for the key results; NVIDIA Pascal TITAN X and NVIDIA Turing RTX2060S for the case studies; Intel Xeon CPU E5-2695 server or other similar system to run Accel-Sim performance and AccelWattch power simulations.

A.3.3 Software dependencies. AccelWattch uses GCC 7.5.0 and NVCC V11.0 (from Cuda Toolkit 11.0) with nvidia-smi from NVIDIA Driver version 465.19.01. AccelWattch is meant to be run on a modern linux distribution and relies on the same dependencies as the Accel-Sim framework. The list of Accel-Sim software dependencies is at https://github.com/accel-sim/accel-sim-framework#dependencies. Note that the Accel-Sim framework uses Python 2.x.x as the default version of Python for its scripts.

A.3.4 Data sets. All datasets for AccelWattch validation benchmarks are from publicly available benchmark suites, and are included in the archived artifact repository. The AccelWattch microbenchmarks do not require any input datasets.

A.4 Installation

AccelWattch is located in the `accel-sim-framework` folder. A thorough setup guide for AccelWattch is available at the `AccelWatch.md` file in the archived repository.

A.5 Experiment workflow

The primary experiments consist of running AccelWattch simulations of several CUDA kernels from popular benchmark suites and

correlating AccelWattch power estimates with hardware power measurements for the same set of kernels. A thorough experiment workflow for AccelWattch is available at the `AccelWatch.md` file in the repository.

A.6 Evaluation and expected results

The process to generate Figures 7 through 12 involves the following primary steps:

- Compiling validation set binaries.
- Collecting Accel-Sim SASS traces for validation kernels on the target GPU card.
- Collecting hardware power measurements for the validation kernels on the target GPU card.
- Collecting hardware performance counter information for the validation kernels on the target GPU card. This step is required for AccelWattch HW and AccelWattch HYBRID models.
- Building AccelWattch and running AccelWattch jobs for all validation kernels for all 8 validated configurations.
- Collecting AccelWattch power estimatess from completed AccelWattch jobs and placing them in the provided excel sheet to generate Figures 7 through 12.

These steps are explained in more detail at the `AccelWatch.md` file in the repository. The mean absolute percentage error (MAPE) reported for each AccelWattch configuration after reproducing results should not vary by more than ±2%. The provided excel sheet is pre-filled with the raw data used to generate the graphs.

A.7 Experiment customization

AccelWattch can estimate power consumption for any CUDA application that is compatible with Accel-Sim simulations. Follow the methodology of Accel-Sim detailed at https://github.com/accel-sim/gpu-app-collection/blob/release/README.md to add a new benchmark to the Accel-Sim framework and run Accel-Sim simulations. AccelWattch power estimatess are then enabled by using a configuration option in Accel-Sim's job launching script `run_simulations.py`.

REFERENCES

[1] Almutaz Adileh, Cecilia González-Álvarez, Juan Miguel De Haro Ruiz, and Lieven Eeckhout. 2019. Racing to Hardware-Validated Simulation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 58–67. https://doi.org/10.1109/ISPASS.2019.00014

[2] Akhil Arunkumar, Evgeny Bolotin, David Nellans, and Carole-Jean Wu. 2019. Understanding the Future of Energy Efficiency in Multi-Module GPUs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 519–532. https://doi.org/10.1109/HPCA.2019.00063

[3] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. 2009. Analyzing CUDA workloads using a detailed GPU simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. 163–174. https://doi.org/10.1109/ISPASS.2009.4919648

[4] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, Cambridge, UK.

[5] David Brooks, Vivek Tiwari, and Margaret 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)*. 83–94. https://doi.org/10.1145/339647.339657

[6] Trevor E. Carlson, Wim Heirman, Stijn Eyerman, Ibrahim Hur, and Lieven Eeckhout. 2014. An Evaluation of High-Level Mechanistic Core Models. *ACM Transactions on Architecture and Code Optimization* 11, 3, Article 28 (Aug. 2014), 25 pages. https://doi.org/10.1145/2629677

[7] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 44–54. https://doi.org/10.1109/IISWC.2009.5306797

[8] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. *CoRR* abs/1410.0759 (2014). arXiv:1410.0759 http://arxiv.org/abs/1410.0759

[9] Arnaud de Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. 2016. Mean Absolute Percentage Error for regression models. *Neurocomputing* 192 (2016), 38–48. https://doi.org/10.1016/j.neucom.2015.12.114

[10] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. 2001. Measuring Experimental Error in Microprocessor Simulation. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA '01)*. 266–277. https://doi.org/10.1145/379240.565338

[11] Forbes. 2019. NVIDIA Dominates The Market For Cloud AI Accelerators More Than You Think. https://www.forbes.com/sites/paulteich/2019/06/17/nvidia-dominates-the-market-for-cloud-ai-accelerators-more-than-you-think/#676dea375edb. Accessed: 2020-11-24.

