2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)

An Approach to Estimate Power Consumption of a CUDA Kernel

Gargi Alavani
Department of CS & IS
BITS Pilani K. K. Birla Goa Campus
Goa, India
p20160008@goa.bits-pilani.ac.in

Jineet Desai
Department of CS & IS
BITS Pilani K. K. Birla Goa Campus
Goa, India
f20170168@goa.bits-pilani.ac.in

Santonu Sarkar
Department of CS & IS
BITS Pilani K. K. Birla Goa Campus
Goa, India
santonus@acm.org

***Abstract*—Graphics Processing Unit (GPU) has emerged as a popular computing device to achieve Exa-scale performance in High-Performance Computing (HPC) applications. While the power-performance ratio is relatively high for a GPU, it still draws a significant amount of power during computation. In this paper, we propose a preliminary power prediction model which can be used by developers for building power-efficient GPU applications. Using this proposed work, developers can estimate the power consumption of a GPU application during implementation without having to execute it on actual hardware. Our model combines the information derived from static analysis of a CUDA program and a machine learning-based model. We have utilised decision tree technique to validate results across three different GPU architectures: Kepler, Maxwell and Volta. Observed R^2 score value using the decision tree model is 0.8973 for Volta architecture.**

***Index Terms*—GPGPU; High Performance Computing; Power Modelling; Prediction Technique**

I. Introduction

After more than two decades of evolvement in the Graphics Processing Units (GPU), a lot has changed in terms of its die size, performance and power consumption. First 3D rendering GPUs hosted about a million transistors, were smaller than $100mm^2$ in die size and consumed a few watts of power. Todays GPU, an NVidia Titan V, hosts 21.1 billion transistors on a die size of $815mm^2$ and consume 250W of power. The performance gain for the new GPUs is far more significant than its predecessors at the cost of the power consumed [1].

With the increase in demand for performance, developers must understand the crucial features of their code which contribute most to the power consumption. To do so, they must be equipped with a tool to analyze and predict power consumption employing their built code. This tool can help them to refactor their code in order to become more power-efficient. Compared to CPU power measurement techniques such as Wattch [2], and McPAT [3], existing tools for GPU power measurement are not standardized and flexible. The complexity of GPU architecture has always been a hurdle in power modelling of GPUs. Direct measurement of power using hardware-based solutions such as counters is considered to be the most reliable technique [1]. However, it is not always the best solution since it needs a physical presence which impacts the flexibility in the use of these techniques.

Researchers have built power prediction tools in the past based on using performance counters collected by running an application on a GPU [4] [5] [6] [7]. These power prediction tools are built either using a hardware simulator and are specific to an architecture [4] [7], or to a hardware component [8]. In contrast, we attempt to build a prediction approach purely based on static analysis of a CUDA program. Such an approach is lightweight and does not demand users to take power consumption measurement using special measurement equipment. Since this is an off-line tool, it can help developers to understand power consumption profile during application development.

In this work, we plan to answer two research questions (RQ):

- RQ1: Is it possible to build an architecture agnostic model to predict the power consumed by a CUDA program with tolerable error and precision?
- RQ2: Which features of the CUDA program considered in this work impact most significantly to the power consumption of GPU?

The rest of this paper is organized as follows: Section II presents the related work. We discuss the proposed power prediction model in section III. The experimental dataset is discussed in section IV. We present the modelling technique and results in section V. Further analysis of this work is provided in section VI. We conclude this work in section VII.

II. Related Work

In one of the popular work on predicting power by Sunpyo, et al. [4], they proposed an Integrated Power and Performance prediction (IPP) model for GPU architecture. Their empirical power model for GPU predicts execution time to compute the dynamic power events using microbenchmarks and hardware features. Their model predicts power consumption and performance with an average error of 8.94%. However, their model fails to predict for control-flow intensive applications and asymmetric applications. Also, since they predict using access rates for each instruction, IPP does not require the actual number of total instructions. If the number of instructions significantly affect the power consumption, they will have to incorporate the same in their model.

