How a Single Chip Causes Massive Power Bills
GPUSimPow: A GPGPU Power Simulator

Jan Lucas, Sohan Lal, Michael Andersch
Mauricio Alvarez-Mesa, Ben Juurlink
Embedded Systems Architecture Department
TU Berlin, Einsteinufer 17, D-10587 Berlin, Germany
http://www.aes.tu-berlin.de
{j.lucas,sohan.lal,michael.andersch,alvarez,juurlink}@aes.tu-berlin.de

***Abstract*—Modern GPUs are true power houses in every meaning of the word: While they offer general-purpose (GPGPU) compute performance an order of magnitude higher than that of conventional CPUs, they have also been rapidly approaching the infamous "power wall", as a single chip sometimes consumes more than 300W. Thus, the design space of GPGPU microarchitecture has been extended by another dimension: power. While GPU researchers have previously relied on cycle-accurate simulators for estimating performance during design cycles, there are no simulation tools that include power as well. To mitigate this issue, we introduce the GPUSimPow power estimation framework for GPGPUs consisting of both analytical and empirical models for regular and irregular hardware components. To validate this framework, we build a custom measurement setup to obtain power numbers from real graphics cards. An evaluation on a set of well-known benchmarks reveals an average relative error of 11.7% between simulated and hardware power for GT240 and an average relative error of 10.8% for GTX580. The simulator has been made available to the public [1].**

I. Introduction

With processor designs becoming ever more complex and chip manufacturing processes becoming harder to control, the inability of computer architects to produce working prototypes of their designs for testing is a more pressing problem than ever before. With chips rapidly approaching (and, nowadays, touching) the power wall, the conventional design space of processor architecture has been extended by another dimension: Energy consumption.

Over the past years, it has become apparent that the chips consuming the most energy by far are modern Graphics Processing Units (GPUs). With GPUs turning into major devices for general purpose computations, so-called general-purpose computation on GPUs (GPGPU), this trend has only accelerated as more and more parties are striving to drive GPU performance up. The inability to manufacture chips to evaluate architectural design choices, however, remains, as does the looming power wall.

So how do the design of new GPU architectures, the inability to manufacture chips just for testing, and the requirement to not only estimate a chip's performance, but also its power *during development* come together? On the one hand, if we disregard power and only consider performance, this question has been answered by several researchers over the past years: GPU architects now must rely on building cycle-accurate architectural simulators in high-level languages and evaluate novel designs using these simulators. On the other hand, there are several accepted tools and frameworks to model and estimate power consumption for *CPUs* such as Wattch [2]. To the best of our knowledge, however, no one ever combined architectural GPU simulators with power models to create *GPU power simulators* before.

In this work, we seek to mitigate this issue by introducing *GPUSimPow*, a power simulation framework for contemporary GPGPU architectures. GPUSimPow is able to estimate multiple characteristics of a hypothetical GPU architecture such as chip area, gate leakage, and peak dynamic power, as well as precisely simulate the power consumed during execution of GPGPU workloads. With this power simulator, computer architects can evaluate design choices early from a power perspective, and GPGPU programmers gain an effective way to investigate their GPGPU codes, so-called *kernels*, to optimize power consumption from a software perspective. To make GPUSimPow flexible enough that architectural design choices can easily be carried out while still maintaining reasonably high accuracy of the power predictions compared to real hardware, we model the components of a GPU architecture in two ways: Regular components such as memories are modeled *analytically* using the well-known McPAT [3] tool (which, by itself, integrates CACTI6.5 [4]). Irregular components such as address generation units (AGUs) or special-function units (SFUs), on the other hand, are modeled *empirically* by acquiring measurement data on real hardware. To obtain accurate measurements, we propose a specialized *measurement testbed*.

In summary, we make the following contributions:

- We develop the GPUSimPow power simulator that is able to generate area, power and runtime power estimations for contemporary GPGPU micro-architectures and GPGPU kernels.
- We propose a novel measurement methodology to be able to accurately measure GPU power consumption on real hardware down to the individual kernel.

This paper is organized as follows: First and foremost, Section II introduces related work. Then, in Section III, we introduce our power simulator, describe its structure and the simulated architecture. Then, Section IV presents the measure-

ment setup we used to estimate component power and validate the simulator results, as well as the rest of our experimental setup for simulator runs. Section V presents the simulator results and compares them with power measurements on real hardware. Finally, we draw conclusions in Section VI.

II. Related Work

Relevant related work for this paper can be classified into two categories: First, work on GPU power modeling in general, and second, previous attempts at reliable measurements of GPU power consumption.

For general GPU power modeling, the available body of previous work is rather small. On the one hand, there have been approaches such as the ones from Hong and Kim [5] or Ma et al. [6] which are based entirely on measured data. While this type of power model is able to deliver superior accuracy for the architecture it was built from, it lacks the capability to make accurate predictions about GPUs with other architectural parameters and designs. On the other hand, several researchers have built purely analytic power models, such as Ramani et al. [7] and Wang [8]. While such approaches generally show strong correlation between different simulated and hardware GPU architecture configurations, they typically cannot provide reasonable absolute accuracy due to the lack of either industrial or measured anchor data. Our power simulator improves upon all these prior approaches by *combining* both empirical and analytical component models to create a system that is both architecturally flexible and shows reasonable absolute accuracy. A similar approach to ours has previously been used to estimate CPU power consumption by the well-known McPAT tool [3].

In terms of measuring GPU power consumption, many previous approaches have made strong assumptions about the hardware they measure, leading to inaccurate measurement methodologies. For example, Hong and Kim [5] assume the GPU power can be calculated from measuring the power of the entire PC under load and subtracting the power of the PC in idle state. This assumption is highly naive since the power used by the remaining PC components is usually not constant and the measurement results will include ATX power supply losses. Large bypass capacitors inside the ATX power supply prevent the accurate measurement of power for kernels which run fewer than 50 ms. Other papers [9], [10], [6] use improved measurement methodologies, but still exhibit multiple weaknesses. To the best of our knowledge, all these published methodologies either fail to measure all power sources, e.g. do not measure the power provided via the graphics card slot [9], measure only current and assume constant voltages [10], or use low sampling frequencies that prevent them from measuring short-term power variations [6], [9]. Our methodology improves upon all of these aspects (see Section IV-A).

