

An Integrated GPU Power and Performance Model

Sunpyo Hong
Electrical and Computer Engineering
Georgia Institute of Technology
shong9@gatech.edu

Hyesoon Kim
School of Computer Science
Georgia Institute of Technology
hyesoon@cc.gatech.edu

ABSTRACT

GPU architectures are increasingly important in the multi-core era due to their high number of parallel processors. Performance optimization for multi-core processors has been a challenge for programmers. Furthermore, optimizing for power consumption is even more difficult. Unfortunately, as a result of the high number of processors, the power consumption of many-core processors such as GPUs has increased significantly.

Hence, in this paper, we propose an integrated power and performance (IPP) prediction model for a GPU architecture to predict the optimal number of active processors for a given application. The basic intuition is that when an application reaches the peak memory bandwidth, using more cores does not result in performance improvement.

We develop an empirical power model for the GPU. Unlike most previous models, which require measured execution times, hardware performance counters, or architectural simulations, IPP predicts execution times to calculate dynamic power events. We then use the outcome of IPP to control the number of running cores. We also model the increases in power consumption that resulted from the increases in temperature.

With the predicted optimal number of active cores, we show that we can save up to 22.09% of runtime GPU energy consumption and on average 10.99% of that for the five memory bandwidth-limited benchmarks.

Categories and Subject Descriptors

C.1.4 [Processor Architectures]: Parallel Architectures
; C.4 [Performance of Systems]: Modeling techniques
; C.5.3 [Computer System Implementation]: Microcomputers

General Terms

Measurement, Performance

Keywords

Analytical model, CUDA, GPU architecture, Performance, Power estimation, Energy

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ISCA'10, June 19–23, 2010, Saint-Malo, France.

Copyright 2010 ACM 978-1-4503-0053-7/10/06 ...\$10.00.

1. INTRODUCTION

The increasing computing power of GPUs gives them a considerably higher throughput than that of CPUs. As a result, many programmers try to use GPUs for more than just graphics applications. However, optimizing GPU kernels to achieve high performance is still a challenge. Furthermore, optimizing an application to achieve a better power efficiency is even more difficult.

The number of cores inside a chip, especially in GPUs, is increasing dramatically. For example, NVIDIA's GTX280 [2] has 30 streaming multiprocessors (SMs) with 240 CUDA cores, and the next generation GPU will have 512 CUDA cores [3]. Even though GPU applications are highly throughput-oriented, not all applications require all available cores to achieve the best performance.

In this study, we aim to answer the following important questions: Do we need *all cores* to achieve the highest performance? Can we save power and energy by using *fewer cores*?

Figure 1 shows performance, power consumption, and efficiency (performance per watt) as we vary the number of active cores.¹ The power consumption increases as we increase the number of cores. Depending on the circuit design (power gating, clock gating, etc.), the gradient of an increase in power consumption also varies. Figure 1(left) shows the performances of two different types of applications. In Type 1, the performance increases linearly, because applications can utilize the computing powers in all the cores. However, in Type 2, the performance is saturated after a certain number of cores due to bandwidth limitations [22, 23]. Once the number of memory requests from cores exceeds the peak memory bandwidth, increasing the number of active cores does not lead to a better performance. Figure 1(right) shows performance per watt. In this paper, the number of cores that shows the highest performance per watt is called the optimal number of cores.

In Type 2, since the performance does not increase linearly, using all the cores consumes more energy than using the optimal number of cores. However, for application Type 1, utilizing all the cores would consume the least amount of energy because of a reduction in execution time. The optimal number of cores for Type 1 is the maximum number of available cores but that of Type 2 is less than the maximum value. Hence, if we can predict the optimal number of cores at static time, either the compiler (or the programmer) can configure the number of threads/blocks² to utilize fewer cores, or hardware or a dynamic thread manager can use fewer cores.

To achieve this goal, we propose an integrated power and performance prediction system, which we call *IPP*. Figure 2 shows an overview of IPP. It takes a GPU kernel as an input and predicts both power consumption and performance together, whereas previ-

¹Active cores mean the cores that are executing a program.

²The term, *block*, is defined in the CUDA programming model.

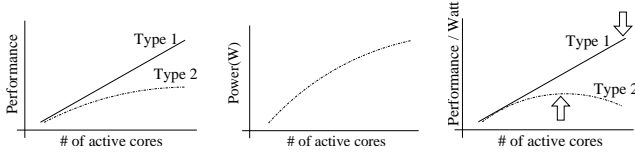


Figure 1: Performance, power, and efficiency vs. # of active cores (left: performance, middle: power, right: performance per watt)

ous analytical models predict only execution time or power. More importantly, unlike previous power models, IPP does not require architectural timing simulations or hardware performance counters; instead it uses outcomes of a timing model.

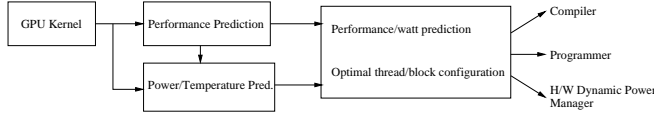


Figure 2: Overview of the IPP System

Using the power and performance outcomes, IPP predicts the optimal number of cores that results in the highest performance per watt. We evaluate the IPP system and demonstrate energy savings in a real GPU system. The results show that by using fewer cores based on the IPP prediction, we can save up to 22.09% and on average 10.99% of runtime energy consumption for the five memory bandwidth-limited benchmarks. We also estimate the amount of energy savings for GPUs that employ a power gating mechanism. Our evaluation shows that with power gating, the IPP can save on average 25.85% of the total GPU power for the five bandwidth-limited benchmarks.

In summary, our work makes the following contributions:

1. We propose what, to the best of our knowledge, is the first analytical model to predict performance, power and efficiency (performance/watt) of GPGPU applications on a GPU architecture.
2. We develop an empirical runtime power prediction model for a GPU. In addition, we also model the increases in runtime power consumption that resulted from the increases in temperature.
3. We propose the IPP system that predicts the optimal number of active cores to save energy.
4. We successfully demonstrate energy savings in a real system by activating fewer cores based on the outcome of IPP.

2. BACKGROUND ON POWER

Power consumption can be divided into two parts: dynamic power and static power, as shown in Equation (1).

$$Power = Dynamic_power + Static_power \quad (1)$$

Dynamic power is the switching overhead in transistors, so it is determined by runtime events. Static power is mainly determined by circuit technology, chip layout and operating temperature.

2.1 Building a Power Model Using an Empirical Method

Isaci and Martonosi [12] proposed an empirical method to building a power model. They measured and modeled the Intel Pentium 4 processor.

$$Power = \sum_{i=0}^n (AccessRate(C_i) \times ArchitecturalScaling(C_i) \times MaxPower(C_i) + NonGatedClockPower(C_i)) + IdlePower \quad (2)$$

Equation (2) shows the basic power model discussed in [12]. It consists of the idle power plus the dynamic power for each hardware component, where the *MaxPower* and *ArchitecturalScaling* terms are heuristically determined. For example, *MaxPower* is empirically determined by running several training benchmarks that stress fewer architectural components. Access rates are obtained from performance counters. They indicate how often an architectural unit is accessed per unit of time, where one is the maximum value.

