Tree Structured Analysis on GPU Power Study

Jianmin Chen[1] Bin Li[2] Ying Zhang[3] Lu Peng[3] Jih-kwon Peir[1]

[1]Department of CISE
University of Florida
Gainesville, FL
{jichen, peir}@cise.ufl.edu

[2]Department of Experimental Statistics
Louisiana State University
Baton Rouge, LA
bli@lsu.edu

[3]Department of ECE
Louisiana State University
Baton Rouge, LA
{yzhan29, lpeng}@lsu.edu

Abstract—**Graphics Processing Units (GPUs) have emerged as a promising platform for parallel computation. With a large number of processor cores and abundant memory bandwidth, GPUs deliver substantial computation power. While providing high computation performance, a GPU consumes high power and needs sufficient power supplies and cooling systems. It is essential to institute an efficient mechanism for evaluating and understanding the power consumption when running real applications on high-end GPUs.**

In this paper, we present a high-level GPU power consumption model using sophisticated tree-based random forest methods which correlate and predict the power consumption using a set of performance variables. We demonstrate that this statistical model not only predicts the GPU runtime power consumption more accurately than existing regression based approaches, but more importantly, it provides sufficient insights into understanding the correlation of the GPU power consumption with individual performance metrics. We use a GPU simulator that can collect more runtime performance metrics than hardware counters. We measure the power consumption of a wide-range of CUDA kernels on an experimental system with GTX 280 GPU to collect statistical samples for power analysis. The proposed method is applicable to other GPUs as well.

I. Introduction

Due to excessive power consumptions, limited instruction level parallelism, and escalating processor-memory walls, the computer industry has moved away from building expensive single processor chips with limited performance improvement to multi-core chip for higher chip-level IPCs (Instructions Per Cycle) with an acceptable power budget. Instead of replicating general-purpose CPUs (cores) in a single chip, the recent introduction of Nvidia's GPUs [17][26] take a different approach by building many-core GPU chip as co-processors to be connected through a PCI-Express bus to the host CPU. The host executes the source program and initiates computation kernels, each with multiple thread blocks to be executed on the GPU. In the GPU chip, multiple streaming processors (or SPs) are grouped into a few streaming multiprocessors (or SMs) as a scheduling unit. Based on the resource requirement, one or more thread blocks can be scheduled on an SM. Each thread block contains one or more 32-thread warps to be executed on multiple SPs in a Single-Instruction-Multiple-Threads (SIMT) fashion for achieving high floating-point operations per second.

Nvidia's Compute Unified Device Architecture (CUDA), a C++-like programming language [25], or OpenCL [15] is used to parallelize the applications. To supply the instruction and data, Nvidia's GPU has a sophisticated memory system. With a limited silicon budget, the on-chip fast storage in each SM is small and relies on programmers to efficiently use it for hiding the memory access latency.

While delivering high computation performance, the GPU consumes high power and thus needs to be equipped with sufficient power supplies and cooling systems. For example, the Tesla 20-series GPU based on the new Fermi architecture [22] consists of 512 SPs grouped into 32 SMs. The Tesla GPU is optimized for general-purpose scientific applications to deliver 500+ gigaflops of IEEE standard double-precision floating point operations and over one teraflop of single-precision peak performance. However, it also consumes 200-300W of power [23]. The high-end Appro 1426G4 server [1] features four Tesla 20-series GPUs, two Quad or Six-Core Intel Xeon (5600) Processors with 96GB DDR memory and requires a 1400W high-efficiency power supply and cooling fans. Therefore, it is essential to institute an efficient mechanism for evaluating and understanding the power consumption and dissipation requirements when running real applications in these high-end GPUs.

Existing GPU power measurement tools are very limited. Since the current GPUs are not equipped with power sensors to directly assess the power dissipation at runtime, the GPU power is either observed from a separate power meter or derived from the probed voltage and current on the GPU. Such approaches require additional hardware devices and are not a practical solution. On the other hand, software approaches to estimate the power consumption are still in the early stages. Unlike CPU power, which can be modeled at different levels by simulation tools such as Wattch [5] and Simple power [35], the GPU community lacks an accurate and flexible tool for power simulation. Although some preliminary attempts have been made, the existing tools are limited to a specific architecture or component and more importantly, they cannot provide much insight about GPU power consumption due to limited analyzing ability and metrics they can collect. For example, Qsilver [33], an OpenGL based performance simulator, was extended to include a power model for the OpenGL rendering pipeline. PowerRed [29] includes both analytical and empirical power model along with an area-aware interconnect model to estimate power at the architectural level.

In this paper, we present high-level GPU power consumption models using a linear regression tree [4] and sophisticated random forest methods [3]. The random forest method provides the correlation analysis between the power consumption and the independent performance variables so we can better understand GPU's power characteristics. These models are established based on the execution frequencies of instructions and memory accesses in a wide-variety of application samples. To complement the execution frequency profile, we include several performance-sensitive architectural metrics into the analysis for accurate power modeling. Our methodology consists of the following steps.

Workload profiling and power measurement: We collect 52 application kernels from popular GPU benchmark suites [6][24][28] and from a few real applications [7]. These applications are parallelized using CUDA and run on an experimental system with GTX 280. The energy consumptions of these kernels are measured using a YOKOGAWA WT210 Digital Power Meter [36] as statistical power samples for power consumption prediction and analysis.