III. The GPUSimPow Tool

In this section, we present our power simulation framework called *GPUSimPow*. An overview over GPUSimPow's structure is given in Section III-A. In Section III-B we are providing information about the power model and its link to the performance simulator. The baseline architecture details are explained in Section III-C. Finally, Section III-D gives details on how the empirical parts of the model have been derived.

A. Overview

GPUSimPow is a power simulator for GPGPU workloads, i.e. given a configuration of a particular GPU architecture and a GPGPU kernel written in CUDA [11] or OpenCL [12], GPUSimPow is capable of producing both architectural information such as static power, peak dynamic power, and area, as well as runtime dynamic power for the kernel at hand. The simulator is designed to be flexible regarding the architecture that is simulated to allow architects to utilize the simulator as a high-level tool to explore the GPU architecture design space. Therefore, the key parameters of the simulated architecture are supplied using a simple XML-based interface. For example, GPUSimPow is able to coherently simulate an architecture with a varied number of cores.

B. Power Model

In general, power for switching circuits is described by the well-known Eq. 1 [3].

$$P_{\text{total}} = \alpha C V_{dd} \Delta V f_{\text{clk}} + V_{dd} I_{\text{Short-circuit}} + V_{dd} I_{\text{leakage}} \quad (1)$$

The first term is the dynamic power that is spent charging and discharging capacitive loads when the circuit switches state. An important factor for this paper is the *activity factor* α that describes the percentage of the circuit's capacitance being charged during switching. The second term of Eq. 1 is the short-circuit power being consumed when *both* pull-up and pull-down networks in a CMOS circuit are on for a short amount of time. Thus, the total power consumed by a circuit during switching is the sum of the dynamic and short-circuit power terms. Finally, the third term of Eq. 1 is the *static* power consumed due to leakage current through the transistors, where leakage consists of two distinct types: Subthreshold leakage, where a transistor that is switched off leaks current between its source and drain, and gate leakage, where current leaks through the transistor's gate terminal.

Structurally, GPUSimPow consists of two main parts, visualized in Figure 1: First, a cycle-accurate GPGPU simulator that simulates the given kernel and thereby generates utilization information and activity factors α for all components of the GPU architecture, and second, a chip representation with a power model for each component that uses the activity information from the simulator to produce power numbers for a particular kernel. From the chip representation, statistics about area and peak, leakage, and short-circuit power are inferred as well.

For the cycle-accurate GPGPU simulator, we employ a modified version of the most recent GPGPU-Sim [13] that has been altered to produce access counts and other activity

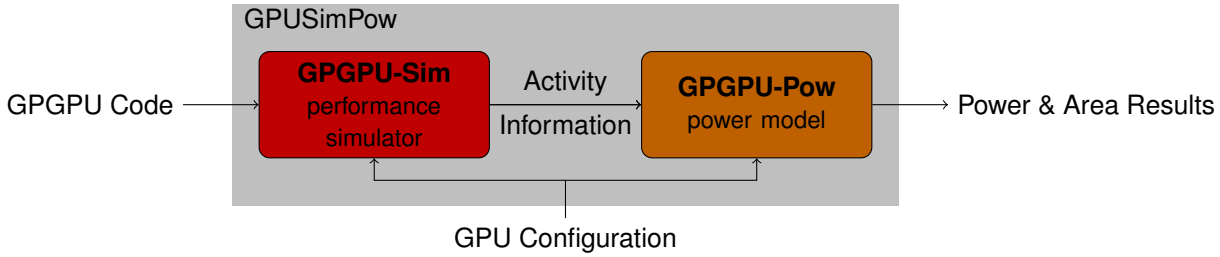


Fig. 1: Overview over the GPUSimPow simulation framework.

information for all parts of the simulated architecture. GPGPU-Sim has been developed for an architecture that is not equal but comparable to many current off-the-shelve GPUs such as NVIDIA's Fermi [14] or AMD's GCN [15]. Further details about the architecture are given in the next section.

The chip representation and power model are provided by a heavily modified variant of McPAT [3] we name *GPGPU-Pow*. McPAT integrates three different modeling tiers *hierarchically* to provide a flexible and highly accurate power model for *CPUs*: The architectural tier, where a processor is broken down into major components such as cores, caches, and memory controllers, the circuit tier, where the architectural components are mapped to basic circuit structures such as arrays or clocking networks, and the technology tier, which provides the physical parameters, such as current densities and capacitances, of the circuits. Besides this hierarchy, a unique advantage of McPAT is its *combination* of analytical and empirical models for the individual components. We embrace both the hierarchical as well as the combined nature of McPAT and develop a McPAT-based model for GPUs. On the one hand, this requires many modifications to McPAT, as multiple components that are present in the CPU architecture model such as register alias tables cannot be reused for GPUs, and various core components of GPU architectures, such as stacks to handle thread divergence, are not present in CPUs. On the other hand, using McPAT enables us to utilize all the integrated low-level technological information, e.g. to scale the GPU power model for a specific manufacturing process node, we can use the ITRS roadmap scaling techniques within McPAT.

C. Modeled Architecture

The GPU micro-architecture we designed our power model for is comparable to the one given in GPGPU-Sim to ensure a good "fit" between GPGPU simulator and power model. Generally, it is a single-instruction multiple-thread (SIMT) architecture that uses a stack-based divergence handling mechanism, well representative of many current GPUs.

On the highest level, a GPU chip in our model consists of a memory controller (MC), a Network-on-Chip (NoC), a PCIe controller (PCIeC), and a collection of cores. Besides the actual chip, we model the GDDR5 graphics DRAM as well. For NoC, MC, and PCIeC, we re-used the highly configurable models already present in McPAT and adjusted their parameters to fit the different requirements of a GPU. The internal structure of a core consists of a *Warp Control Unit* (WCU), a highly banked register file, a set of SIMD execution units (INT, FP, SFU), and a load/store unit (LDSTU). In the following, each of these components as well as our GDDR model are briefly introduced.