2.2 Static Power

As the technology is scaled, static power consumption is increased [4]. To understand static power consumption and temperature effects, we briefly describe static power models.

Butts and Sohi [6] presented the following simplified leakage power model for an architecture-level study.

$$P_{static} = V_{cc} \cdot N \cdot K_{design} \cdot \hat{I}_{leak} \quad (3)$$

V_{cc} is the supply voltage, N is the number of transistors in the design, and K_{design} is a constant factor that represents the technology characteristics. \hat{I}_{leak} is a normalized leakage current for a single transistor that depends on V_{th} , which is the threshold voltage. Later, Zhang et al. [24] improved this static power model to consider temperature effects and operating voltages in *HotLeakage*, a software tool. In their model, K_{design} is no longer constant and depends on temperature, where \hat{I}_{leak} is a function of temperature and supply voltage. The leakage current can be expressed as shown in Equation (4).

$$\hat{I}_{leak} = \mu_0 \cdot C_{OX} \cdot \frac{W}{L} \cdot e^{b(V_{dd} - V_{dd0})} \cdot v_t^2 \cdot (1 - e^{-\frac{V_{dd}}{v_t}}) \cdot e^{-\frac{|V_{th}| - V_{off}}{n \cdot v_t}} \quad (4)$$

v_t is the thermal voltage that is represented by kT/q , and it depends on temperature. The threshold voltage, V_{th} , is also a function of temperature. Since v_t^2 is the dominant temperature-dependent factor in Equation (4), the leakage power quadratically increases with temperature. However, in a normal operating temperature range, the leakage power can be simplified as a linear model [21].

3. POWER AND TEMPERATURE MODELS

3.1 Overall Model

GPU power consumption (*GPU_power*) is modeled similar to Equation (2) [12] in Section 2. The *GPU_power* term consists of *Runtime_power* and *IdlePower* terms, as shown in Equation (5). The *NonGatedClockPower* term is not used in this model, because the evaluated GPUs do not employ clock gating. *IdlePower* is the power consumption when a GPU is on but no application is running. *Runtime_power* is the additional power consumption required to execute programs on a GPU. It is the sum of runtime-powers from all SMs (*RP_SMs*) and GDDR memory (*RP_Memory*).

Table 1: List of instructions that access each architectural unit

| PTX Instruction | Architectural Unit | Variable Name |
|---|----------------------|------------------|
| add_int sub_int addc_int subc_int sad_int div_int rem_int abs_int mul_int mad_int mul24_int mad24_int min_int neg_int | Int. arithmetic unit | RP_Int |
| add_fp sub_fp mul_fp fma_fp neg_fp min_fp lg2_fp ex2_fp mad_fp div_fp abs_fp | Floating point unit | RP_Fp |
| sin_fp cos_fp rcp_fp sqrt_fp rsqrt_fp | SFU | RP_Sfu |
| xor cnot shl shr mov cvt set setp selp slct and or | ALU | RP_Alu |
| st_global ld_global | Global memory | RP_GlobalMem |
| st_local ld_local | Local memory | RP_LocalMem |
| tex | Texture cache | RP_Texture_Cache |
| ld_const | Constant cache | RP_Const_Cache |
| ld_shared st_shared | Shared memory | RP_Shared |
| setp selp slct and or xor shr mov cvt st_global ld_global ld_const add mad24 sad div rem abs neg shl min sin cos rcp sqrt rsqrt set mul24 sub addc subc mul mad cnot ld_shared st_local ld_local tex | Register file | RP_Reg |
| All instructions | FDS (Fetch/Dec/Sch) | RP_FDS |

$$GPU_power = Runtime_power + IdlePower \quad (5)$$

$$Runtime_power = \sum_{i=0}^n RP_Component_i \quad (6)$$

$$= RP_SMs + RP_Memory$$

3.2 Modeling Power Consumption from Streaming Multiprocessors

In order to model the runtime power of SMs, we decompose the SM into several physical components, as shown in Equation (7) and Table 1. The texture and constant caches are included in the $SM_Component$ term, because they are shared between multiple SMs in the evaluated GPU system. One texture cache is shared by three SMs, and each SM has its own constant cache. RP_Const_SM is a constant runtime power component for each active SM. It models power consumption from several units, including I-cache, and the frame buffer, which always consume relatively constant amount of power when a core is active.

$$\sum_{i=0}^n SM_Component_i = RP_Int + RP_Fp + RP_Sfu \quad (7)$$

$$+ RP_Alu + RP_Texture_Cache + RP_Const_Cache$$

$$+ RP_Shared + RP_Reg + RP_FDS + RP_Const_SM$$

$$RP_SMs = Num_SMs \times \sum_{i=0}^n SM_Component_i \quad (8)$$

Num_SMs : Total number of SMs in a GPU

Table 1 summarizes the modeled architectural components used by each instruction type and the corresponding variable names in Equation (7). All instructions access the FDS unit (Fetch/Dec/Sch). For the register unit, we assume that all instructions accessing the register file have the same number of register operands per instruction to simplify the model. The exact number of register accesses

per instruction depends on the instruction type and the number of operands, but we found that the power consumption difference due to the number of register operands is negligible.

Access Rate: As Equation 2 shows, dynamic power consumption is dependent on access rate of each hardware component. Isci and Martonosi used a combination of hardware performance counters to measure access rates [12]. Since GPUs do not have any speculative execution, we can estimate hardware access rates based on the dynamic number of instructions and execution times without hardware performance counters.

Equation (9) shows how to calculate the runtime power for each component (RP_{comp}) such as RP_Reg . RP_{comp} is the multiplication of $AccessRate_{comp}$ and $MaxPower_{comp}$. $MaxPower_{comp}$ is described in Table 2 and will be discussed in Section 3.4. Note that RP_Const_SM is not dependent on $AccessRate_{comp}$.