Collecting performance metrics from application kernels: A GPU simulator *GPGPUSim* [2] for simulating application execution is used to measure the executed instructions and memory accesses as well as other important architectural parameters. Unlike the GPU profiler [24], *GPGPUSim* is flexible and can be modified to collect execution statistics. In this study, we modify *GPGPUSim* to collect 18 execution related and 4 architecture related metrics and use them as the independent variables for power analysis.

Statistical analysis from the samples: Based on 52 sample application kernels, we first use multiple linear regression analysis to build the power equation with a set of independent variables. We then apply the sophisticated random-forest method [3] to model the power with high accuracy. The random forest approach can also shed insights on the correlation between power and individual variables.

Power correlation, prediction and verification: The random forest model identifies most influential variables in power prediction including register access (Register), single-precision floating-point instruction (S.FP) and global memory access (GMInst). It also shows that several performance-sensitive architecture metrics indeed play a role in modeling the power consumption. In addition, a random forest model can output a proximity plot, which provides an indication of which observations (kernels) are effectively close together in the eyes of the random forest model Furthermore, we verify the accuracy of the model using leave-one-out cross validation (LOOCV). The average percentage error (PE) is 7.77% in comparison with the measured power consumption for the random forest method. The average PEs increase to 11.68% and 11.70% respectively for using the regression tree and the multiple linear regression approaches.

This paper makes two key contributions. First, we present a powerful statistical model to analyze and predict GPU power consumption. Our model uses the sophisticated random forest method with high accuracy and insightful analysis to understand the correlation of power consumption and other independent performance variables. To the best of our knowledge, this is the first attempt to apply such sophisticated statistical methods for GPU power analysis. Second, we modify *GPGPUSim*, a GPU simulator to collect workload execution characteristics for the power analysis. It can collect detailed information for each type of instruction as well as architecture related metrics. Comparing with previous works, the simulator lifts many restrictions in terms of the type of variables that can be collected and the type of applications that can be modeled, which is very important for providing a complete study of GPU power characteristics.

The outline of the paper is as follows. Section II provides a general description of CUDA GPU architecture. Section III describes the collection of the kernel execution characteristics using *GPGPUSim*. This is followed by the benchmark selection and power measurement in Section IV. Statistical modeling using random forest and other methods is given in Section V. Prediction of power consumption, analysis and verification are in Section VI. Section VII presents the correlation analysis of the power consumption with other independent variables. Section VIII describes related work. Finally, Section IX concludes this work and gives future directions.

II. NVIDIA GPU ARCHITECTURES

GPU power consumption is closely related to the runtime characteristics during the kernel execution on GPUs. In this section, we highlight the important many-core and memory organization of Nvidia's GPUs that impact the program execution and the power consumption.

Nvidia's GPUs contain a number of Streaming Processors (SPs) grouped into a set of Streaming Multiprocessors (SMs). The GTX 280 GPU which was used for the experiment has 30 SMs, each with 8 SPs, for a total of 240 processors. CUDA extends ANSI C with a few new program constructs and key words [25]. The programmers can use these CUDA extensions to parallelize the programs into three levels. In the highest level, a *kernel* can be invoked from the host CPU to create a single *grid* to run on the GPU. Parallel kernel invocations are allowed on the same or multiple GPUs [13][26]. Each grid has multiple *thread blocks* which can be specified as a three-dimensional array. Each thread block consists of multiple *threads* also specified in a three-dimension format. A thread block is scheduled to run on one SM independently with other thread blocks. Within each thread block, parallel threads are further grouped into 32-thread *warps*. A warp is the smallest unit to be scheduled for execution on SPs. Upon execution, a half warp is assigned to be executed in all the SPs in the SM. Multiple warps in the same or different thread blocks can be scheduled on the same SMs simultaneously limited only by the hardware resource available in each SM. Active warps in each SM are assigned to the functional units in SPs when the warp is ready to execute and the hardware is available. There is no penalty in switching executions between different warps.

The CUDA GPU has multiple levels of memory hierarchies. The *global* memory (also called *device* memory) located off-chip with long access latency can be accessed by all thread blocks in a grid. The on-chip *shared* memory with fast access latency is accessible from all threads within a thread block. An efficient usage of the fast but limited shared memory is critical in reducing expensive global memory accesses. Each SM has on-chip registers to be used by all thread blocks. The off-chip *local* memory can only be accessed by a single thread. Due to its long latency, the local memory is used for certain functions, e.g. register spilling. The CUDA GPU also has *constant* and *texture* memories for the read-only data. Although located off-chip, the data in these two memories can be cached for fast accesses. When a kernel is invoked from the host CPU, the needed data must be copied from the host main memory to the device's global memory before the start of executions.

Due to the resource constraint, there are limited numbers of thread blocks and threads that can be scheduled on each SM in

GTX280 as summarized in Table 1. Such constraints influence the execution efficiency and the power consumption.