978-1-6654-1485-2/20/$31.00 ©2020 IEEE
DOI 10.1109/ISPA-BDCloud-SocialCom-SustainCom51426.2020.00149

A simulator called GPUSimPow [9] helps estimate the power consumption of a kernel on a given GPGPU architecture without physically running the kernel on the GPU, but by using hardware models for the internal components of GPUs. It breaks the overall power estimate to individual component power usage. Because the hardware is modelled directly, authors claim that simulation is possibly the most accurate method of estimating the energy consumption of a kernel. However, it is also a more time-consuming method as the entire execution of the kernel from start to finish needs to be modelled precisely. For a set of benchmarks, GPUSimPow shows an average relative error of 11.7% for GT240 and 10.8% for GTX580.

Lim et al. [10] constructed a power model for GPUs by combining empirical data with data obtained from McPAT, a CPU power tool. The model is trained using a particular set of benchmarks after which it is used for predicting power. The average error for Merge Benchmarks is reported to be 12.8%. Their model also deals with a perspective from the architecture point of view and not from an application point of view, which we wish to propose. Wang [11] extended the popular GPU simulator GPGPUSim [12] to include power modelling. However, the accuracy of his results was not published.

Attempts have been made in the past to utilize statistical and machine learning models for power prediction. Nagasaka et al. [5] proposed a statistical method to estimate the power consumption of a kernel. This is done using GPU performance counters for CUDA applications. The approach uses GPU performance counters as the independent variable and power consumption as the dependent variable. Utilizing this data, a linear regression model is trained. However, this method fails to yield accurate estimates when performance counters are not available for features.

A statistical model for power and performance by Zhang et al. [6] demonstrate how GPU throughput and power consumption is dependent on architecture attributes for an ATI GPU. They utilized Random Forest, which resulted in R-square value of 79.7% and Mean Absolute Error of 13.1%, which they claim indicate high accuracy. Their study is based on understanding the hardware architecture features, which contributes the most to power consumption.

Though power estimation models are designed to provide accurate measurements, direct hardware measurement is still considered the most precise and simplified approach. Designing an efficient prediction model of power consumption in GPUs is a critical problem and will benefit programmers immensely since it can point out the design issues in their application. The observations such as by Sunpyo et al. [4] that power consumption increases as we increase the number of cores can benefit developers to refactor their code accordingly. Power consumption depends on the hardware components utilized for an application which depends on the type of instructions executed. However, to the best of our knowledge, not much work is carried out in program analysis based power models. A survey by Bridges et al. [1] strongly recommends designing a flexible and accurate GPU power prediction framework which requires less expertise as a future direction for research work.

III. POWER PREDICTION MODEL

In this work, we aim to build a model purely based on program analysis using hardware features and PTX code. This model if built successfully can replace the need of using external hardware for collecting power data, save energy in running an application for determining its power usage, and serve developers to refactor their code to build an energy-efficient code. Building a static analysis based power prediction model involves using program details and hardware features.

We need a data-driven approach for designing a prediction model since we do not have any past research literature on the relationship of power and program features from algorithmic or analytical model perspective. We utilize machine learning techniques since they build a reliable power prediction model by repeatedly learning from existing program analysis data with our limited knowledge of the relationships between application features and GPU power consumption.

Since we aim to predict power consumption using static analysis of CUDA code, the features considered for prediction model must be obtained without actually running the code on GPU. Hence we utilized a program analysis algorithm, a JAVA based tool. Data for all the features based on the architecture constraints of GPU were obtained using this model.

We use a data-driven approach wherein we consider all the features which we think might contribute to the power consumption. We try to consider features which can be extracted from program analysis that can lead to an increase in power consumption. Some initial features include the number of Waves (waves), inst_issue_cycles. Later on, as we progressed further, we added more features like Simulated number of Compute, Global, Shared and Miscellaneous instructions for observing the effect of program execution specifics on power consumption. We considered Occupancy as a feature to understand the effect of thread-level parallelism on GPU power consumption. Out of all these features, we systematically try to select the features that are likely to contribute to predicting power consumption accurately. We have listed all the features considered for feature selection and its detailed description in Table I.