Equation (10) shows how to calculate the access rate for each component, $AccessRate_{comp}$. The dynamic number of instructions per component ($DAC_per_th_{comp}$) is the sum of instructions that access an architectural component. $Warps_per_SM$ indicates how many warps³ are executed in one SM. We divide execution cycles by four because one instruction is fetched, scheduled, and executed every four cycles. This normalization also makes the maximum value of the $AccessRate_{comp}$ term be one.

$$RP_{comp} = MaxPower_{comp} \times AccessRate_{comp} \quad (9)$$

$$AccessRate_{comp} = \frac{DAC_per_th_{comp} \times Warps_per_SM}{Exec_cycles/4} \quad (10)$$

$$DAC_per_th_{comp} = \sum_{i=0}^n Number_Inst_per_warps_i(comp) \quad (11)$$

$$Warps_per_SM = \left(\frac{\#Threads_per_block}{\#Threads_per_warp} \times \frac{\#Blocks}{\#Active_SMs} \right) \quad (12)$$

3.3 Modeling Memory Power

The evaluated GPU system has five different memory spaces: global, shared, local, texture, and constant. The shared memory space uses a software managed cache that is inside an SM. The texture and constant memories are located in the GDDR memory, but they mainly use caches inside an SM. The global memory and the local memory are sharing the same physical GDDR memory, hence RP_Memory considers both. Shared, constant, and texture memory spaces are modeled separately as SM components.

$$RP_Memory = \sum_{i=0}^n Memory_component_i \quad (13)$$

$$= RP_GlobalMem + RP_LocalMem$$

3.4 Power Model Parameters

To obtain the power model parameters, we design a set of synthetic microbenchmarks that stress different architectural components in the GPU. Each microbenchmark has a loop that repeats a certain set of instructions. For example, the microbenchmark that stresses FP units contains a high ratio of FP instructions.

The optimum set of $MaxPower_{comp}$ values in Equation (9) that minimize the error between the measured power and the outcome of the equation is searched. However, to avoid searching through a large space of values, the initial seed value for each architecture unit is estimated based on the relative physical die sizes of the unit [12]. Table 2 shows the parameters used for $MaxPower_{comp}$. Eight

³Warp is a group of threads that are fetched/executed together inside the GPU architecture.

power components require a special piecewise linear approach [12]: an initial increase from idle to relatively low access rate causes a large granularity of increase in power consumption while a further increase causes a smaller increase. Spec.Linear column indicates whether the $AccessRate_{comp}$ term in Equation (9) needs to be replaced with the special piecewise linear access rate based on the following simple conversion; $0.1365 * \ln(AccessRate_{comp}) + 1.001375$. The parameters in this conversion are empirically determined to have a piecewise linear function.

Table 2: Empirical power parameters

| Units | MaxPower | OnChip | Spec.Linear |
|---------------------|----------|--------|-------------|
| FP | 0.2 | Yes | Yes |
| REG | 0.3 | Yes | Yes |
| ALU | 0.2 | Yes | No |
| SFU | 0.5 | Yes | No |
| INT | 0.25 | Yes | Yes |
| FDS (Fetch/Dec/Sch) | 0.5 | Yes | Yes |
| Shared memory | 1 | Yes | No |
| Texture cache | 0.9 | Yes | Yes |
| Constant cache | 0.4 | Yes | Yes |
| Const_SM | 0.813 | Yes | No |
| Global memory | 52 | No | Yes |
| Local memory | 52 | No | Yes |

Figure 3 shows how the overall power is distributed among the individual architectural components for all the evaluated benchmarks (Section 5 presents the detailed benchmark descriptions and the evaluation methodology). On average, the memory, idle power, and RP_{Const_SM} consume more than 60% of the total GPU power. REG and FDS also consume relatively higher power than other components because almost all instructions access these units.

3.5 Active SMs vs. Power Consumption

To measure the power consumption of each SM, we design another set of microbenchmarks to control the number of active SMs. These microbenchmarks are designed such that only one block can be executed in each SM, thus as we vary the number of blocks, the number of active SMs varies as well. Even though the evaluated GPU does not employ power gating, idle SMs do not consume as much power as active SMs do because of low-activity factors [18] (i.e., idle SMs do not change values in circuits as often as active SMs). Hence, there are still significant differences in the total power consumption depending on the number of active SMs in a GPU.

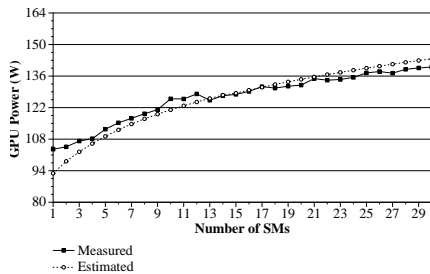


Figure 4: Power consumption vs. active SMs

Figure 4 shows an increase in power consumption as we increase the number of active SMs. The maximum power delta between using only one SM versus all SMs is 37W. Since there is no power gating, the power consumption does not increase linearly as we increase the number of SMs. We use a log-based model instead of a linear curve, as shown in Equation (14). We also model the memory power consumption following the exact same log-based

trend although it is not directly dependent on the number of active SMs. Finally, runtime power can be modeled by taking the number of active SMs as shown in Equation (16)

$$RP_{SMs} = Max_SM \times \log_{10}(\alpha \times Active_SMs + \beta) \quad (14)$$

$$Max_SM = (Num_SMs \times \sum_{i=0}^n SM_Component_i) \quad (15)$$

$$\alpha = (10 - \beta) / Num_SMs, \beta = 1.1$$

$$Runtime_power = (Max_SM + RP_Memory) \times \log_{10}(\alpha \times Active_SMs + \beta) \quad (16)$$

$Active_SMs$: Number of active SMs in the GPU

3.6 Temperature Model

CPU Temperature models are typically represented by an RC model [20]. We determine the model parameters empirically by using a step function experiment. Equation (17) models the rising temperature, and Equation (18) models the decaying temperature.

$$Temperature_{rise}(t) = Idle_temp + \delta \left(1 - e^{-t/RC_{Rise}}\right) \quad (17)$$

$$Temperature_{decay}(t) = Idle_temp + \gamma \left(e^{-t/RC_{Decay}}\right) \quad (18)$$

$$\delta = Max_temp - Idle_temp, \gamma = Decay_temp - Idle_temp$$

$Idle_temp$: Idle operating chip temperature

Max_temp : Maximum temperature, which depends on runtime power

$Decay_temp$: Chip temperature right before decay

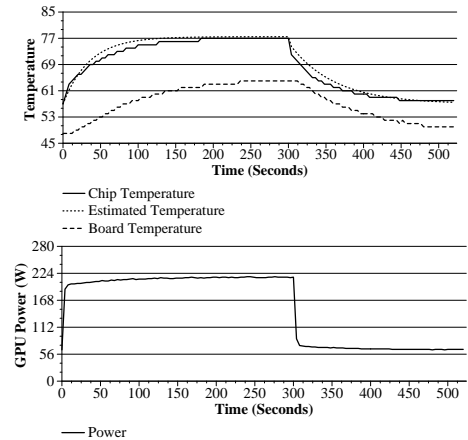


Figure 5: Temperature effects from power, (top): Measured and estimated temperature, (bottom): Measured power

Figure 5 shows estimated and measured temperature variations. Both the chip temperature and the board temperature are measured with the built-in sensors in the GPU. Max_temp is a function of runtime power, which depends on application characteristics. We discovered that the chip temperature is strongly affected by the rate of GDDR memory accesses, not only runtime power consumption. Hence, the maximum temperature is modeled with a combination of them as shown in Equation (19). The model parameters are described in Table 3. Note that $Memory_Insts$ includes global and local memory instructions.