Resource	Limitation
Threads per SM	1024
Thread blocks per SM	8
Threads per warp	32
Warps per SM	32
Threads per thread block	512
Registers (32bit) per SM	16384
Shared Memory (bytes) per SM	16KB

Table 1: Resource constrains of GTX280

III. CHARACTERISTICS OF KERNEL EXECUTION AND PROFILING

In this section, we present the runtime characteristics that we analyzed and describe the methods for profiling them. In general, the total execution frequency of different types of instructions and memory accesses largely determines the GPU power consumption. However, the execution efficiency also influences the power assumption. In this study, we include four performance-sensitive architectural parameters.

In order to collect the kernel execution characteristics, we use a GPU simulator *GPGPUSim*, which is a cycle-level simulator that simulates the PTX instruction set running on Nvidia GPUs. The PTX resembles the real ISA of Nvidia GPU and has demonstrated reasonably close IPC results [21]. *GPGPUSim* consists of a function simulator (*cuda-sim*) that executes PTX kernels, a performance simulator (*gpgpu-sim*) that simulates the timing behavior of a GPU, and an interconnection network simulator (*intersim*) that models the interconnect delays. After the *cudafe* separates GPU and CPU code, the *nvcc* compiles and produces the PTX assembly code for GPU. Then, the *ptxas* assembler generates thread-level register, block-level shared memory, and block-level constant memory usage information to determine how many blocks can be assigned to each SM. *GPGPUSim* also implements a custom CUDA runtime library to divert CUDA API calls to *GPGPUSim*. During the simulation, the simulated kernel C code runs natively on the CPU of the simulated machine.

Table 2 summarizes the 22 runtime characteristics including 18 execution frequencies and 4 architecture parameters we collected from *GPGPUSim* for power modeling and analysis. We separate accesses to register, shared memory, local memory, global memory, texture memory and constant memory for more accurate modeling. Since the texture and constant memory accesses are cacheable, we further separate them into hits and misses to reflect their power consumption.

We consider four performance-sensitive parameters to complement the execution frequency profile. First, individual global memory accesses with certain regularity from all threads in a half-warp can be coalesced into one or fewer memory moves for saving the bandwidth requirement. Due to the limitation of PTX simulator, we cannot separate different granularities of global memory accesses. In establishing the power model, the *un-coalesced* global memory accesses are used to complement the total global memory accesses. Second, in the SIMT mode, a single instruction must be executed by all threads in a half-warp. *Divergent* branches will result in executing the true path and the false path separately, hence can

Notation	Description
ShMInst	Total shared memory instructions
PMInst	Total parameter memory instructions
LMInst	Total local memory instructions
GMInst	Total global memory instructions
TMInst.hit	Total texture cache hit
TMInst.miss	Total texture cache miss
CMInst.hit	Total constant cache hit
CMInst.miss	Total constant cache miss
Register	Total register accesses
S.FP	Total single-precision FP instructions
D.FP	Total double-precision FP instructions
INT	Total integer instructions
ALU	Total arithmetic logic unit instructions
SFU	Total special function unit instructions
Atomic	Total atomic instructions
Barrier	Total barrier instructions
M.Barrier	Total memory barrier instructions
Branch	Total branch instructions
UncoalesMem	Total global memory accesses that cannot be coalesced
D.Branch	Total divergent branches
BankConf	Total bank conflicts in accessing caches and shared memory
Occupancy	Ratio of active warps to maximum warps can be supported in a SM of GPU

Table 2: Runtime characteristics from GPGPUSim

be used as a complement to the total number of branches. Third, on-chip storage such as shared memory, caches, and registers are constructed with 16 banks to improve the bandwidth. *Bank access conflicts* cause serialization of thread execution in a half-warp. These bank conflicts impact the power consumption for instructions accessing the shared memory, caches, and registers. Fourth, Nvidia's GPUs and CUDA present constraints in creating and scheduling parallel threads on multiple SMs. It relies on the programmers to optimize the CUDA code for fully populating all SPs in a SM with simultaneous thread blocks and threads to achieve the best utilization. Although the optimization space of CUDA/GPU is multi-dimensional and non-linear [31], the *SM occupancy* is an important metric reflecting the scheduling and execution of thread blocks and is considered in the power analysis.

It is important to note that due to the limited sample size we collected, it may encounter the over-fitting problem in constructing the power model with large number of independent variables. As a result, the prediction accuracy may be sacrificed. Nevertheless, the goal of this paper is not only to build an accurate model for power prediction, but more importantly, to analyze and understand the correlation between power consumption and different runtime characteristics.

IV. SAMPLE WORKLOADS AND POWER MEASUREMENT

To cover a wide variety of GPU application kernels, we selected kernels from *CUDA SDK* [24] with many popular applications. We also use kernels from *Rodinia* Benchmark [6] which covers a wide range of applications in parallel communication patterns, synchronization techniques, and power consumption studies, and from *Parboil* Benchmark [28] which is comprised of various applications developed to be better suited for measuring GPU performance. We also collect kernels from a computer graphics application *Poisson Image Editing* [7] which seamlessly integrates new image contents into a given background image, and a computer vision application *3D Shape from Shading* [7] which deals with recovering the 3-D structure of an object from its image. Due to the constraints in the simulator with unsupported instructions, we cannot run a few kernels from these benchmark suits. In addition, we adjust a few small kernels (<0.5ms) to a bigger problem size. In case that the cost of kernel invocation and host return dominate the execution time, these small kernels are not included. Lastly, we exclude kernel *besect_large* in *eigenvalues* which has only 1 thread block. Table 3 lists all the 52 kernels we used.