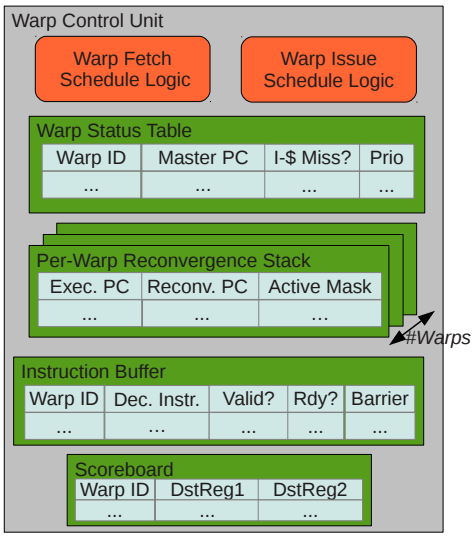


Fig. 2: High-level overview over the internals of the GPU's frontend, in our model called *warp control unit*.

1) Warp Control Unit: The WCU represents the front-end of a single core. As such, the WCU is responsible for keeping the execution back-end, i.e. the functional units and the load/store hardware, supplied with instructions at all times. Thus, the WCU handles thread management (i.e. formation of *warps* from threads and the relation of per-thread control flow under warp constraints), warp scheduling, warp instruction fetching, decoding, dependency resolution, and renaming. An overview over our WCU model is depicted in Figure 2.

The information needed for each warp to fetch instructions and manage the warp threads is contained in a single multi-ported RAM table, the Warp Status Table (WST). The WST contains one entry for each in-flight warp the core can handle. To select a warp to fetch an instruction for, a rotating-priority (round-robin) warp scheduler is modeled. Such schedulers consist of a set of inverters, a wide priority encoder, and a phase counter. These components have been modeled from

appropriate circuit plans [16] using McPAT's circuit and technology layers. After instructions have been fetched from the I-Cache, they are decoded. For this, we re-use the instruction decoder hardware models already present in McPAT.

As is common for most GPGPU applications on modern GPU architectures, the individual in-flight threads often execute different dynamic instruction paths. The grouping of threads into SIMD bundles (warps) implicitly forces the thread PCs to have the same value at all times, however. If this is not the case due to the threads branching into different dynamic execution paths, the execution of threads in a single warp but with different PCs is serialized. To achieve this serialization and keep track of the thread IDs that have to execute certain branch outcomes, the hardware uses a stack memory called the reconvergence stack [17]. For each individual in-flight warp, the hardware maintains a separate stack. In our model, a stack consists of tokens, each of which contains an execution PC, a reconvergence PC, and an active mask for that warp and code block.

Once an instruction has been decoded, the WCU places the instruction into an instruction buffer (IB) slot. The instruction resides in its buffer slot until it is ready to execute, that is, if its register dependencies have been resolved (in scoreboarded architectures) or the previous instruction from the same warp has been committed (in blocking barrel-processing architectures). The instruction buffer is a cache-like structure that is tagged by the warp ID and has an associativity greater than one, i.e. each instruction can be buffered in one of several slots tagged by its parent warp ID.

For resolving register dependencies, GPUs (e.g. NVIDIA Fermi) use simple approaches based on scoreboarding [18]. In our models, a scoreboard is a cache-like table tagged by the warp ID.

2) Register File: The GPU register file model is based on an NVIDIA patent [19] and built from multiple single ported RAM banks. Operands are collected over multiple cycles to simulate a multi-ported register file. Different threads will have their registers stored in different banks. This scheme increases the area density of the register file. A crossbar is used to connect the different register banks to a set of operand collector units which are two-ported four-entry register files.

3) Execution Units: The basic unit of execution flow in the SIMT core is the warp. The GPU has a set of SIMD execution units which execute the warp threads in lock step. For example, the SIMT core in the NVIDIA GT240 has eight fully pipelined floating point units (FPUs), eight pipelined integer units (IUs) and two special function units (SFUs) to execute transcendental instructions such as sine, cosine, reciprocal, and square root. In our power model, we used the area numbers published by Sameh et al. [20] for FPUs, the results of Caro et al. [21] for power and area of the SFUs with scaling for the desired process technology, and our own measurements for power of integer units and floating point units (see Section III-D).

4) Load/Store Unit: The load-store unit (LDSTU) is functionally responsible for handling instructions that read or write any kind of memory. In the power model, the LDSTU encapsulates the top-tier memory structures of the core, i.e. L1/SMEM, the constant caches and the L2 caches. In a future variant of the model, the LDSTU will contain the texture caching subsystem, i.e. texture caches and texture mapping units, as well. An overall overview of the LDSTU is depicted in Figure 3.

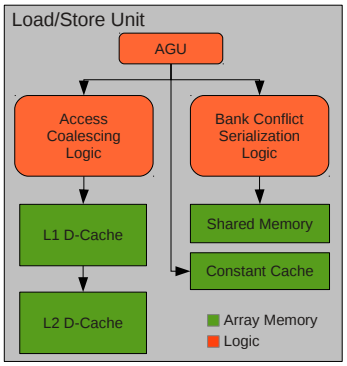


Fig. 3: High-level overview over the internals of the GPU's load/store unit.

As the figure shows, a memory access instruction for an entire warp is first passed to the address generators. Given base addresses as well as strides and offsets, the address generation unit (AGU) generates one memory address per thread in the warp. Given reasonable warp sizes of 32/64 threads in modern architectures, this requires very high bandwidth address generation units that supply the later stages of the memory subsystem with 32/64 memory addresses each cycle. We model the complete AGU as an array of parallel high-bandwidth sub-AGUs (SAGU), each of which is able to generate 8 memory addresses per cycle [22]. Given the memory address bundle for all threads in the warp, the address bundle is further analyzed depending on the type of memory the instruction accesses.

If the instruction accesses constant memory, the addresses are checked for equality. The number of generated constant cache / constant memory accesses is equal to the number of different addresses in the address bundle, e.g. if all addresses are equal, the memory access can be serviced with a single constant memory request, allowing for high-bandwidth operation. The constant memory segment is cached in an entire hierarchy of constant caches [23].