$$Max_temp(Runtime_Power) = (\mu \times Runtime_Power) + \lambda + \rho \times MemAccess_intensity \quad (19)$$

$$MemAccess_intensity = \frac{Memory_Insts}{NonMemory_Insts} \quad (20)$$

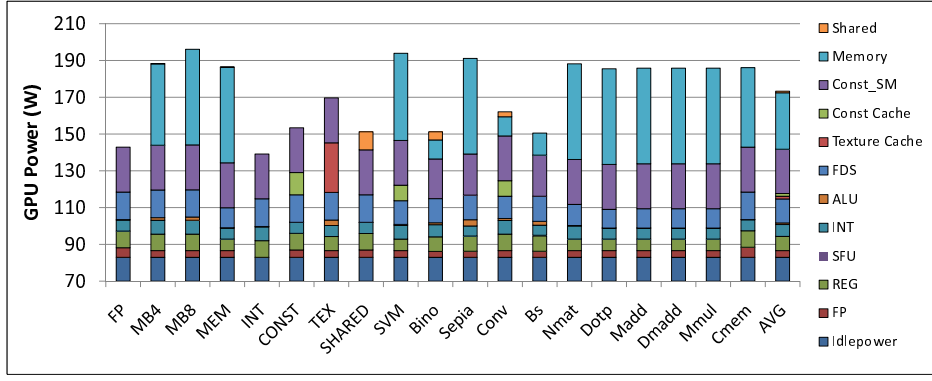


Figure 3: Power breakdown graph for all the evaluated benchmarks

Table 3: Parameters for GTX280

| Parameter | Value |
|-----------|--------|
| μ | 0.120 |
| λ | 5.5 |
| ρ | 21.505 |
| RC_Rise | 35 |
| RC_Decay | 60 |

3.7 Modeling Increases in Static Power Consumption

Section 2.2 discussed the impact of temperature on static power consumption. Because of the high number of processors in the GPU chip, we observe an increase in runtime power consumption as the chip temperature increases, as shown in Figure 6. To consider increases in static power consumption, we include the temperature model (Equations (17) and (18)) into the runtime power consumption model. We use a linear model to represent increases in static power as discussed in Section 2.2. Since we cannot control the operating voltage of the evaluated GPUs at runtime, we only consider operating temperature effects.

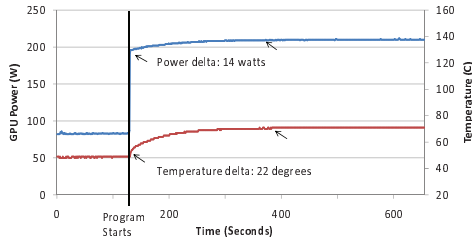


Figure 6: Static power effects

Figure 6 shows that power consumption increases gradually over time after an application starts,⁴ and the delta is 14 watts. This delta could be caused by increases in static power consumption or additional fan power. By manually controlling the fan speed from lowest to highest, we measure that the additional fan power consumption is only 4W. Hence, the remaining 10 watts of the power consumption increase is modeled as the additional static power

⁴The initial jump of power consumption exists when an application starts.

increase that resulted from the increases in temperature. Equation (21) shows the comprehensive power equation over time that includes the increased static power consumption, which depends on σ (the ratio of power delta over temperature delta ($\sigma = 10 / 22$)). Note that $Runtime_power_0$ is an initial power consumption obtained from (16), and the model assumes a cold start (i.e., the system is in the idle state). $Temperature(t)$ in (23) is obtained from (17) or (18).

$$GPU_power(t) = Runtime_power(t) + IdlePower \quad (21)$$

$$Runtime_power(t) = Runtime_power_0 + \sigma \times Delta_temp(t) \quad (22)$$

$$Delta_temp(t) = Temperature(t) - Idle_temp \quad (23)$$

4. IPP: INTEGRATED POWER AND PERFORMANCE MODEL

In this section, we describe the integrated power and performance model to predict performance per watt and the optimal number of active cores. The integrated power and performance model (IPP) uses predicted execution times to predict power consumption instead of measured execution times.

4.1 Execution Time and Access Rate Prediction

In Section 3, we developed the power model that computes access rates by using measured execution time information. Predicting power at static time requires access rates in advance. In other words, we also need to predict the execution time of an application to predict power. We used a recently developed GPU analytical timing model [9] to predict the execution time. The model is briefly explained in this section. Please refer to the analytical timing model paper [9] for the detailed descriptions.

In the timing model, the total execution time of a GPGPU application is calculated with one of Equations (24), (25), and (26) based on the number of running threads, MWP, and CWP in the application. MWP represents the number of memory requests that can be serviced concurrently and CWP represents the number of warps that can finish one computational period during one memory access period. N is the number of running warps. Mem_L is the average memory latency (430 cycles for the evaluated GPU architecture). Mem_cycles is the processor waiting cycles for memory operations. $Comp_cycles$ is the execution time of all instructions. $Repw$ is the number of times that each SM needs to repeat the same set of computation.

Case1: If (MWP is N warps per SM) and (CWP is N warps per SM)

$$(Mem_cycles + Comp_cycles + \frac{Comp_cycles}{\#Mem_insts} \times (MWP - 1))(\#Repw) \quad (24)$$

Case2: If (CWP >= MWP) or (Comp_cycles > Mem_cycles)

$$(Mem_cycles \times \frac{N}{MWP} + \frac{Comp_cycles}{\#Mem_insts} \times (MWP - 1))(\#Repw) \quad (25)$$

Case3: If (MWP > CWP)

$$(Mem_L + Comp_cycles \times N)(\#Repw) \quad (26)$$

IPP calculates the AccessRate using Equation (27), where the predicted execution cycles (*Predicted_Exec_Cycles*) are calculated with one of the Equations (24),(25), and (26).

$$AccessRate_{comp} = \frac{DAC_per_th_{comp} \times Warps_per_SM}{Predicted_Exec_Cycles/4} \quad (27)$$

4.2 Optimal Number of Cores for Highest Performance/Watt

IPP predicts the optimal number of SMs that would achieve the highest performance/watt. As we showed in Figure 1, the performance of an application either increases linearly (in this case, the optimal number of SMs is always the maximum number of cores) or non-linearly (the optimal number of SMs is less than the maximum number of cores). Performance per watt can be calculated by using Equation (28).

$$perf.\ per\ watt(cores) = \frac{(work/execution\ time(\#\ of\ cores))}{(power(\#\ of\ cores))} \quad (28)$$