We use YOKOGAWA WT210 Digital Power Meter to measure the overall system energy consumption. Each kernel execution is repeated multiple times for more accurate measurement. The power consumption P can be calculated from the measured energy E such that $P = E/T$, where T is the running time. However, the measured power includes both the GPU power and the system power of CPU and other components on the mother board. To isolate the system power, we remove the GTX280 card and measure the idle system energy repeatedly. The average measured system power without GPU is about 113W which is about 35-40% of the overall measured power. This constant system power can then be subtracted from the measured power of each benchmark to get the correct GPU power for the kernel samples.

V. STATISTICAL MODELING

Tree-based models provide an alternative to the classic linear regression models [4]. The tree models are fitted through a recursive partitioning algorithm whereby a dataset is successively split into increasingly homogenous subsets until the information gained by additional splits is not out-weighed by the additional complexity due to the tree's growth. Tree structured models are adept at capturing non-linear and non-additive behavior, e.g. interactions among independent variables are routinely and automatically handled. Random forest, proposed by Leo Breiman [3], is an ensemble learning algorithm that combines many individual trees in the following way:

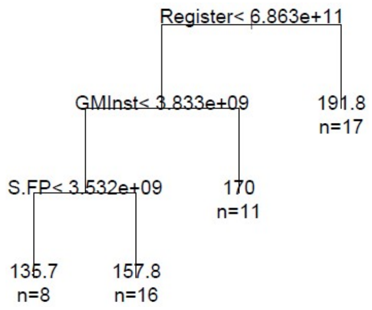

Figure 1: Regression tree for 52 kernels based on the parameters from *GPGPUSim*.

For each tree, (1) a bootstrap sample is drawn from the original sample; (2) a tree is grown using the bootstrap sample of the data generated in the step (1), and at each split the candidate set of variables is a random subset of all the variables.

Benchmark	Kernels
3DFD	1.stencil_3D_16x16_order8
binomialOptions	2.binomialOptionsKernel
BlackScholes	3.BlackScholesGPU
convolutionSeparable	4.convolutionColumnsKernel 5.convolutionRowsKernel
convolutionTexture	6.convolutionColumnsKernel 7.convolutionRowsKernel
dct8x8	8.CUDAkernel1DCT 9.CUDAkernel1IDCT
dxtc	10.compress
fastWalshTransform	11.fwtBatch1Kernel 12.fwtBatch2Kernel
Histogram	13.histogram64Kernel

matrixMul	14.matrixMul(single)
	15.matrixMul(Double)

MersenneTwister	16.RandomGPU 17.BoxMullerGPU
quasirandomGenerator	18.quasirandomGeneratorKernel
scalarProd	19.scalarProdGPU
scan	20.scan_best 21.scan_naive 22.scan_workefficient
simpleAtomicIntrinsics	23.testKernel
sobolQRNG	24.sobolGPU_kernel
Transpose	25.transpose 26.transpose_naïve
sad	27.mb_sad_calc
mri-q	28.ComputeQ_GPU
lbm	29.performStreamCollide_kernel
backprop	30.bpnn_layerforward CUDA 31.bpnn_adjust_weights_cuda
Heartwall	32.kernel
hotspot	33.calculate_temp
Particlefilter	34.kernel
lud	35.lud_internal
nw	36.needle_cuda_shared_1 37.needle_cuda_shared_2
histogram	38.histogram256Kernel
MonteCarlo	39.MonteCarloOneBlockOption
3D-shape	40.SI-Base 41.SI-Coarse 42.SI-UD-comm-square 43.SI-4w-comm-square 44.SI-4w-comm-combine-square
Poisson-image	45.GS-Base 46.GS-Coarse 47.GS-UD-comm-square 48.GS-4w-comm-square 49.GS-4w-comm-combine-square
Kmeans	50.kmeansPoint 51.invert_mapping
particlefiter	52.likelihood_kernel

Table 3: List of kernels

The response variable is predicted using the average of the predictions of all the trees in the forest.

Figure 1 shows a single regression tree with four leaves (terminal nodes) from analyzing the 52 benchmark kernels.