A. Feature Engineering

We collected fifteen features using program analysis and simulation algorithm. However, having highly correlated features in our model does not improve the performance of the model, and on the other hand, they may mask the interactions among various features. This suggests us to include only one of the highly correlated features in our model.

To start with, we use the correlation analysis to observe the relationship of attributes among themselves. We consider the Pearson correlation coefficient for this study.

a) ***Pearson coefficient:*** This coefficient evaluates the linear relationship between two variables. It has a value between +1 and 1, where 1 is a total positive linear correlation,

TABLE I: Features Considered for feature selection

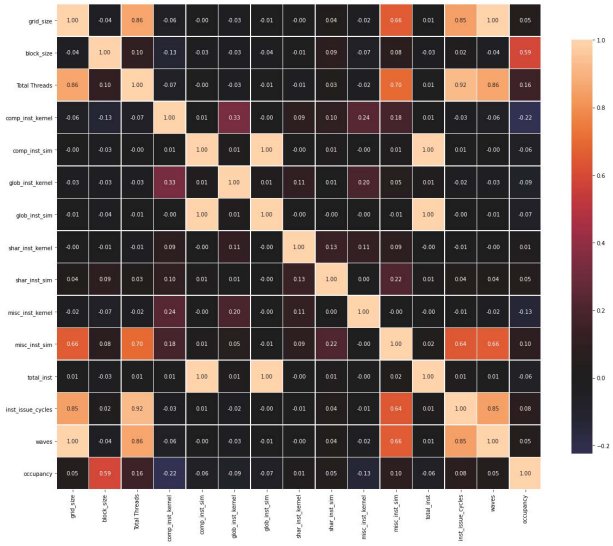
Notation	Description
block_size	Number of threads per block
grid_size	Number of blocks
comp_inst_kernel	Number of computation instruction in kernel
comp_inst_sim	Number of simulated computation instructions
glob_inst_kernel	Number of global memory instruction in kernel
glob_inst_sim	Number of simulated global memory instructions
inst_issue_cycles	Instruction Issue Cycles
misc_inst_kernel	Number of miscellaneous instruction in kernel
misc_inst_sim	Number of miscellaneous instructions
occupancy	Ratio of active warps on an SM to maximum number of active warps supported by the SM
shar_inst_kernel	Number of shared memory instruction in kernel
shar_inst_sim	Number of simulated computation instructions
total_threads	Total number of threads launched
total_inst	Total number of instructions simulated
waves	Number of waves of blocks executed on SMs

0 is no linear correlation, and 1 is a total negative linear correlation.

As seen in the heat-map for Pearson coefficient in Fig. 1 we can finalize feature selection by reading heat-map as:

- The correlation between two variables(x and y) can be obtained from the cell whose row corresponds to x and column corresponds to y or vice versa.
- The colour map towards the right end of the heat-map represents the range of values for the coefficient represented by them, although their exact values are mentioned in their corresponding cells.
- Generally absolute values of correlation greater than 0.9 are considered highly correlated while values close to zero are considered independent variables.

Fig. 1: Pearson Correlation Coefficient



B. Result of feature selection techniques employed

With the resulting heat-map of the Pearson correlation coefficient in Fig. 1, we can observe the following:

- Features like waves, grid_size and Total Threads are highly correlated with the feature inst_issue_cycles with a correlation coefficient of 0.85,0.92 and 0.85 respectively. Hence we drop these features from our model
- comp_inst_sim, glob_inst_sim, and total_inst are highly correlated with each other with a correlation coefficient of 1. So we consider only glob_inst_sim for our model and drop the other two features.