Equations (24),(25), and (26) calculate execution times. Among the three cases, only Case 2 has a memory bandwidth limited case. Case 1 is used when there are not enough number of running threads in the system, and Case 3 models when an application is computationally intensive. So both Cases 1 and 3 would never reach the peak memory bandwidth. To understand the memory bandwidth limited case, let's look at MWP more carefully. The following equations show the steps in calculating MWP. *MWP* is the number of memory requests that can be serviced concurrently. As shown in Equation(29), *MWP* is the minimum of *MWP_Without_BW*, *MWP_peak_BW*, and *N*. *N* is the number of running warps. If there are not enough warps, *MWP* is limited by the number of running warps. If an application is limited by memory bandwidth, *MWP* is determined by *MWP_peak_BW*, which is a function of a memory bandwidth and the number of active SMs. Note that *Departure_delay* represents the pipeline delay between two consecutive memory accesses, and it is dependent on both the memory system and the memory access types (coalesced or uncoalesced) in applications.

$$MWP = MIN(MWP_Without_BW, MWP_peak_BW, N) \quad (29)$$

$$MWP_peak_BW = \frac{Mem_Bandwidth}{BW_per_warp \times \#ActiveSM} \quad (30)$$

$$BW_per_warp = \frac{Freq \times Load_bytes_per_warp}{Mem_L} \quad (31)$$

$$MWP_Without_BW_full = Mem_L/Departure_delay \quad (32)$$

$$MWP_Without_BW = MIN(MWP_Without_BW_full, N) \quad (33)$$

We could calculate $\frac{d(perf.\ per\ watt(\#\ of\ active\ cores))}{d(\#\ of\ active\ cores)} = 0$ to find the optimal number of cores. However, we observed that once *MWP_peak_BW* reaches *N*, the application usually reaches the peak bandwidth. Hence, based on Equation (30), we conclude that the optimal number of cores can be calculated using the following equations to simplify the calculation.

$$\begin{aligned} & \text{if (1) (MWP == N) or (CWP == N) or} & (34) \\ & \text{(2) MWP > CWP or} \\ & \text{(3) MWP < MWP_peak_BW} \\ & \text{Optimal \# of cores = Maximum available \# of cores} \\ & \text{else} \end{aligned}$$

$$Optimal\ \#\ of\ cores = \frac{Mem_Bandwidth}{(BW_per_warp) \times N}$$

4.3 Limitations of IPP

IPP requires both power and timing models thereby inheriting the limitations from them. Some examples of limitations include the following: control flow intensive applications, asymmetric applications, and texture cache intensive applications.

IPP also requires an instruction information. However, IPP does not require an actual number of total instructions. It calculates only access rates that can be easily normalized with an input data size. Nonetheless, if an application shows a significantly different behavior depending on input sizes, IPP needs to consider the input size effects, which will be addressed in our future work.

4.4 Using Results of IPP

In this paper, we constrain the number of active cores based on an output of IPP by only limiting the number of blocks inside an application, since we cannot change the hardware or the thread scheduler. If the number of active cores can be directly controlled by hardware or by a runtime thread scheduler, compilers or programmers do not have to change their applications to utilize fewer cores. Instead, IPP only passes the information of the number of optimal cores to the runtime system, and either the hardware or runtime thread scheduler enables only the required number of cores to save energy.

5. METHODOLOGY

5.1 Power and Temperature Measurement

The NVIDIA GTX280 GPU, which has 30 SMs and uses a 65nm technology, is used in this work. We use the Extech 380801 AC/DC Power Analyzer [1] to measure the overall system power consumption. The raw power data is sent to a data-log machine every 0.5 second. Each microbenchmark executes for an average of 10 seconds.

Since we measure the input power to the entire system, we have to subtract *Idlepower_System* (159W) from the total system input power to obtain *GPU_Power*.⁵ The *Idle_Power* value for the evaluated GPU is 83W. The GPU temperature is measured with the *nvclock* utility [16]. The command "nvclock -i" outputs board and chip temperatures. Temperature is measured every second.

⁵IdlePower_System is obtained by measuring system power with another GPU card whose idle power is known

5.2 Benchmarks

To test the accuracy of our IPP system, we use the Merge benchmarks [15, 9], five additional memory bandwidth-limited benchmarks (Nmat, Dotp, Madd, Dmadd, and Mmul), and one computational intensive (i.e., non-memory bandwidth limited) benchmark (Cmem). Table 4 describes each benchmark and summarizes the characteristics of them.

To calculate the number of dynamic instructions, we use a GPU PTX emulator, Ocelot [13]. It also classifies instruction types.

6. RESULTS

6.1 Evaluation of Runtime Power Model

Figure 7 compares the predicted power consumption with the measured power value for the microbenchmarks. According to Figure 3, the global memory consumes the most amount of power. MB4, MB8, and MEM benchmarks consume much greater power than the FP benchmark, which consists of mainly floating point instructions. Surprisingly, the benchmarks that use texture cache or constant cache also consume high power. This is because both the texture cache and the constant cache have higher *MaxPower* than that of the FP unit. The geometric mean of the error in the power prediction for microbenchmark is 2.5%. Figure 8 shows the access rates for each microbenchmark. When an application does not have many memory operations such as the FP benchmark, dynamic access rates for FP or REG can be very close to one. FDS is one when an application reaches the peak performance of the machine.

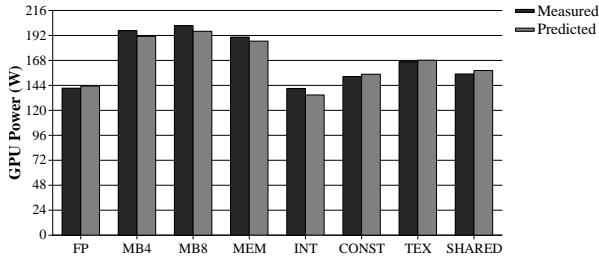


Figure 7: Comparison of measured and predicted GPU power consumption for the microbenchmarks

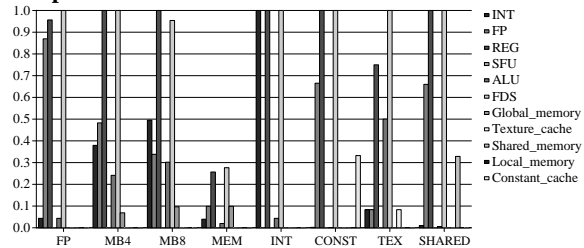


Figure 8: Dynamic access rate of the microbenchmarks

Figure 9 compares the predicted power and the measured power consumptions for the evaluated GPGPU kernels. The geometric mean of the power prediction error is 9.18% for the GPGPU kernels. Figure 10 shows the dynamic access rates. The complete breakdown of the GPU power consumption is shown in Figure 3. Bino and Conv have lower global memory access rates than others, which results in less power consumption than others. Sepia and Bs are high performance applications. This explains why they have high REG and FDS values. All the memory bandwidth-limited benchmarks have higher power consumption even though they have relatively lower FP/REG/FDS access rates.