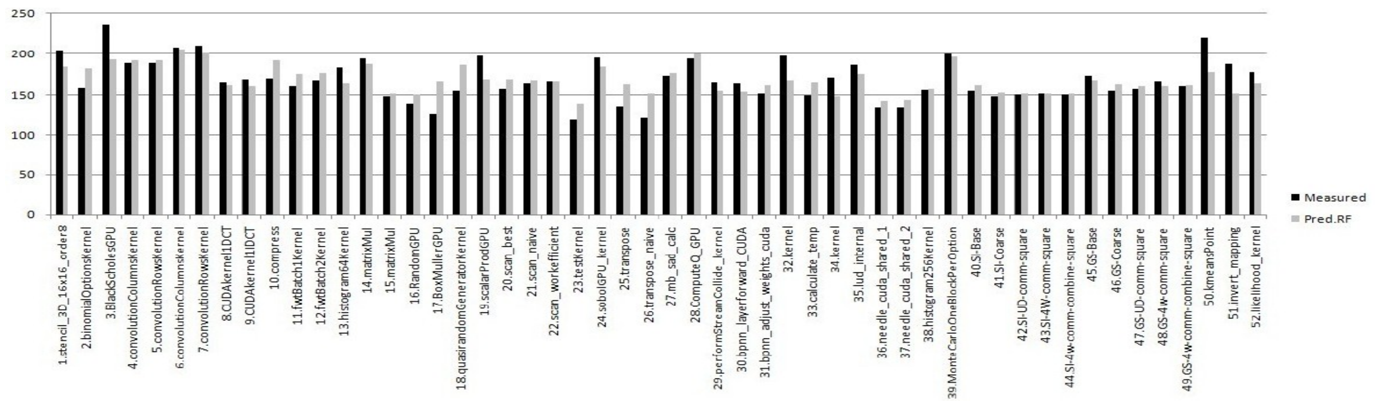

Figure 2: Comparison of predicted and measured power

For example, the leftmost leaf, which includes 8 kernels, indicates the average GPU power consumption for the kernels with "Register<6.863e+11" (register access less than 6.863e+11), "GMInst < 3.833e+9" (global memory instructions less than 3.833e+9) and "S.FP < 3.532e+9" (single precision floating point instructions less than 3.532e+9) is 135.7 watts. Studies have shown that using random forest ensemble usually substantially improves the prediction performance from a single tree.

Although a random forest model usually consists of hundreds of trees, it does not have to be treated like a black box. A random forest model can be summarized, interpreted and visualized similarly to conventional regression models. This includes identifying parameters that are most influential in contributing to the response's variation, and visualizing the nature of dependence of the fitted model on these important parameters. The random forest model provides a relative variable importance measure which is based on the number of times a variable is selected for splitting, weighted by the squared improvement to the model as a result of each split, and then average over all trees. The relative influence is scaled so that the sum adds to 100, with a higher number indicating a stronger influence. The ranking of relative influence of power consumption allows us to gain insight on which runtime characteristics has the most impact to the power consumption.

Another useful interpretation tool is *partial dependence* plot, which shows the effect of a subset of variables on the response after accounting for the average effects of all other variables in the model. Hence, it helps us to visualize the trend of impact of a subset of variable on the response in a low dimensional plot. In addition, a random forest model can also output a proximity plot, which provides an indication of which observations (kernels) are effectively close together in the eyes of the random forest model through a two-dimensional plot. Such information helps us understanding the similarity and difference with respect to power consumption among the sample application kernels. In this study, we fixed the number of trees in random forest to be 500. The experiments on the random forest and regression tree are done by using the **randomForest** and **rpart** packages in **R**, respectively [11], a free software environment for statistical computing.

VI. POWER PREDICTION AND VERIFICATION

The prediction accuracy is evaluated through leave-one-out cross-validation (LOOCV). LOOCV is a repeated procedure which uses a single kernel from the 52 kernels as the validation data, and the remaining kernels as the training data. Table 4 summarizes the prediction accuracy of three competing methods (random forest, regression tree and linear regression) using the LOOCV. We see that random forest is superior with an average percentage error (PE) 7.77%. In comparison, the average PEs for regression tree and linear regression approaches are 11.68% and 11.70% respectively.

	Random Forest	Regression Tree	Linear Regression
PE	7.77%	11.68%	11.70%
MSE	302.6	637.8	2548.0

Table 4: Average percentage error (PE) and Mean squared error (MSE) of the three methods

Comparisons between the estimated and the measured power of individual application kernels using Random Forest method are given in figure 2. A majority of the kernels show high accuracy in their power prediction. However, there are also a few kernels with low accuracy. For example, kernel 3.BlackScholes's predicted power is much lower than the measured power. This kernel not only has high frequencies on regular computing instructions like S.FP and ALU, it also has high frequency of SFU instructions. The SFU instruction is executed on a separate special function units other than using the SPs. As a result, both hardware units can be active simultaneously for a long period of time and consume higher power. However, since such high frequency of SFU usage together with high frequency of regular computing instructions is rarely presented in other kernels, the power model underestimates the power in this special case. Another example is kernel 17.BoxMullerGPU in MersenneTwister. This kernel has 32 thread blocks running on 30 SMs. So 2 SMs execute 2 blocks while the rest only execute 1 block. In this case, there are just 2 SMs working for about half of the time, resulting in low power consumption. No other kernel has this scenario. Therefore, the model predicted much higher power consumption. In both cases, increasing the sample size should ease the inaccuracy.

VII. CORRELATION ANALYSIS

In this section, we show the correlation analysis between the dependent power consumption and the independent variables. We also show the proximity among the sample kernels. Figure 3 shows the relative importance for both runtime performance variables and architecture related parameters. Among all the input variables, we see that Register, S.FP, GMInst, ALU and INT are the five most influential variables. Obviously, register usage is influential since it is likely to be used in most instructions. Single-precision floating-point instructions produce high power since they represent a significant portion of instructions in floating-point kernels. Global memory moves consume high powers and are often necessary due to limited on-chip storage. ALU and Integer instructions are also ranked high since they are used for manipulating indices, decision-making instructions, and temporary variables.

It is interesting to see that *uncoalesced* global memory (UncoalesMem) is ranked sixth in relative importance. Given the fact that *GPGPUSim* does not handle multiple granularities of physical global memory moves, UncoalesMem represents the amount of GMInst that is uncoalesced, hence causes more physical global memory moves. The other three architecture metrics, D.Branch, BankConf, and Occupancy also play a role, but are not as significant as UncoalesMem.