We also utilize the decision tree's feature importance results while finalizing the features in order to include the features which contribute the most to power consumption. The features included in building power model after applying feature selection technique are presented in Table II.

IV. Experimental Dataset of Features

The experimental dataset is collected by either using available tools such as CUDA Occupancy Calculator and program analysis algorithm using PTX code. GPU power consumption varies with different kernel workloads. For example, accessing the global memory consumes more power than accessing shared memory, accessing on-chip registers, and performing floating-point arithmetic operations [13]. We launch benchmark kernels with different launch parameters, i.e. grid size and block size to gather this dataset. Here, we discuss the selected features taken into consideration for building a power prediction model and the methods used to obtain them:

A. Power Consumption

Power values obtained for the dataset were generated by utilizing two popular tools. These tools used for collecting the amount of power consumed are:

- Unified Power Profiling Application Programming Interface(UPPAPI) [14]
- NVML API [15]

In both the methods used, power values are obtained by polling during the execution of the kernel. We then take the average value of power to minimize the error in the measured power.

B. Occupancy

Occupancy measures the ratio of the actual number of warps executing on an SM to the maximum number of warps allowed for an SM. We consider occupancy as a feature in our model to understand the effect of thread-level parallelism (TLP) on GPU power consumption. The occupancy value considered in this work is theoretical occupancy which captures the limit for occupancy restricted by the kernel launch configuration and the capabilities of the CUDA device. Achieved occupancy is measured during runtime of code, and the two values may slightly differ. However, to achieve our goal of prediction using static analysis, we consider theoretical occupancy in our work.

We utilize CUDA Occupancy Calculator [16], which reports the theoretical maximum occupancy that can be achieved on the execution of a particular kernel by taking in four inputs:

- number of threads per block
- number of registers used per thread
- number of shared memory used by a block of thread
- CUDA Compute Capability of the device under consideration

C. Computing Instruction Features

PTX code consists of simple computing instructions (e.g. add, mul) which run on single precision cores, floating-point instructions (e.g. divf) which run on double precision cores and special instructions (e.g. sqrt) which execute on special function unit. We count the number of each type of instruction present in a GPU kernel by static analysis of PTX code for GPU kernel. This generates comp_inst_kernel for computing instructions. The number of simulated computed instructions per SM (comp_inst_sim) is computed using program analysis algorithm ,since one needs to take into account GPU hardware specification for this feature.

D. Global Memory Instruction Features

Power consumption of a GPU is sensitive to accessing global memory [17]. Global memory access instructions include global load and global store which are considered in this study as glob_inst_kernel. They are counted by dissecting the PTX code. This glob_inst_kernel describes the number of global access instruction present in the code. We compute the number of instructions executing on one SM (glob_inst_sim) using program analysis algorithm. Effect of global memory on the power consumption could be due to the access pattern (coalescing or non-coalescing access) of CUDA code. However, since computing this with static analysis is not possible, we could not consider this factor in this work. This can be explored as future work.

E. Shared Memory Instruction Features

Shared memory instructions can also contribute to the power consumption of GPU, especially when they are a significant number of bank conflicts [17]. Shared memory instructions include a shared load and a shared store. We analyze the PTX code to compute shar_inst_kernel which captures the count of these instructions in a CUDA kernel. We could not capture the effect of number of bank conflicts on power consumption with static analysis for this work. However, we observed the effect of a significant number of shared memory instructions by including shar_inst_sim per SM computed using program analysis algorithm.

F. Program Analysis Algorithm

We develop a Java-based program analysis algorithm which enacts GPU execution behaviour. This algorithm uses static analysis of CUDA code and hardware features made available by GPU vendor. We generate simulated features for the dataset using this application. We present this program analysis algorithm in detail in algorithm 1. Notations used in program analysis algorithm are mentioned in Table III.

In algorithm 1, we take as input the PTX code along with launch parameters, i.e. the number of blocks, number of threads per block, and number of loop iterations (n_loop). These inputs cannot be extracted by static analysis and are very crucial for generating features. Also, developers can easily provide these details based on application development.