If the instruction accesses global memory, it is first coalesced before being passed to the L1/L2 caches and/or DRAM. The coalescing system is modeled after a corresponding NVIDIA patent [24] and consists of an input queue, output queue, pending request table, and a finite state machine. The goal of coalescing is to service the addresses requested by the memory access in as few memory requests as possible. As our research revealed, CACTI cannot be used to model buffers with few but very large entries such as the pending

request table and input queue of the coalescer. Instead, we compute the total amount of bits which must be held in the coalescing system at any time and model the required storage using D-FlipFlops.

In several modern GPUs, shared memory (SMEM) and the L1 data cache are portions of the same physical memory structure. The distribution of physical memory to L1 and SMEM is configurable. Therefore, we model L1 and SMEM as an integrated physical memory structure and convert accesses to SMEM and L1 hits to accesses to that memory structure. The physical memory consists of multiple banks to be able to supply multiple accessing threads with data at a high rate. Besides the physical memory banks, the SMEM/L1 consists of interconnects for addresses and data, both modeled as crossbars, and a bank conflict checking unit [25]. The L2 cache is shared over the entire GPU and connected to the cores through the NoC.

5) Global Memory: The global memory in GPUs has high bandwidth but long latency. The current generation of GPUs such as Fermi use either DDR3 SDRAM or GDDR5 SGRAM chips to implement the global memory. The power consumed by typical DDR or GDDR chips can be divided into background, activate, read/write, termination, and refresh power [26]. We extract numbers for each of these components from industry data sheets [27].

D. Deriving Power Empirically

Using our custom power measurement setup described in Section IV-A, we were able to build an empirical, measurement-based model for the INT and FP execution units by running custom microbenchmarks to estimate the energy per INT/FP operation. As described earlier in Section III-C, todays GPUs use an SIMT approach to execute multiple threads at the same time on SIMD hardware.

We can use this SIMT-style of execution to our advantage by enabling different numbers of execution units while keeping the activity of all other units, except for the register files, constant. This way, we can estimate power for the execution units with reasonably high accuracy. For both integer and floating point operations, we launch one thread block for each core and use 512 threads per block to ensure all cores and targeted execution units are busy. We used unrolling to make the loop overhead of our testing loops negligible. In the loop nest of our integer test code, we are simulating Linear Shift Feedback Registers while for the floating point case we are using Mandelbrot set iterations. In both cases, we are alternately configuring the test kernels to use 31 enabled threads per warp and 1 enabled thread per warp. Both configurations have the same execution time. We then calculate the energy difference between these two kernel launches and divide the result by the number of executed instructions, number of cores and difference in execution units enabled to arrive at an estimate for the energy used by a single execution unit executing a single instruction.

Our measurements show that integer instructions are using approximately 40 pJ while floating point instructions are using about 75 pJ per instruction. NVIDIA reports 50 pJ per floating point instruction [28]. The power model of the execution units is based on our measurements.

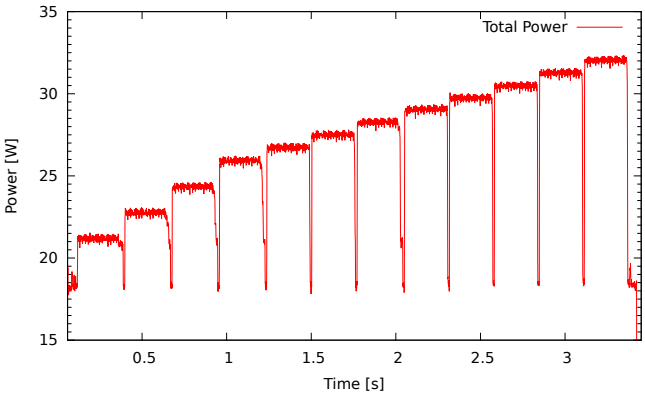


Fig. 4: Power measurement results of a GT240 card running the same kernel 12 times with increasing number of thread blocks. The GT240 features 12 cores distributed evenly over 4 core clusters.

While we developed all component models to the best of our knowledge, there are areas of GPU architecture where publicly available information is especially scarce, such as the raster operations pipelines (ROPs) or fixed-function video decode hardware, which consume power due to leakage even if they are not used in GPGPU applications. While we cannot model such components accurately because of the lack of information, we know nonetheless that these components are part of the chip. To be able to account for the amount of power they contribute, we used our measurement equipment to build empirical models of "base power" for cores and core clusters (called TPCs[1] or, more recently, GPCs[2] in NVIDIA terminology). These base power numbers are derived by measuring core/cluster power and subtracting the power of all components we know about. An example of a measurement used to estimate cluster power is shown in Figure 4. The figure shows that increasing the number of thread blocks used for computation gradually increases the power, up to a certain limit when the entire chip is occupied (not shown). More interestingly, the figure shows that up to 4 blocks, adding another thread block to the computation increases power by a larger margin than beyond 4 blocks. The reason for this is the way the hardware scheduler distributes thread blocks: Until the entire chip is occupied, blocks are distributed first not only to unoccupied cores, but also *to unoccupied clusters*, i.e. when a second thread block is added after the first one, it is placed on a core in another cluster than the first one. As we see in the figure, the activation of such a core cluster consumes 0.692W additional power. Once all clusters are activated, in this case after the fourth block, adding more thread blocks increases power only by the power of the additional core. On

[1] "Thread Processing Cluster"
[2] "Graphics Processing Clusters"

another note, the figure also illustrates how the activation of the very first core/cluster consumes even more power than for the remaining clusters. This extra power (3.34W) can be attributed to the activation of the global scheduler which distributes thread blocks to cores.

IV. Experimental Methodology

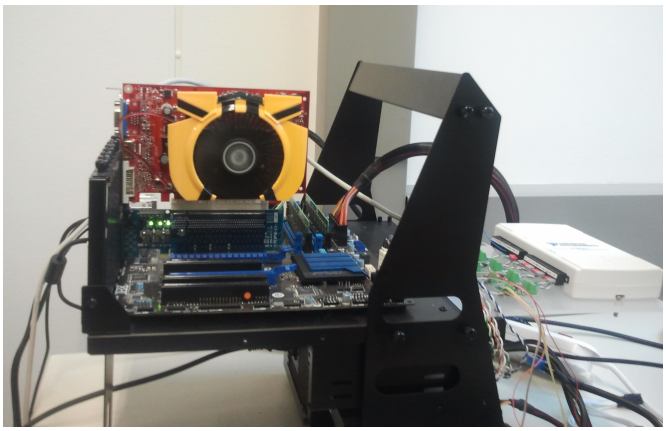


Fig. 5: A photograph of the power measurement testbed running the GT240 graphics card.