[12] Jeff Gibson, Robert Kunz, David Ofelt, Mark Horowitz, John Hennessy, and Mark Heinrich. 2000. FLASH vs. (Simulated) FLASH: Closing the Simulation Loop. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*. 49–58. https://doi.org/10.1145/378993.379000

[13] Joao Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomas. 2018. GPGPU Power Modeling for Multi-domain Voltage-Frequency Scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 789–800. https://doi.org/10.1109/HPCA.2018.00072

[14] Anthony Gutierrez, Bradford M. Beckmann, Alexandru Dutu, Joseph Gross, Michael LeBeane, John Kalamatianos, Onur Kayiran, Matthew Poremba, Brandon Potter, Sooraj Puthoor, Matthew D. Sinclair, Mark Wyse, Jieming Yin, Xianwei Zhang, Akshay Jain, and Timothy Rogers. 2018. Lost in Abstraction: Pitfalls of Analyzing GPUs at the Intermediate Language Level. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 608–619. https://doi.org/10.1109/HPCA.2018.00058

[15] Anthony Gutierrez, Joseph Pusdesris, Ronald G. Dreslinski, Trevor Mudge, Chander Sudanthi, Christopher D. Emmons, Mitchell Hayenga, and Nigel Paver. 2014. Sources of error in full-system simulation. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 13–22. https://doi.org/10.1109/ISPASS.2014.6844457

[16] Sunpyo Hong and Hyesoon Kim. 2010. An Integrated GPU Power and Performance Model. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. 280–289. https://doi.org/10.1145/1815961.1815998

[17] IEEE. 2016. International Roadmap for Devices and Systems. https://irds.ieee.org/editions/2016/. Accessed: 2020-11-24.

[18] Shailendra Jain, Surhud Khare, Satish Yada, V Ambili, Praveen Salihundam, Shiva Ramani, Sriram Muthukumar, M Srinivasan, Arun Kumar, Shasi Kumar Gb, Rajaraman Ramanarayanan, Vasantha Erraguntla, Jason Howard, Sriram Vangal, Saurabh Dighe, Greg Ruhl, Paolo Aseron, Howard Wilson, Nitin Borkar, Vivek De, and Shekhar Borkar. 2012. A 280mV-to-1.2V wide-operating-range IA-32 processor in 32nm CMOS. In *2012 IEEE International Solid-State Circuits Conference*. 66–68. https://doi.org/10.1109/ISSCC.2012.6176932

[19] Zhe Jia, Marco Maggioni, Benjamin Staiger, and Daniele Paolo Scarpazza. 2018. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. *CoRR* abs/1804.06826 (April 2018). arXiv:1804.06826 http://arxiv.org/abs/1804.06826

[20] Mahmoud Khairy, Zhesheng Shen, Tor M. Aamodt, and Timothy G. Rogers. 2020. Accel-Sim: An Extensible Simulation Framework for Validated GPU Modeling. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 473–486. https://doi.org/10.1109/ISCA45697.2020.00047

[21] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling Energy Optimizations in GPGPUs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13)*. 487–498. https://doi.org/10.1145/2485922.2485964

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

[23] 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. https://doi.org/10.1145/2611758

[24] Jan Lucas, Sohan Lal, Michael Andersch, Mauricio Alvarez-Mesa, and Ben Juurlink. 2013. How a single chip causes massive power bills GPUSimPow: A GPGPU power simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 97–106. https://doi.org/10.1109/ISPASS.2013.6557150

[25] Sharan Narang. 2016. DeepBench. https://svail.github.io/DeepBench/. Accessed: 2021-09-08.

[26] Tony Nowatzki, Jaikrishnan Menon, Chen-Han Ho, and Karthikeyan Sankaralingam. 2015. Architectural Simulators Considered Harmful. *IEEE Micro* 35, 6 (2015), 4–12. https://doi.org/10.1109/MM.2015.74

[27] NVIDIA. 2009. Whitepaper: NVIDIA's Next Generation CUDA Compute Architecture: Fermi. https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf. Accessed: 2020-11-24.

[28] NVIDIA. 2016. nvidia-smi - NVIDIA System Management Interface program. http://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf. Accessed: 2020-11-24.

[29] NVIDIA. 2016. Whitepaper: NVIDIA Tesla P100. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf. Accessed: 2020-11-24.

[30] NVIDIA. 2017. Whitepaper: NVIDIA Telsa V100 GPU Architecture. http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf. Accessed: 2020-11-24.

[31] NVIDIA. 2018. Whitepaper: NVIDIA Turing GPU Architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf. Accessed: 2020-11-21.