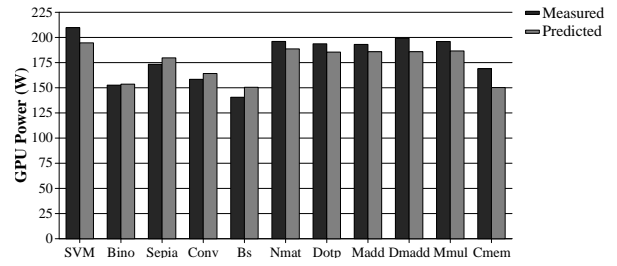


Figure 9: Comparison of measured and predicted GPU power consumption for the GPGPU kernels

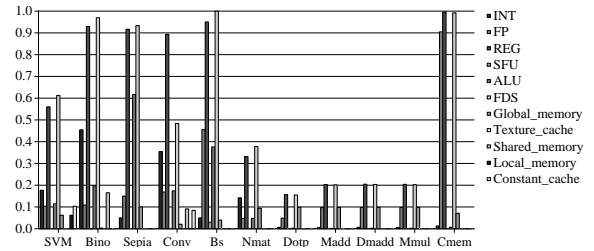


Figure 10: Dynamic access rate of the GPGPU kernels

6.2 Temperature Model

Figure 11 displays the predicted chip temperature over time for all the evaluated benchmarks. The initial temperature is 57°C , the typical GPU cold state temperature in our evaluated system. The temperature is saturated after around 600 secs. The peak temperature depends on the peak run-time power consumption, and it varies from 68°C (the INT benchmark) to 78°C (SVM). Based on Equation (22), we can predict that the runtime power of SVM would increase by 10W after 600 seconds. However, for the INT benchmark, it would increase by only 5W after 600 seconds.

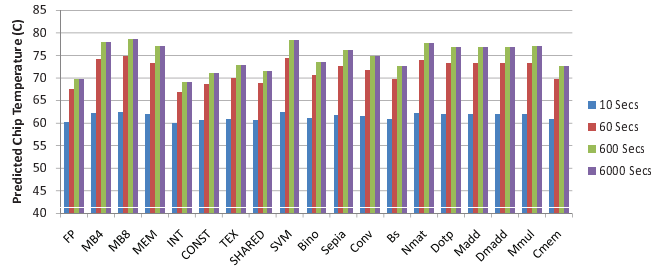


Figure 11: Peak temperature prediction for the benchmarks. Initial temperature: 57°C

6.3 Power Prediction Using IPP

Figure 12 shows the power prediction of IPP for both the microbenchmarks and the GPGPU kernels. These are equivalent to the experiments in Section 6.1. The main difference is that Section 6.1 requires measured execution times while IPP uses predicted times using the equations in Section 4. Using predicted times could have increased the error in prediction of power values, but since the error of timing model is not high, the overall error of the IPP system is not significantly increased. The geometric mean of the power prediction of IPP is 8.94% for the GPGPU kernels and 2.7% for the microbenchmarks, which are similar to using real execution time measurements.

Table 4: Characteristics of the Evaluated Benchmarks (AI means arithmetic intensity.)

| Benchmark | Description | Peak Bandwidth (GB/s) | MWP | CWP | AI |
|-----------------------|--|--------------------------------|--------|--------|---------|
| SVM [15] | Kernel from a SVM-based algorithm | 54.679 (non-bandwidth limited) | 5.875 | 11.226 | 11.489 |
| Binomial(Bino) [15] | American option pricing | 3.689 (non-bandwidth limited) | 14.737 | 1.345 | 314.306 |
| Sepia [15] | Filter for artificially aging images | 12.012 (non-bandwidth limited) | 12 | 12 | 8.334 |
| Convolve(Conv) [15] | 2D Separable image convolution | 16.208 (non-bandwidth limited) | 10.982 | 3.511 | 43.923 |
| Blackscholes(Bs) [17] | European option pricing | 51.033 (non-bandwidth limited) | 3 | 5.472 | 24.258 |
| Matrixmul(Nmat) | Naive version of matrix multiplication | 123.33 (bandwidth limited) | 10.764 | 32 | 3.011 |
| Dotp | Matrix dotproduct | 111.313 (bandwidth limited) | 10.802 | 16 | 0.574 |
| Madd | Matrix multiply-add | 115.058 (bandwidth limited) | 10.802 | 16 | 1.049 |
| Dmadd | Matrix double memory multiply add | 109.996 (bandwidth limited) | 10.802 | 16 | 1.0718 |
| Mmul | Matrix single multiply | 114.997 (bandwidth limited) | 10.802 | 16 | 1.060 |
| Cmem | Matrix add FP operations | 64.617 (non-bandwidth limited) | 10.802 | 9.356 | 12.983 |

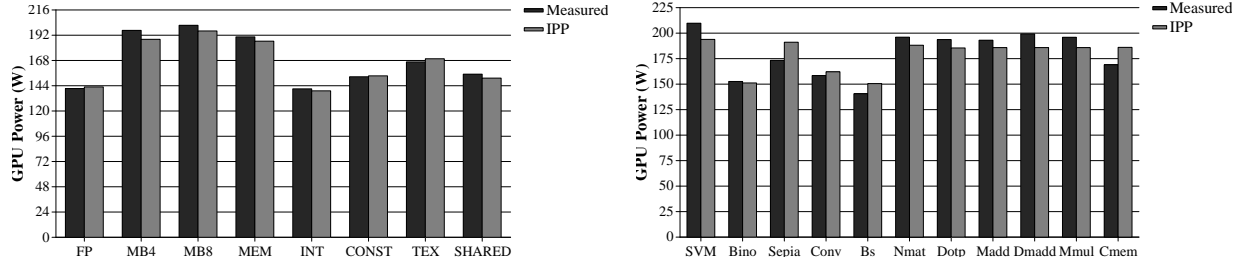


Figure 12: Comparison of measured and IPP predicted GPU power comparison (Left:Microbenchmarks, Right:GPGPU kernels)

6.4 Performance and Power Efficiency Prediction Using IPP

Based on the conditions in Equation (34), we identify the benchmarks that reach the peak memory bandwidth. The five merge benchmarks do not reach the peak memory bandwidth as shown in Table 4. CWP values in Bino, Sepia and Conv are equal to or less than the MWP values of them, so these benchmarks cannot reach the peak memory bandwidth. Both SVM’s MWP (5.878) and Bs’s MWP (3) are less than MWP_peak_BW (10.8). Thus they cannot reach the peak memory bandwidth also.

To further evaluate our IPP system, we use the benchmarks that reach the peak memory bandwidth (the 3rd column in Table 4 shows the average memory bandwidth of each application). We also include one non-bandwidth limited benchmark (Cmem) for a comparison. In this experiment, we vary the number of active cores by varying the number of blocks in the CUDA applications. We design the applications such that one SM executes only one block. Note that, all different configurations (in this section) of one application have the exact same amount work. So, as we use fewer cores (i.e., fewer blocks), each core (or block) executes more number of instructions. We use Giga Instructions Per Sec (GIPS)⁶ instead of Gflops/s for a metric.