To validate our GPUSimPow power models for contemporary GPUs, we must compare the simulator output to the power consumption of real hardware for various GPGPU workloads. Thus, in this section, we present our experimental methodology, i.e. our test setup to measure power consumption both for validation of the simulator output as well as to infer power models for some of the irregular components. Besides the test setup, we also describe our test system configuration and the benchmark suite we used for evaluation.

A. Measurement Equipment

We developed our own measurement setup to validate the simulator against real GPUs. Figure 5 shows the hardware part of this setup with a GT240 card. Overall, the measurement setup consists of hardware and software components. On the hardware side, we are using a riser card with $20m\Omega$ probing resistors on the 12V and 3.3V rails to probe voltages and currents going to the GPU via the PCIe slot. The GTX580 also has two external PCIe power connector, to measure the power transmitted via these connectors we inserted $10m\Omega$ probing resistors into the PCIe power cables going to card. We designed a custom signal conditioning board to convert the voltages into signals that can be easily measured by off-the-shelf data acquisition (DAQ) hardware. A resistive divider is used to scale the voltages into the $0-5V$ range. The voltage drops over the sensing resistors are amplified and shifted into a usable common mode range using Analog Devices AD8210 Current Shunt Monitors. After this signal conditioning, the signals are sampled using a NI USB-6210 USB DAQ at a rate of $31.2kHz$. Our resistive voltage divider was built from 1% resistors and has a gain accuracy of $\pm 1.7\%$ and no offset error. The AD8210 has a gain accuracy of $\pm 0.5\%$ and an offset error of $\pm 1mV$ at its output. At $12V$, this offset error translates to an error of up to $60mW$ in power measurements. The error range of signal conditioning and measurement is thus $\pm 1.5\%$ for currents and $\pm 1.7\%$ for voltages. In the relevant -5 to $5V$ range, the DAQ has a specified gain accuracy of 0.0085% and an offset error of $0.1mV$. Not taking the small offset errors into account, overall, our system thus measures power within $\pm 3.2\%$. We developed a custom measurement tool that controls the DAQ and calculates power and energy from the measured voltages and currents. This tool is capable of using the GPU profiler to get start and end timestamps of the kernels running on the GPU. Using this information and the measured power waveform, the average power and amount of consumed energy can be calculated for each kernel execution.

Feature	GT240	GTX580
#Cores	12	16
#Threads per core	768	1536
#FUs per core	8	32
Uncore clock	550 MHz	882 MHz
Shader-to-Uncore	2.47×	2×
#Warps in-flight	24	48
Scoreboard	×	✓
L2-$ size	×	768KByte
Process node	40nm	40nm

TABLE II: Summary of the key features of the GPU architectures used in experimental evaluation.

B. System Configurations

For evaluating the output of the power simulator, we chose two real GPUs, the NVIDIA GT215 chip on a GeForce GT240 graphics card and the GF110 chip on a GeForce GTX580. Core parameters for both chips are given in Table II. The GT215 GPU is based on the GT200 Tesla design from 2009 and provides a good insight into many key features of modern GPUs. An initial advantage of using a GT200-based architecture is that the GPGPU-Sim simulator shows the highest correlation to real hardware for such architectures. The GF110 is based on the more recent Fermi architecture from 2010. The GT240 is a low-end card while the GTX580 is high-end enthusiast market card.

Feature	Measurement	Simulation
OS	Ubuntu 10.10	Ubuntu 10.10
Kernel	2.6.35-22	2.6.35-22
NVIDIA driver	304.43	-
CUDA version	3.1	3.1
GPGPU-sim base version	-	3.1.1
McPAT base version	-	0.8

TABLE III: Summary of our experimental setup.

We performed both measurements and simulations for a series of kernels selected from recent GPGPU benchmark suites (see next Section IV-C) for each of the two GPUs presented in Table II. For each kernel and GPU, we recorded hardware dynamic and static power, simulated dynamic and static power, and simulated as well as hardware execution

Name	#Kernels	Description	Origin
backprop	2	Multi-layer perceptron training	Rodinia
heartwall	1	Ultrasound image tracking	Rodinia
kmeans	2	k-means clustering	Rodinia
pathfinder	1	Dynamic programming path search	Rodinia
bfs	2	Breadth-first search	Rodinia
hotspot	1	Processor temperature estimation	Rodinia
matmul	1	Matrix-matrix multiplication	CUDA SDK
blackscholes	1	Black-Scholes PDE solver	CUDA SDK
mergesort	4	Parallel merge-sort	CUDA SDK
scalarprod	1	Scalar product of two vectors	CUDA SDK
vectoradd	1	Addition of two vectors	CUDA SDK

TABLE I: Overview over the various GPGPU benchmarks used for experimental evaluation.

time. Table III details the parameters of our experimental environment used to acquire the results. To estimate hardware static power, we ran the same benchmark at stock frequency and at a 20% lower frequency. Then, we performed linear extrapolation from the two data points to estimate the power the chip would consume at a frequency of 0 Hz. As Eq. 1 shows, there is no dynamic power consumption at 0 Hz and therefore, the result of the extrapolation must be equal to the static power of the chip. Unfortunately, using this methodology was only possible on the GT240 card, as the NVIDIA Linux drivers do not yet support changing the clock speed for the GTX580. Therefore, we estimate hardware static power for the GTX580 by measuring the idle power between two kernel executions and multiplying it by the ratio between idle power and static power we found on the GT240.

C. Benchmarks

The benchmarks whose kernels are used in our evaluation are shown in Table I. As the table reveals, all benchmarks originate either from the popular Rodinia benchmark suite [29] or from the CUDA SDK [11]. These 11 benchmarks span not only a variety of application domains, but, as Section V-B will show, an equally wide variety of algorithmic (and thus, dynamic power) characteristics. As our analysis focuses on the power consumed by the graphics card and GPU, we are only interested in the *GPGPU kernels* contained in each benchmark. The second column of Table I shows the number of different kernels in each benchmark. In some benchmarks, there are kernels with very short execution times (less than $500\mu s$). Because these kernels are too short for reliable measurements, we modified such benchmarks to execute the same kernels 100 times.