We believe one wave of execution on SMs is dependent on the maximum number of threads on SM allowed by hardware and the maximum number of threads launched by an application. Considering the maximum number of threads which can reside per SM (nTh_m), and the total number of threads which are scheduled per SM (nTh_schd) (computed using launch configuration), the number of threads per wave (nTh_wave) is calculated. This continues till nTh_schd is zero, which ensures that all the threads complete their execution.

Each basic block is represented by data flow graph G. Each node in this graph represents instructions of a basic block. By reading the type of instruction represented by each node, the number of instruction for a particular type is counted. We calculate computing instruction latency (comp_lat_sim), global memory instruction latency (glob_lat_sim). If this basic block is a loop, we multiply each of these features by the number of loop iterations (n_loop) supplied as input by the user.

G. Other Features

Launch parameter (block_size, grid_size) are supplied by the user since they cannot be obtained from PTX analysis. Further using the published architecture details by vendors such as the number of warp scheduler and dispatch unit per SM along with total instructions executed, we calculate inst_issue_cycle. We also use misc_inst_kernel for accounting miscellaneous instructions (e.g. return, sync, label). Total instruction represents the sum of all types of simulated instructions.

All the features considered for our power prediction model are obtained by static analysis of the CUDA PTX code. Data collection of no feature requires the user to run the CUDA application. All the features and its sources are mentioned in detail in Table II.

TABLE II: Features Used for Power Prediction Model

Application Features	Source
block_size	Provided by user
comp_inst_kernel	PTX Analysis
glob_inst_kernel	PTX Analysis
glob_inst_sim	Program Analysis Algorithm
shar_inst_kernel	PTX Analysis
shar_inst_sim	Program Analysis Algorithm
misc_inst_kernel	PTX Analysis
misc_inst_sim	Program Analysis Algorithm
inst_issue_cycles	Calculated based on user input
occupancy	Cuda Occupancy Calculator

TABLE III: Notations for Program Analysis Algorithm

Parameter	Notation	Source
Number of Streaming Multiprocessors (SM)	nSM	Device Query
Maximum number of threads per SM	nTh_m	Device Query
Number of blocks launched	nB	User Supplied
Number of threads per block	nT	User Supplied
Number of loop iterations	$loop\_count$	User Supplied
Total threads scheduled per SM	nTh_schd	Computed during algorithm
Number of threads in current wave	nTh_wave	Computed during algorithm

Algorithm 1 Program Analysis Algorithm

procedure PROGRAMANALYSIS(PTX,nB, nT, n_loop)
  nTh_schd $= \lceil \frac{nB}{nSM} * nT \rceil$;
  waves=0;
  while *nTh_schd* ≥ 0 **do**
    nTh_wave=nTh_schd >nTh_m ? nTh_m: nTh_schd;
    for each $BasicBlock$ in PTX **do**
      for each instruction i in G **do**
        if i.Type == Computing **then**
          comp_inst_sim+=1;
        end if
        if i.Type == Global Memory **then**
          glob_inst_sim+=1;
        end if
        if i.Type == Shared Memory **then**
          shar_inst_sim+=1;
        end if
        if i.Type == Miscelleneous **then**
          misc_inst_sim+=1;
        end if
      end for
      if $BasicBlock$ has a loop **then**
        Multiply each feature by n_loop
      end if
    end for
    nTh_schd = *nTh_schd* $- nTh_m$;
    waves=waves+1;
  end while
  return populated feature values;
end procedure

H. Dataset Collection

The model aims to be flexible and accurate on several GPU architectures; hence we are utilizing the Rodinia [18], CUDA SDK [19] and the Tango GPU [20] benchmarks for our purpose. All the benchmarks under study are tabulated in Table IV. These benchmark applications represent a variation of benchmarks in terms of its sizes, nature (e.g. compute-bound, memory-bound) and complexity (e.g. control divergence, memory access patterns). We ran each of these benchmarks with different launch configurations (grid size, block size) and datasets. We collected 1100 such data points using these benchmarks for each architecture under consideration.