Figure 4 shows the partial dependence plots for the four most influential operations (Register, S.FP, GMInst, and ALU) and four architecture related parameters. In each plot, all other metrics remain the average value. Note that in these plots, the Y-axis is scaled from 160 to 180 watts for the four most influential variables while Y-axis is from 165 to 172 watts for the architecture parameters to accommodate different range of influence. From the partial dependence plots for the four most influential variables, we can see that the increase of power generally goes along with the variable ranking. Due to the small sample size the increment slope is not very smooth. Among the variables, we observe very little changes when the total number of ALU instructions reaches 1.6E+11. This is due to the lack of samples beyond this amount of ALU instructions.

For the architecture related parameters, we see that UncoalesMem shows a clear correlation that more power is consumed with more uncoalesced global memory accesses, which require more physical moves. D.Branch and BankConf are seemly on the opposite direction such that with higher volume, less power consumption is produced. This is generally true since divergent branch and bank conflict slow down the execution, cause inefficient use of the resources and result in lower power consumption. However, the overall power impact is more complicated. Due to small sample size, the trend is not very stable. Occupancy, on the other hand, shows more consistent results such that given higher occupancy, the more power consumption is observed. Given the fact that the execution rate is generally faster by providing more overlapping thread blocks to hide memory or computation latency, high occupancy consumes more power. But, the power drops a little with highest occupancy because there might be more than enough active warps in each SM to fill the pipeline, which may result in contention for on chip resources and slow down the execution [2].

Figure 5 shows the proximity plot which can help us filtering out kernels with similar power behavior to select a set of benchmarking kernels for future power studies. This plot gives an indication of which observations are effectively close together in the eyes of the random forest. Since an individual tree is unpruned, the terminal nodes (also called leaf) will contain only a small number of instances (i.e. kernels).

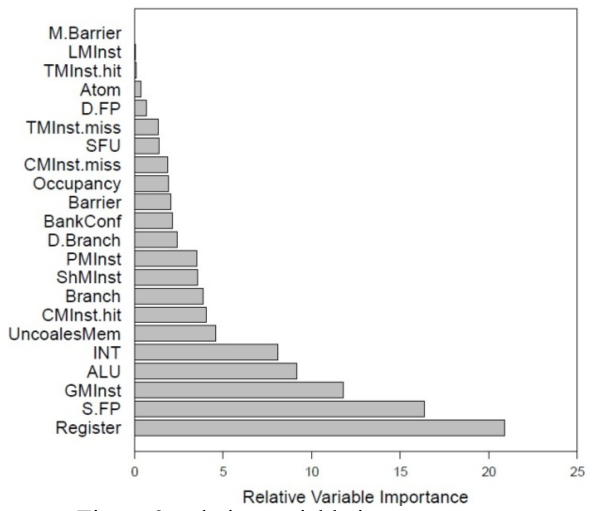

Figure 3: relative variable importance

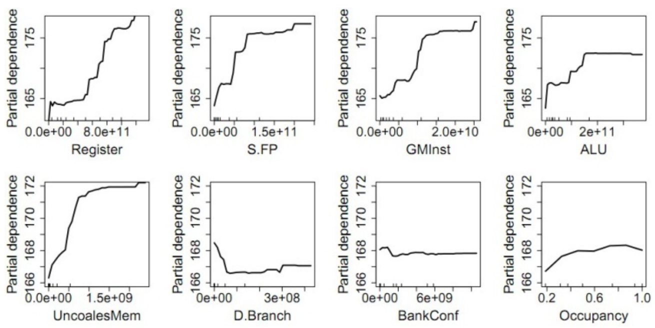

Figure 4: Partial dependence plots for four most influential and four architecture related parameters. The rug plots on inside bottom of plots show distribution of kernels across that variable, in deciles.

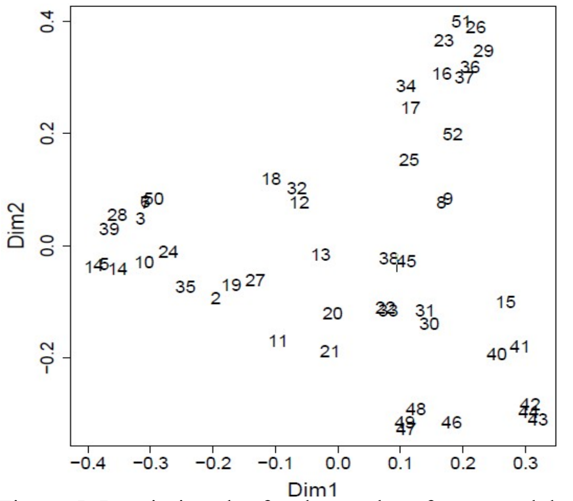

Figure 5: Proximity plot for the random forest model

If kernel i and kernel j both land in the same terminal node, increase the proximity between i and j by one. At the end of the run, the proximities are divided by the number of trees and proximity between a case and itself set equal to one. Then the distance is calculated by one minus the proximity value. After that, the multidimensional scaling [10] is applied to the distance matrix. The key idea of the proximity plot is to