Figure 13 shows how GIPS varies with the number of active cores for both the actual measured data and the predictions of IPP. Only Cmem has a linear performance improvement in both the measured data and the predicted values. The rest of the benchmarks show a nearly saturated performance as we increase the number of active cores. IPP still predicts GIPS values accurately except for Cmem. Although the predicted performance of Cmem does not exactly match the actual performance, IPP still correctly predicts the trend. Nmat shows higher performance than other bandwidth limited benchmarks, because it has a higher arithmetic intensity.

Figure 14 shows the actual bandwidth consumption of the experiment in Figure 13. Cmem shows a linear correlation between

⁶We decide to use GIPS instead of Gflop/s because the performance efficiency should include non-floating point instructions.

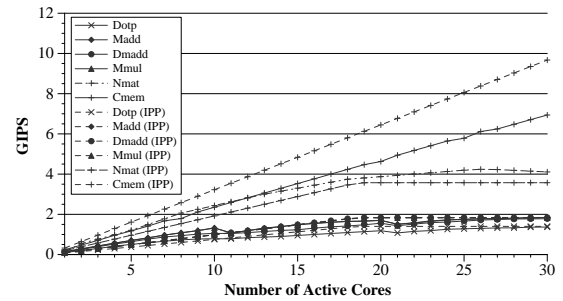


Figure 13: GIPS vs. Active Cores

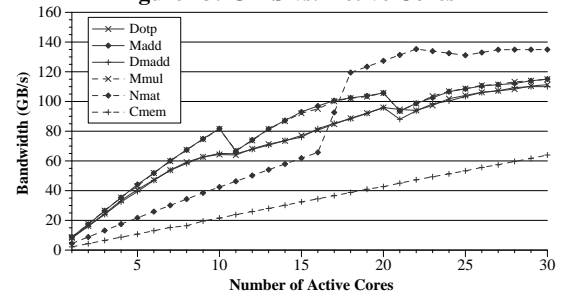


Figure 14: Average measured bandwidth consumption vs. # of active cores

its bandwidth consumption and the number of active cores, but it still cannot reach the peak memory bandwidth. The memory bandwidths of the remaining benchmarks are saturated when the number of active cores is around 19. This explains why the performance of these benchmarks is not improved significantly after approximately 19 active cores.

Figure 15 shows GIPS/W for the same experiment. The results show both the actual GIPS/W and the predicted GIPS/W using IPP. Nmat shows a salient peak point, but for the rest of benchmarks,

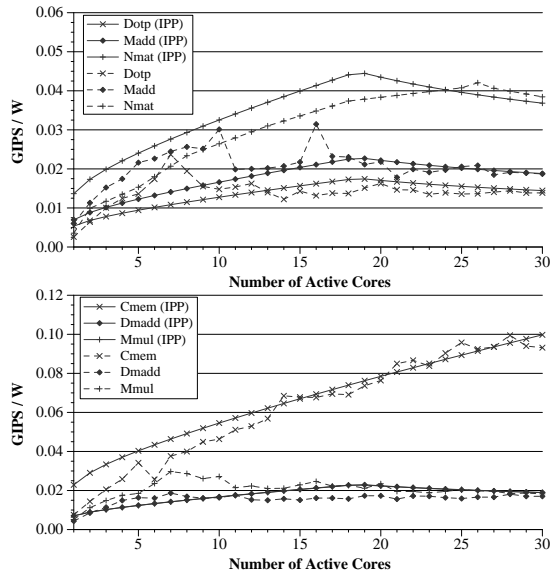


Figure 15: Performance per watt variation vs. # of active cores for measured and the predicted values

the efficiency (GIPS/W) has a very smooth curve. As we have expected, only GIPS/W of Cmem increases linearly in both the measured data and the predicted data.

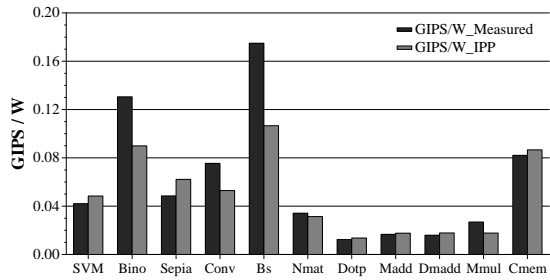


Figure 16: GIPS/W for the GPGPU kernels

Figure 16 shows GIPS/W for all the GPGPU kernels running on 30 active cores. The GIPS/W values of the non-bandwidth limited benchmarks are much higher than those of the bandwidth limited benchmarks. GIPS/W values can vary significantly from application to application depending on their performance. The results also include the predicted GIPS/W using IPP. Except for Bino and Bs, IPP predicts GIPS/W values fairly accurately. The errors in the predicted GIPS/W values of Bino and Bs are attributed to the differences between their predicted and measured runtime performance.

6.4.1 Energy Savings by Using the Optimal Number of Cores Based on IPP

Based on Equation (34), IPP calculates the optimal number of cores for a given application. This is a simple way of choosing the highest GIPS/W point among different number of cores. IPP returns 20 for all the evaluated memory bandwidth limited benchmarks and 30 for Cmem.

Figure 17 shows the difference in energy savings between the use of the optimal number of cores and the maximum number (30) of cores. *Runtime+Idle* shows the energy savings when the total GPU power is used in the calculation. *Runtime* shows the energy savings when only the runtime power from the equation (5) is used. *Powergating*

is the predicted energy savings if power gating is applied. The average energy savings for *Runtime* cases is 10.99%.

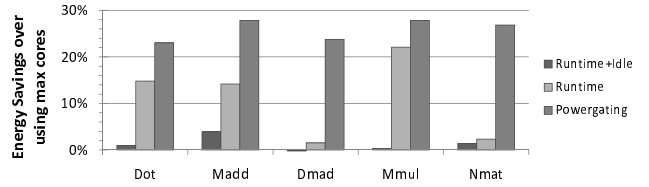


Figure 17: Energy savings using the optimal number of cores based on the IPP system (NVIDIA GTX 280 and power gating GPUs)

6.4.2 Energy Savings in Power Gating GPUs

The current NVIDIA GPUs do not employ any per-core power gating mechanism. However, future GPU architectures could employ power gating mechanisms as a result of the growth in the number of cores. As a concrete example, CPUs have already made use of per-core power gating [11].

To evaluate the energy savings in power gating processors, we predict the GPU power consumption as a linear function of the number of active cores. For example, if 30 SMs consume total 120W for an application, we assume that each core consumes 4W when per-core power gating is used. There is no reason to differentiate between Runtime+Idle and Runtime power since the power gating mechanism eliminates idle power consumption from in-active cores. Figure 17 shows the predicted amount of energy savings for the GPU cores that employ power gating. Since power consumption of each individual core is much smaller in a power-gating system, the amount of energy savings is much higher than in the current NVIDIA GTX280 processors. When power gating is applied, the average energy savings is 25.85%. Hence, utilizing only fewer cores based on the outcomes of IPP will be more beneficial in future per-core power-gating processors.