I. Data Scaling

Machine Learning models work much better if the values of the features in the dataset are relatively on a similar scale. However, the tree-based models are scale-invariant; it is better to scale the data in order to fasten the calculations in a machine learning algorithm. We use the MinMaxScalar here since it preserves the shape of the distribution of the features and reduces the loss of information.

TABLE IV: CUDA Benchmarks Considered in this Study

Benchmark Name	Source
Convolution Seperable	CUDA Toolkit Samples
Convolution Texture	CUDA Toolkit Samples
Concurrent Kernels	CUDA Toolkit Samples
Fast Walsh Transform	CUDA Toolkit Samples
FDTD	CUDA Toolkit Samples
Histogram	CUDA Toolkit Samples
Heartwall	CUDA Toolkit Samples
N Body	CUDA Toolkit Samples
QRNG	CUDA Toolkit Samples
Monte Carlo Multi GPU	CUDA Toolkit Samples
Merge Sort	CUDA Toolkit Samples
Scan	CUDA Toolkit Samples
Sobol Qring	CUDA Toolkit Samples
Clock	CUDA Toolkit Samples
BlackScholes	CUDA Toolkit Samples
Binomial Options	CUDA Toolkit Samples
Aligned Types	CUDA Toolkit Samples
Cdp Simple Quicksort	CUDA Toolkit Samples
Matrix Multiplication	CUDA Toolkit Samples
Transpose	CUDA Toolkit Samples
Vector Addition	CUDA Toolkit Samples
Stereo Disparity	CUDA Toolkit Samples
simpleCUFFT	CUDA Toolkit Samples
simpleHyperQ	CUDA Toolkit Samples
Saxpy	CUDA Toolkit Samples
Breadth-First Search	Rodinia Benchmark
Needleman-Wunsch	Rodinia Benchmark
LU Decomposition	Rodinia Benchmark
Shfl Scan	Rodinia Benchmark
Kmeans	Rodinia Benchmark
Srad	Rodinia Benchmark
Hotspot	Rodinia Benchmark
Gaussian	Rodinia Benchmark
CFD Solver	Rodinia Benchmark
NN	Rodinia Benchmark
RESNET	Tango GPU
SqueezeNet	Tango GPU
LSTM	Tango GPU
CifarNet	Tango GPU
GRU	Tango GPU

V. MODELLING TECHNIQUES AND RESULTS

Advanced machine learning tools are being effectively used to predict architecture nuances for modern architectures. Among the wide variety of these available techniques, tree-based machine learning techniques have proved to be highly efficient when the data is limited and has a significant number of feature attributes. Since this is preliminary work, we utilize Decision Trees to predict the power consumption using our selected features.

A. Decision Trees

Decision Tree [21] has been proved to be a very effective technique. Decision Trees builds the regression or classification model based on a tree-based structure, containing decision nodes and leaf nodes. The cost of using a decision tree is logarithmic in the number of data points used to train the model. Decision Trees could provide a basic idea for evaluating the performance of other complex Tree-Based models on the power consumption data. We use the sklearn library for building our Decision Tree model, which uses the

TABLE V: Hardware Features values for GPUs under consideration

GPU Hardware Features	Tesla K20	Tesla M60	Tesla V100
Architecture	Kepler	Maxwell	Volta
Compute Capability	3.5	5.2	7.0
Number of SMs	13	16	80
Number of cores Per SM	192	128	64

TABLE VI: Results across GPU architectures

Architecture	**R^2 score**	**RMSE**	**MAE**
Tesla K20	0.8199	10.8832	5.7835
Tesla M60	0.8533	7.3676	3.3581
Tesla V100	0.8973	12.2025	5.4570

modified version of the CART (Classification and Regression Trees) algorithm.
We train the model by tuning various hyper-parameters such as minimum samples required to split an internal node, maximum depth of the tree and maximum features required for the best split and employing pruning which avoids the overfitting of the model.