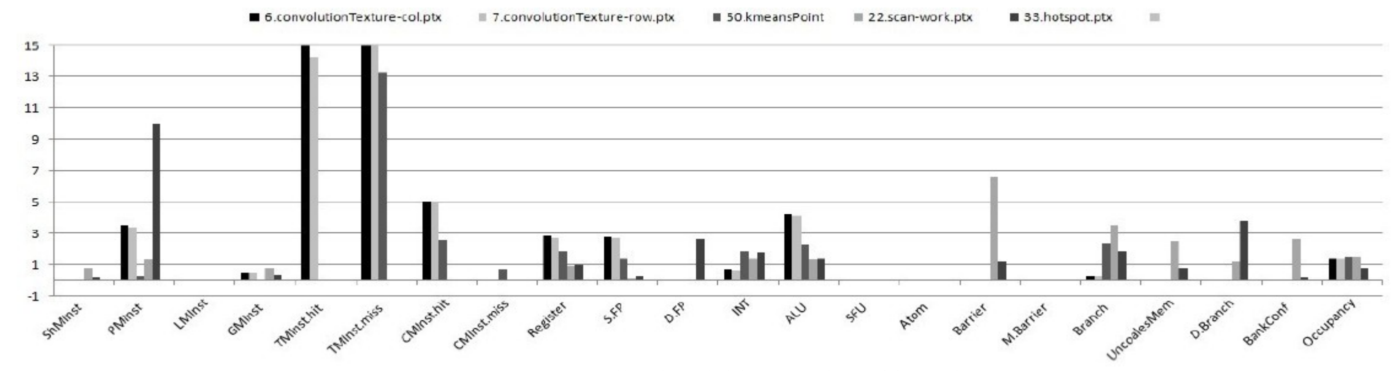

Figure 6: Normalized variables of selected kernels

approximate the original set of distances with distances corresponding to a configuration of points in a low-dimensional Euclidean space, so that the distances between the items in the low-dimensional space (e.g. 2D space in Figure 5) will be as close to the original distances as possible.

In Figure 5, we can see that some kernels are very close to each other. For example, there are two sets of kernels; each consists of kernels with high similarity. The first set has kernels 6.convolutionColumnsKernel, 7.convolutionRowsKernel and 50.kmeansPoint while the second set has kernels 22.scanworkefficient and 33.hotspot. Figure 6 shows the value for the 5 kernel's metrics normalized to the average value of all the kernels. The first set of kernels have above average number of register, S.FP, GMInst and ALU which are the most important 4 variables in our model and much higher than average number of texture cache misses and constant cache hits. The second set of kernels has close to average amount of register and ALU. They also have similar amount of GMinst and almost no S.FP instructions. Their power consumptions also follow this similarity. The first set has high measured power consumption of 207.3, 210.3 and 220.3 and the second set has 165.6 and 149.1 respectively.

VIII. RELATED WORK

Running large-scale parallel applications on modern GPUs has been widely adopted recently. Although there have been many reports for the performance analysis on GPUs [18][32][34][37], the study of the important GPU power consumption issues is very limited. Hong and Kim [13] presented an empirical model which accurately predicts GPU run time power from activities of individual components including floating point unit, register file, ALU, etc. They rely on an early performance model [12] to estimate the execution behavior, instead of predicting the power using statistical samples. Due to the limitation of their model, applications with intensive control flows, asymmetric execution among thread blocks, and intensive use of texture and constant caches cannot be modeled accurately. Ma et al. [19] presented an SVR regression model to predict the dynamic power consumption of a GPU with the counters from *perfkit*. Since *perfkit* is developed to debug and profile OpenGL and Direct3D applications, their model is more suitable for graphics applications. Nagasaka et. al [20] proposed a linear regression based statistical model to predict GPU power using performance counters collected from the runtime profiler [24]. With GTX 285 GPU and CUDA version 2.3, their model is able to predict the dynamic power reasonably accurate. A recent work-in-progress [8] is the first using the random forest method to model GPU power consumption with limited correlation analysis.

Our approach extends the model using the random forest method and is superior in two aspects. First, we show that the random forest model for predicting GPU power consumption is more accurate than the approaches using linear regression and regression tree. Second, with more performance variables collected from *GPGPUSim*, we use the random forest model to rank the importance of individual variables and their partial dependences to the power consumption. This correlation analysis helps computer architects for better GPU designs. It also helps compiler developers for better use of energy/power efficient GPUs.

Efforts have been made to design frameworks for the GPU power simulation. For example, Ramani et al [29] introduce the PowerRed tool, which is an attempt to model the GPU power at the architectural level. There are also few works concentrating on the tradeoff between performance and energy consumption of GPUs. In [30], Rofouei et al. use a novel platform to collect runtime energy dissipation of a computing system when running different applications. They demonstrate that a GPU is more energy efficient compared to a CPU when the performance reaches above certain bound. Huang et al. [14] conduct a similar study to investigate the energy efficiency on a hybrid CPU+GPU environment. Ren et al. [27] consider different implementations of matrix multiplication kernels and run them on different devices (i.e., CPU, CPU+GPU, CPU+GPUs) to compare the respective performance and energy consumptions. They show that when the CPU is given an appropriate share of workload, the best energy efficiency can be delivered.