V. Results

In this section, we present the simulation results for the benchmarks described in the previous section and compare these results to measurements. In Section V-A, we describe the results from a per-benchmark perspective. Then, in Section V-B, we show how modeling the GPU on the architectural level enables code developers and chip architects to generate *power profiles* that break down the power to the individual components on the chip.

A. Simulated and Measured Power

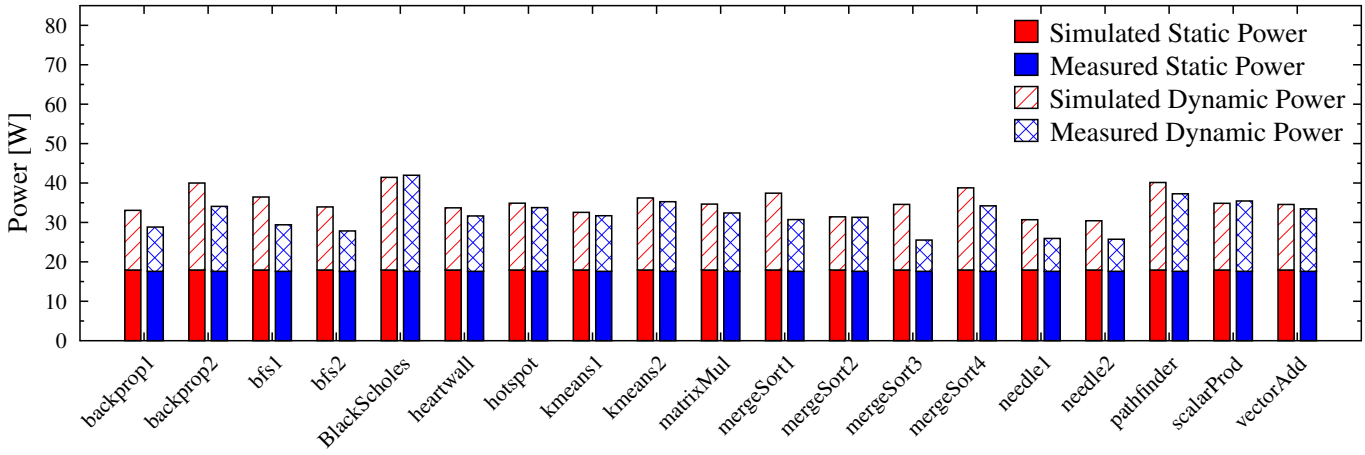
For each individual benchmark, we measured the total power consumed by the cards during the execution of each of the benchmark's kernels. For kernels that are executed multiple times during one benchmark run, we calculated arithmetic averages of all relevant power numbers. In the end, the power numbers we obtain are simulated and measured dynamic power and runtime *for each kernel* as well as simulated and measured static power *for the GPU*. As static power is consumed regardless of the circuit's switching activity, it is the same for each kernel. Table IV shows the results from the static power and area estimation for both GPUs.

		Static [W]	Area [mm^2]
GT240	Simulated	17.9	105
	Real	17.6	133
GTX580	Simulated	81.5	306
	Real	80	520

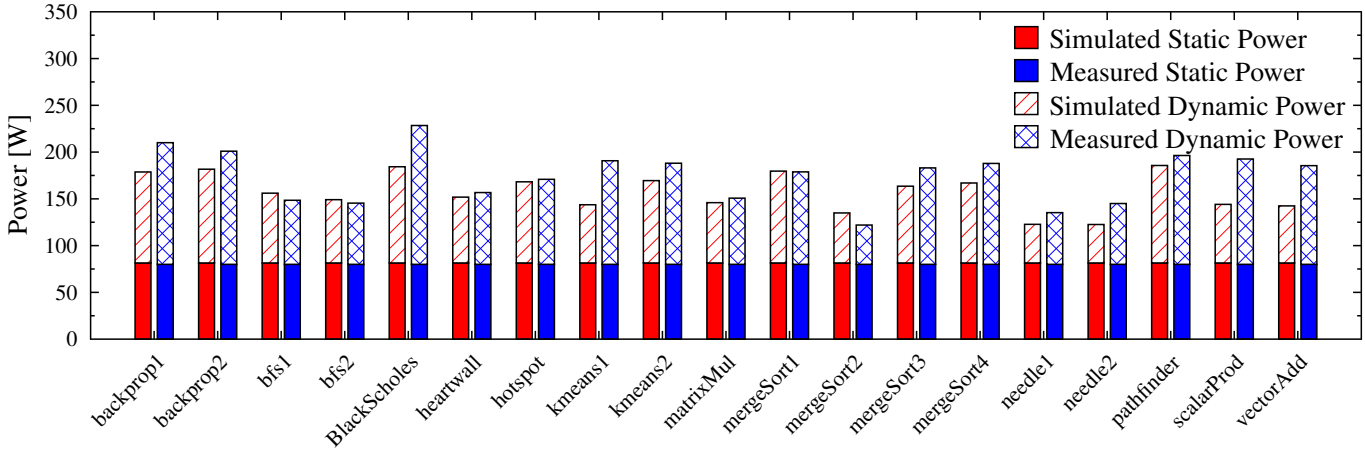
TABLE IV: Static Power and Area for GT240 and GTX580

The results of our experiments on the GT240 card are shown in Figure 6a. The figure shows total measured and simulated power for all benchmark kernels. Each total power bar in the figure is divided into a static power part common to all kernels and a kernel-specific dynamic power amount. Using the static power measurement technique explained in Section IV-B, we estimate the static power for GT240 to be 17.6 W. The card seems to do some power gating to reduce static power while no kernels are being executed. If no kernel was executed the card is using around 15 W, while for some milliseconds before and after the execution of a kernel the card consumes 19.5 W. About 90% of the power consumed by the card in this state thus seems to be static power. The GTX580 is using 90 W in the same state, so we estimate its static power to be 80 W.