7. RELATED WORK

7.1 Power Modeling

Isci and Martonosi proposed power modeling using empirical data [12]. There have been follow-up studies that use similar techniques for other architectures [7]. Watch [5] has been widely used to model dynamic power consumption using event counters from architectural simulations. HotLeakage models leakage current and power based on circuit modeling and dynamic events [24]. Skadron et al. proposed temperature aware microarchitecture modeling [20] and also released a software, *HotSpot*. Both HotLeakage and HotSpot require architectural simulators to model dynamic power consumption. All these studies were done only for CPUs.

Sheaffer et al. studied a thermal management for GPUs [19]. In their work, the GPU was a fixed graphics hardware. Fu et al. presented experimental data of a GPU system and evaluated the efficiency of energy and power [8].

Our work is also based on empirical CPU power modeling. The biggest contribution of our GPU model over the previous CPU models is that we propose a GPU power model that does not require performance measurements. By integrating an analytical timing model and an empirical power model, we are able to predict the power consumption of GPGPU workloads with only the instruction mixture information. We also extend the GPU power model

to model increases in the leakage power consumption over time, which is becoming a critical component in many-core processors.

7.2 Using Fewer Number of Cores

Huang et al. evaluated the energy efficiency of GPUs for scientific computing [10]. Their work demonstrated the efficiency for only one benchmark and concluded that using all the cores provides the best efficiency. They did not consider any bandwidth limitation effects.

Li and Martinez studied power and performance considerations for CMPs [14]. They also analytically evaluated the optimal number of processors for best power/energy/EDP. However, their work was focused on CMP and presented heuristics to reduce design space search using power and performance models.

Suleman et al. proposed a feedback driven threading mechanism [22]. By monitoring the bandwidth consumption using a hardware counter, their feedback system decides how many threads (cores) can be run without degrading performance. Unlike our work, it requires runtime profiling to know the minimum number of threads to reach the peak bandwidth. Furthermore, they demonstrate power savings through simulation without a detailed power model. The IPP system predicts the number of cores that reaches the peak bandwidth at static time, thereby allowing the compiler or thread scheduler to use that information without any runtime profiling. Furthermore, we demonstrate the power savings by using both the detailed power model and the real system.

8. CONCLUSIONS

In this paper, we proposed an integrated power and performance modeling system (IPP) for the GPU architecture and the GPGPU kernels. IPP extends the empirical CPU modeling mechanism to model the GPU power and also considers the increases in leakage power consumption that resulted from the increases in temperature. Using the proposed power model and the newly-developed timing model, IPP predicts performance per watt and also the optimal number of cores to achieve energy savings.

The power model using IPP predicts the power consumption and the execution time with an average of 8.94% error for the evaluated GPGPU kernels. IPP predicts the performance per watt and the optimal number of cores for the five evaluated bandwidth limited GPGPU kernels. Based on IPP, the system can save on average 10.99% of runtime energy consumption for the bandwidth limited applications by using fewer cores. We demonstrated the power savings in the real machine. We also calculated the power savings if a per-core power gating mechanism is employed, and the result shows an average of 25.85% in energy reduction.

The proposed IPP system can be used by a thread scheduler (power management system) as we have discussed in the paper. It can be also used by compilers or programmers to optimize program configurations as we have demonstrated in the paper. In our future work, we will incorporate dynamic voltage and frequency control systems in the power and performance model.

Acknowledgments

Special thanks to Gregory Damos, Andrew Kerr, and Sudhakar Yalamanchili for Ocelot support. We thank the anonymous reviewers for their comments. We also thank Mike O'Connor, Alex Meritt, Tom Dewey, David Tarjan, Dilan Manatunga, Nagesh Lakshminarayana, Richard Vuduc, Chi-keung Luk, and HParch members for their feedback on improving the paper. We gratefully acknowledge the support of NSF CCF0903447, NSF/SRC task 1981, 2009

Georgia Tech Innovation Grant, Intel Corporation, Microsoft Research, and the equipment donations from NVIDIA.

9. REFERENCES

- [1] Exttech 380801. http://www.exttech.com/instrument/products/310_399/380801.html.
- [2] NVIDIA GeForce series GTX280, 8800GTX, 8800GT. <http://www.nvidia.com/geforce>.
- [3] Nvidia's next generation cuda compute architecture. <http://www.nvidia.com/fermi>.
- [4] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, 1999.
- [5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA-27*, 2000.
- [6] J. A. Butts and G. S. Sohi. A static power model for architects. *Microarchitecture*, 0:191–201, 2000.
- [7] G. Contreras and M. Martonosi. Power prediction for intel xscale processors using performance monitoring unit events. In *ISLPED*, 2005.
- [8] R. Fu, A. Zhai, P.-C. Yew, W.-C. Hsu, and J. Lu. Reducing queuing stalls caused by data prefetching. In *INTERACT-11*, 2007.
- [9] S. Hong and H. Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *ISCA*, 2009.
- [10] S. Huang, S. Xiao, and W. Feng. On the energy efficiency of graphics processing units for scientific computing. In *IPDPS*, 2009.
- [11] Intel. Intel®Nehalem Microarchitecture. <http://www.intel.com/technology/architecture-silicon/next-gen/>.
- [12] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*, 2003.
- [13] A. Kerr, G. Damos, and S. Yalamanchili. A characterization and analysis of ptx kernels. In *IISWC*, 2009.
- [14] J. Li and J. F. Martínez. Power-performance considerations of parallel computing on chip multiprocessors. *ACM Trans. Archit. Code Optim.*, 2(4):397–422, 2005.
- [15] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng. Merge: a programming model for heterogeneous multi-core systems. In *ASPLOS XIII*, 2008.
- [16] NVClock. Nvidia overclocking on Linux. <http://www.linuxhardware.org/nvclock/>.
- [17] NVIDIA Corporation. *CUDA Programming Guide, V3.0*.
- [18] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proceedings of the IEEE*, 91(2):305–327, Feb 2003.
- [19] J. W. Sheaffer, D. Luebke, and K. Skadron. A flexible simulation framework for graphics architectures. In *HWWS*, 2004.
- [20] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.*, 1(1):94–125, 2004.
- [21] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full chip leakage estimation considering power supply and temperature variations. In *ISLPED*, 2003.
- [22] M. A. Suleman, M. K. Qureshi, and Y. N. Patt. Feedback driven threading: Power-efficient and high-performance execution of multithreaded workloads on cmps. In *ASPLOS-XIII*, 2008.
- [23] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, 2009.
- [24] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical report, University of Virginia, 2003.