IX. CONCLUSION AND FUTURE WORK

Power consumptions in high-end GPUs require expensive power supplies and cooling systems. We present in this paper a powerful GPU power modeling method to understand and predict the power consumption while running real application kernels. A sophisticated ensemble learning algorithm based on random forest analysis that combines many individual regression trees is used to establish the GPU power consumption model. Comparing with the linear regression or the regression tree approaches, our results show that the random forest model is not only more accurate in power prediction using a set of runtime performance metrics, it also provides sufficient insights for its users to understand the correlations between the GPU runtime power consumption and the individual performance metrics.

This new GPU power modeling method provides many new fronts for further research. For example, in conjunction with a GPU performance model, we can investigate how to balance the power consumptions during kernel executions without sacrificing the overall performance. Given the relative influence ranking of individual instructions, we can also work on the low power circuits to cut down the power consumption for the top ranked instructions. Furthermore, we can extend this work to look into the total energy consumption requirement and optimization for running the kernels. Making changes just to reduce the power consumption may end up with higher total energy consumption due to slower execution.

References

[1] Appro. Appro 1u tetra gpu server - 1426g4. 2010. http://www.appro.com/product/1426G4server_overview.asp

[2] A. Bakhoda, G. Yuan, W. Fung, H. Wong, and T. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *ISPASS 2009*.

[3] L. Breiman. Random forests. In *Machine Learning, 45*, pages 5–32, 2001.

[4] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classi_cation and Regression Trees*. Chapman and Hall/CRC, 1 edition, January 1984.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28:83–94.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC 2009.*

[7] J. Chen, Z. Huang, F. Su, J.-K. Peir, J. Ho, and L. Peng. Weak execution ordering – exploiting iterative methods on many-core gpus. In *ISPASS 2010.*

[8] J. Chen, B. Li, Y. Zhang, L. Peng and J.-K. Peir. Statistical GPU Power Analysis Using Tree-based Methods. In IGCC'11 Work-in-Progress (WiP) workshop and Proceedings

[9] S. Collange, D. Defour, and A. Tisserand. Power consumption of gpus from a software perspective. In *ICCS* 2009.

[10] T.F. Cox and M.A.A. Cox (1994) Multidimensional Scaling. Chapman and Hall.

[11] R. Gentleman and R. Ihaka. The comprehensive r archive network. 2007.

[12] S. Hong and H. Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *ISCA* 2009.

[13] S. Hong and H. Kim. An integrated gpu power and performance model. In *ISCA* 2010.

[14] S. Huang, S. Xiao, and W. Feng. On the energy efficiency of graphics processing units for scientific computing. In *IPDPS 2009*.

[15] Khronos. OpenCL - The open standard for parallel programming of heterogeneous systems http://www.khronos.org/opencl/

[16] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS-2006*

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

[18] W. Liu and M. W. Schmidt. Performance predictions for general-purpose computation on GPUs. In ICPP 2007.

[19] X. Ma, M. Dong, L. Zhong, and Z. Deng. Statistical power consumption analysis and modeling for gpu-based computing. *Workshop on Power-Aware Computing and Systems*, 2009.

[20] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka. Statistical power modeling of gpu kernels using performance counters. In *Green Computing*, 2010.

[21] Nvidia. Tutorial on gpgpu-sim: A performance simulator for massively multithreaded processor research. 2009.

[22] Nvidia. Whitepaper: Nvidia's next generation cudatm compute architecture: Fermi. 2009.

[23] Nvidia. Tesla c2050 and tesla c2070 computing processor board. 2010.

[24] Nvidia. Cuda sdk 2.3. 2010

[25] Nvidia. Cuda: What is cuda? 2011.

[26] Nvidia. Nvidia geforce series gtx400, gtx200. 2011.

[27] D. Q. Ren and R. Suda. Investigation on the power efficiency of multi-core and gpu processing element in large scale simd computation with cuda. In *Green Computing*, 2010.

[28] I. research group. Parboil benchmark suite. 2011.

[29] K. Ramani, A. Ibrahim and D. Shimizu. Powerred: A flexible power modeling framework for power efficiency exploration in gpus. In *Worskshop on GPGPU*, 2007.

[30] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh. Energy-aware high performance computing with graphic processing units. In *HotPower* 2008.

[31] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *PPoPP 2008.*

[32] S. Ryoo, C. I. Rodrigues, S. S. Stone, S. S. Baghsorkhi, S.-Z. Ueng, J. A. Stratton, and W.-m. W. Hwu. Program optimization space pruning for a multithreaded gpu. In *CGO* 2008.

[33] J. Sheaffer, K. Skadron, and D. Luebke. Studying thermal management for graphics-processor architectures. In *ISPASS 2005.*

[34] V. Volkov and J. W. Demmel. Benchmarking gpus to tune dense linear algebra. In *Supercomputing* 2008.

[35] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *DAC* 2000.

[36] YOKOGAVA. WT210: http://tmi.yokogawa.com/us/

[37] Y. Zhuo, X.-L. Wu, J. Haldar, W. mei Hwu, Z. pei Liang, and B. Sutton. Accelerating iterative field-compensated mr image reconstruction on gpus. In *IEEE International Symposium on Biomedical Imaging: From Nano to Macro, 2010.*