In general, the figure shows strong similarity between the measurement and simulation results for most benchmarks. For all benchmarks but `BlackScholes` and `scalarProd` the simulator overestimates the power used by the card. When averaging errors, we always average the absolute value of errors, so that under- and overestimates can not cancel out. On average the simulation is 11.7% off from the real power consumed by the GPU, we call this average relative error. The maximum relative error of 35.4% occurs in the third `mergeSort` kernel. This is likely a measurement artifact. The



(a) GT240



(b) GTX580

Fig. 6: Measurement and simulation results for all benchmarks. Bars with the same benchmark name but different number, e.g. bfs1 and bfs2, correspond to different kernels of the same benchmark. Each bar shows the total runtime power, i.e. the sum of static and dynamic power.

execution time of the kernel is short (1 ms) and the benchmark could not easily be changed to call it multiple times because the kernel does in-place processing of its data.

In nearly every benchmark kernel, the simulator slightly overestimates the true power consumed by the chip. This trend is mostly caused by the estimation of dynamic power, while the simulation result of static power is just 0.3W (1.7%) larger than the real hardware power. Interestingly, the estimated chip area is slightly smaller than the actual chip area (105.0mm^2 vs. 133mm^2). Not considering static power, the average relative error in runtime dynamic power is 28.3%.

Figure 6b shows the results of our experiments using the GTX580 GPU. Again a strong similarity between measurements and simulations can be seen. Our empirically derived models also work well on this card even though they were obtained using the GT240 card. The average relative error for GTX580 is 10.8%. `scalarProd` is the kernel with the largest relative error (25.2%) on GTX580. The average relative error for the runtime dynamic power on GTX580 is 20.9%. GPUSimPow estimates the static power of GTX580 to be 81.5 W which almost completely matches our measurement result from the real hardware (80 W). As already explained in Section IV we could not use measurements at different clock speeds on GTX580 to estimate the static power. As a result of this limitation we used a different method to estimate the hardware static power. The better match of hardware static power with simulated static power could be a result of a more accurate static power estimate from GPUSimPow or it could be caused by the different hardware static power estimation methodology we used for GTX580.

B. *Power Profiling*

While relatively accurate estimations of dynamic and total power for the execution of a particular benchmark are helpful tools, in some cases, the *distribution* of power consumption over the hardware components on the GPU matters. As GPUSimPow contains hardware models for the internals of

each core, interface and controller on the GPU, it automatically produces detailed power statistics for these internals. Therefore, it is possible to generate a *power profile* for a particular benchmark kernel that breaks the overall power down to individual components with the desired level of accuracy. Table V shows such a power profile for the `blackscholes` benchmark. Please note that this table does not include the power consumed by the external DRAM (4.3 W).

		Static [W]	Dynamic [W]	Percent
GPU	Overall	17.934	19.207	100
	Cores	15.393	15.132	82.2
	NoC	1.484	1.229	7.3
	Memory Controller	0.497	1.753	6.1
	PCIe Controller	0.539	0.992	4.1
Core	Overall	1.283	1.031	100
	Base Power	0	0.199	8.6
	WCU	0.042	0.089	5.65
	Register File	0.112	0.173	12.3
	Execution Units	0.0096	0.556	24.43
	LDSTU	0.234	0.014	10.7
	Undiff. Core	0.886	0	38.3

TABLE V: Blackscholes benchmark power breakdown on GT240 for the individual components on the entire GPU (top) and a single GPU core (bottom).

In the top part of the table, both static and dynamic power for the top-level components on the GPU are shown. It can be seen that by far the largest fraction of total power is, as one would expect, consumed by the GPU cores (82.2%). Previous researchers have reported similarly high percentages, for example, in [30], the authors employed a simple, high-level power model to estimate the total core power to be 70% of the entire chip. According to the output of GPUSimPow, the next-most power after the cores themselves is consumed by the network on chip (7.3%), followed by the memory controllers (6.1%) for memory access and PCIe interfacing (4.1%). Note that some other top-level GPU structures such as the global scheduler and video decoder hardware are not modeled in detail and are therefore included in the (per-core) undifferentiated core and base power.

Increasing the level of detail, it is possible to look at the power consumed by the individual parts of a single core (bottom of Table V). Overall, the core consumes 2.31 W. As the table reveals, surprisingly, the largest fraction of the total power is attributed to the core base power and undifferentiated core (8.6% and 38.3%). While the former includes all the *per-core* components we can only model empirically due to lack of information (see Section III-D), the latter includes a per-core fraction of the *global* GPU components that can only be modeled empirically. Naturally, as we have no detailed models for undifferentiated components, we cannot generate any activity factors for them in GPGPU-Sim and thus the entire power consumption for the undifferentiated core is attributed as static power. Taking base power and undifferentiated core aside, the most power is consumed by the execution units (24.43%). After the execution hardware, the next-most power is used in the register file (about 12.3%). This number has been confirmed by previous research [30]. As one might expect from a SIMD architecture, the smallest part of the core power is consumed by the fetch and decode frontend, warp management, and schedulers (5.65%). GPUSimPow enables even more detailed analysis, e.g. investigating the power consumed by the different memories in the warp control unit or investigating the power impact of code sections with branch divergence on each hardware unit in detail. For reasons of conciseness, however, no such investigations are presented in this paper.

VI. Conclusion

Modern GPGPU designs are pervading many areas of research and industry because of the massive compute power they offer. While the development of such designs is trying to drive performance further, GPU chips are increasingly limited by the power they consume and dissipate as heat. With this problem of the power wall, the design of new GPGPU architectures has become even more complex than before as consumed power is now an additional variable in the design space.

In this work, we have demonstrated a novel power simulation framework entitled GPUSimPow. With GPUSimPow, programmers and computer architects can accurately estimate the static and dynamic power consumed by a given GPGPU architecture when executing a particular kernel *without building the actual chip*. As our evaluations on a set of well-known benchmarks have shown, the average relative error of our power simulation results compared to measurements on real hardware is 11.7% for GT240 and 10.8% for GTX580. The simulator is also able to generate the distribution of power consumption over the hardware components of the GPU, and also of the different components of each core. These power profiles can be used to drive architecture of application power optimization. However, as a power breakdown for a selected benchmark revealed, a large fraction of the simulated power is currently attributed to components that are not modeled in detail, i.e. “undifferentiated transistors”. More research needs to be done for creating accurate models of these components.