[32] NVIDIA. 2019. CUTLASS: CUDA template library for dense linear algebra at all levels and scales. https://github.com/NVIDIA/cutlass. Accessed: 2020-11-24.

[33] NVIDIA. 2019. NVML API Reference. https://docs.nvidia.com/deploy/nvml-api/nvml-api-reference.html. Accessed: 2020-11-24.

[34] NVIDIA. 2020. CUDA Compiler Driver NVCC, v11.0. https://docs.nvidia.com/cuda/archive/11.0/cuda-compiler-driver-nvcc/index.html. Accessed: 2021-4-15.

[35] NVIDIA. 2020. CUDA Samples. https://docs.nvidia.com/cuda/archive/11.0/cuda-samples/index.html. Accessed: 2021-4-16.

[36] NVIDIA. 2020. Instruction Set Reference. https://docs.nvidia.com/cuda/cuda-binary-utilities/index.html#instruction-set-ref. Accessed: 2020-11-24.

[37] NVIDIA. 2020. Parallel Thread Execution ISA Version 7.0. https://docs.nvidia.com/cuda/parallel-thread-execution/index.html. Accessed: 2020-11-24.

[38] NVIDIA. 2021. cuBLAS. https://developer.nvidia.com/cublas/. Accessed: 2021-09-08.

[39] NVIDIA. 2021. Nsight Compute. https://docs.nvidia.com/nsight-compute/NsightCompute/index.html. Accessed: 2021-9-5.

[40] Addison Snell and Laura Segervall. 2017. HPC application support for GPU computing. https://www.nvidia.com/content/intersect-360-HPC-application-support.pdf. Accessed: 2020-11-24.

[41] John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, and Wen mei W. Hwu. 2012. Parboil:A revised benchmark suite for scientific and commercial throughput computing. In *IMPACT Technical Report, IMPACT-12-01, University of Illinois, at Urbana-Champaign.*

[42] Yifan Sun, Trinayan Baruah, Saiful A. Mojumder, Shi Dong, Xiang Gong, Shane Treadway, Yuhui Bao, Spencer Hance, Carter McCardwell, Vincent Zhao, Harrison Barclay, Amir Kavyan Ziabari, Zhongliang Chen, Rafael Ubal, José L. Abellán, John Kim, Ajay Joshi, and David Kaeli. 2019. MGPUSim: Enabling Multi-GPU Performance Modeling and Optimization. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. 197–209.

[43] TOP500.org. 2021. TOP500 List. https://www.top500.org/lists/top500/2021/06/. Accessed: 2021-9-5.

[44] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. 2012. Multi2Sim: A simulation framework for CPU-GPU computing. In *2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 335–344.

[45] Oreste Villa, Mark Stephenson, David W. Nellans, and Stephen W. Keckler. 2019. NVBit: A Dynamic Binary Instrumentation Framework for NVIDIA GPUs. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 372–383. https://doi.org/10.1145/3352460.3358307

[46] Matthew Walker, Sascha Bischoff, Stephan Diestelhorst, Geoff Merrett, and Bashir Al-Hashimi. 2018. Hardware-Validated CPU Performance and Energy Modelling. In *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 44–53. https://doi.org/10.1109/ISPASS.2018.00013

[47] Gene Wu, Joseph L. Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU performance and power estimation using machine learning. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 564–576. https://doi.org/10.1109/HPCA.2015.7056063

[48] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 577–589. https://doi.org/10.1109/HPCA.2015.7056064

[49] Wu Ye, Narayanan Vijaykrishnan, Mahmut T. Kandemir, and Mary Jane Irwin. 2000. The design and use of simplePower: a cycle-accurate energy estimation tool. In *Proceedings 37th Design Automation Conference*. 340–345. https://doi.org/10.1145/337292.337436

[50] Ying Zhang, Yue Hu, Bin Li, and Lu Peng. 2011. Performance and Power Analysis of ATI GPU: A Statistical Approach. In *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*. 149–158. https://doi.org/10.1109/NAS.2011.51

[51] Brian Zimmer, Yunsup Lee, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtic, Ben Keller, Stevo Bailey, Milovan Blagojevic, Pi-Feng Chiu, Hanh-Phuc Le, Po-Hung Chen, Nicholas Sutardja, Rimas Avizienis, Andrew Waterman, Brian Richards, Philippe Flatresse, Elad Alon, Krste Asanović, and Borivoje Nikolić. 2015. A RISC-V vector processor with tightly-integrated switched-capacitor DC-DC converters in 28nm FDSOI. In *2015 Symposium on VLSI Circuits (VLSI Circuits)*. C316–C317. https://doi.org/10.1109/VLSIC.2015.7231305