We conclude that in its current state, the GPUSimPow simulator is a helpful tool for both processor architects and GPGPU programmers to gain valuable insights into where power is consumed in the GPU. We hope that as architectural research with the tool begins to get started, the component power models will be further refined and new components added. It would also be interesting research to evaluate many of the recent extensions and enhancements to GPGPU architectures (such as simultaneous branch/warp interweaving [31], two-level scheduling [32] and dynamic warp subdivision [33]) from a power perspective.

VII. Acknowledgements

This project receives funding from the European Community’s Seventh Framework Programme [FP7/2007-2013] under the LPGPU Project (www.lpgpu.org), grant agreement n° 288653.

REFERENCES

[1] Embedded Systems Architecture Group at TU Berlin, "GPUSimPow Power Simulator," 2012. [Online]. Available: http://www.aes.tu-berlin.de/menue/forschung/projekte/gpusimpow_simulator/parameter/en/

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, ser. ISCA, 2000.

[3] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*, 2009.

[4] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies," in *Proceedings of the 35th Annual International Symposium on Computer Architecture, ISCA*, 2008.

[5] S. Hong and H. Kim, "An Integrated GPU Power and Performance Model," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA, 2010.

[6] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical Power Consumption Analysis and Modeling for GPU-based Computing," in *Proceedings of the Workshop on Power Aware Computing and Systems, HotPower*, 2009.

[7] K. Ramani, A. Ibrahim, and D. Shimizu, "PowerRed: A Flexible Power Modeling Framework for Power Efficiency Exploration in GPUs," in *Workshop on General Purpose Processing on Graphics Processing Units, GPGPU*, 2007.

[8] G. Wang, "Power Analysis and Optimizations for GPU Architecture Using a Power Simulator," in *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering, ICACTE*, 2010.

[9] Y. Wang and N. Ranganathan, "An Instruction-Level Energy Estimation and Optimization Methodology for GPU," in *Proceedings of the IEEE 11th International Conference on Computer and Information Technology*, ser. CIT '11, 2011.

[10] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical Power Modeling of GPU Kernels Using Performance Counters," in *Proceedings of the International Green Computing Conference*, 2010.

[11] NVIDIA, "CUDA: Compute Unified Device Architecture," 2007, http://developer.nvidia.com/object/gpucomputing.html.

[12] Khronos Group, "OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems," 2009, http://www.khronos.org/opencl/.

[13] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt, "Analyzing CUDA Workloads Using a Detailed GPU Simulator," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS*, 2009.

[14] N. Leischner, V. Osipov, and P. Sanders, "Fermi Architecture White Paper," 2009.

[15] AMD, "AMD Graphics Core Next GCN Architecture White Paper," 2012.

[16] C. Kun, S. Quan, and A. Mason, "A Power-Optimized 64-Bit Priority Encoder Utilizing Parallel Priority Look-Ahead," in *Proceedings of the 2004 International Symposium on Circuits and Systems, ISCAS*, 2004.

[17] B. W. Coon and J. E. Lindholm, "System and Method for Managing Divergent Threads Using Synchronization Tokens and Program Instructions that Include Set-Synchronization Bits," Patent US 7 543 136 B1, 2009.

[18] B. W. Coon, P. C. Mills, S. F. Oberman, and M. Y. Sui, "Tracking Register Usage During Multithreaded Processing Using a Scoreboard Having Separate Memory Regions and Storing Sequential Register Size Indicators," Patent US 7 434 032 B1, 2008.

[19] J. E. Lindholm, M. Y. Siu, S. S. Moy, L. S., and J. Nickolls, "Simulating Multiported Memories Using Lower Port Count Memories," Patent US 7 339 592, 2008.

[20] S. Galal and M. Horowitz, "Energy-Efficient Floating-Point Unit Design," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 913–922, july 2011.

[21] D. De Caro, N. Petra, and A. Strollo, "A High Performance Floating-Point Special Function Unit Using Constrained Piecewise Quadratic Approximation," in *Proceedings of the IEEE International Symposium on Circuits and Systems. ISCAS*, 2008.

[22] C. Galuzzi, C. Gou, H. Calderón, G. N. Gaydadjiev, and S. Vassiliadis, "High-bandwidth Address Generation Unit," *Journal of Signal Processing Systems*, vol. 57, pp. 33–44, 2009.

[23] H. Wong, M.-M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU Microarchitecture Through Microbenchmarking," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems Software, ISPASS*, 2010.

[24] L. Nyland, J. Nickolls, G. Hirota, and T. Mandal, "Systems and Methods for Coalescing Memory Accesses of Parallel Threads," Patent US 8 086 806 B2, 2011.

[25] B. W. Coon, M. Y. Siu, W. Xu, S. F. Oberman, J. R. Nickolls, and P. C. Mills, "Shared Memory with Parallel Access and Access Conflict Resolution Mechanism," Patent US 8 108 625 B1, 2012.

[26] Micron, "Calculating Memory System Power for DDR3," 2007. [Online]. Available: http://http://www.micron.com/products/dram/ddr3-sdram

[27] Hynix, "GDDR5 Datasheet," 2010. [Online]. Available: http://www.hynix.com/datasheet/pdf/graphics/H5GQ1H24AFR(Rev1.0).pdf

[28] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, sept–oct. 2011.

[29] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Proceedings of the IEEE International Symposium on Workload Characterization*, 2009.

[30] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, "Energy-Efficient Mechanisms for Managing Thread Context in Throughput Processors," in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ser. ISCA, 2011.

[31] N. Brunie, S. Collange, and G. Diamos, "Simultaneous Branch and Warp Interweaving for Sustained GPU Performance," in *Proceedings of the 39th International Symposium on Computer Architecture*, ser. ISCA, 2012.

[32] V. Narasiman, M. Shebanow, C. J. Lee, R. Miftakhutdinov, O. Mutlu, and Y. N. Patt, "Improving GPU Performance via Large Warps and Two-Level Warp Scheduling," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, 2011.

[33] J. Meng, D. Tarjan, and K. Skadron, "Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA, 2010.