B. GPU Architectures Under Study

We want to test our proposed prediction model across multiple architectures to observe its robustness and flexibility. To do so, we tested our model on Tesla K20 (Kepler Architecture), Tesla M60 (Maxwell Architecture), and Tesla V100 (Volta Architecture). Hardware features for these GPU machines are mentioned in Table V. For each of these GPUs, we ran and collected power value using UPPAPI and NVML API. Latency for each instruction type was collected using micro-benchmarking on each of these GPUs. Program Analysis data was collected for each benchmark by running the GPU simulation algorithm which utilizes hardware features of these GPUs.

C. Model Evaluation

We will be using three essential metrics to validate our model based on the mean of 5-fold cross-validation scores: R^2 score, Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). We have tabulated results for these metrics in Table VI for all three GPU machines.

R^2 score for all three architectures are decent, and hence we claim our model produces accurate predictions. RMSE values for all three architectures also suggest that our model is robust and accurate. As seen in fig. 2, for all of the popular benchmarks seen our predicted power is quite precise against measured value.

VI. Analysing Results

Through this work, we attempt to answer the following Research Questions (RQ) based on the observed results and model analysis:

RQ1: Is it possible to build an architecture agnostic model to predict the power consumed by a CUDA program with tolerable error and precision? As seen in the experimental results, we validated our model across three different GPU architectures: Kepler, Maxwell, Volta. All three validation metrics, i.e. R^2 score, RMSE, and MAE results, suggest that our model is architecture agnostic since it performs efficiently across all three architectures. Our model was tested for a heterogeneous mixture of CUDA kernel benchmarks with varying memory access patterns, launch configuration, applicability and execution bound type.

RQ2: Which features of the CUDA program considered in this work impact most significantly to the power consumption of GPU?
The observed power prediction model result suggests that we can relate power consumed by CUDA kernel and program features. We analyze which of the program features considered in this work contribute the most to power prediction. Based on the feature importance analysis using Decision Tree for all three architectures as shown in fig.3, fig.4, and fig.5. We found that instruction inst_issue_cycle contribute the most to power consumption across three architectures. This attribute is dependent on warp scheduler and instruction dispatcher.

Global memory access instructions have a significant effect on power consumption. Both instruction count (glob_inst_kernel) and simulated number of instructions (glob_inst_sim) have ranked higher in all three architectures. This supports the observation by Song et al. [17] on the influence of global memory on power consumption.

Computing instructions count (comp_inst_kernel) also affects the power prediction as effectively as miscellaneous instruction count (misc_inst_kernel) and the number of simulated miscellaneous instructions (misc_inst_sim). Some of the computing instructions (fmadd, divf) have much higher latency compared to others (add, mul) [22]. As future work, we would like to explore the effect of latency of each type of instruction on GPU power consumption.

Effectiveness of the number of blocks launched (block_size) follows features, as mentioned earlier. The number of blocks launched can represent the number of SMs activated during CUDA kernel execution lifetime. This is because each new block is mapped to one SM. However, in our observation, it has affected the power consumption moderately.

Occupancy of the CUDA kernel should have been one of the most crucial features which contribute to an increase in power consumption since it describes the maximum number of threads residing on SM. However, its feature importance value has been lower for all three architectures. This could be since we used a CUDA Occupancy Calculate, which gives an approximate value of occupancy, which differs from the one collected by profiling the CUDA kernel.

Among all the instruction types considered in this study, shared memory instructions have proved to be least contributing to the power consumption. Both the shared memory features (shar_inst_kernel, shar_inst_sim) have ranked least in feature importance across all three architectures. Such an observation could be since shared memory is much faster compared to global memory. As future work, we can also explore the effect of register memory on GPU power consumption.

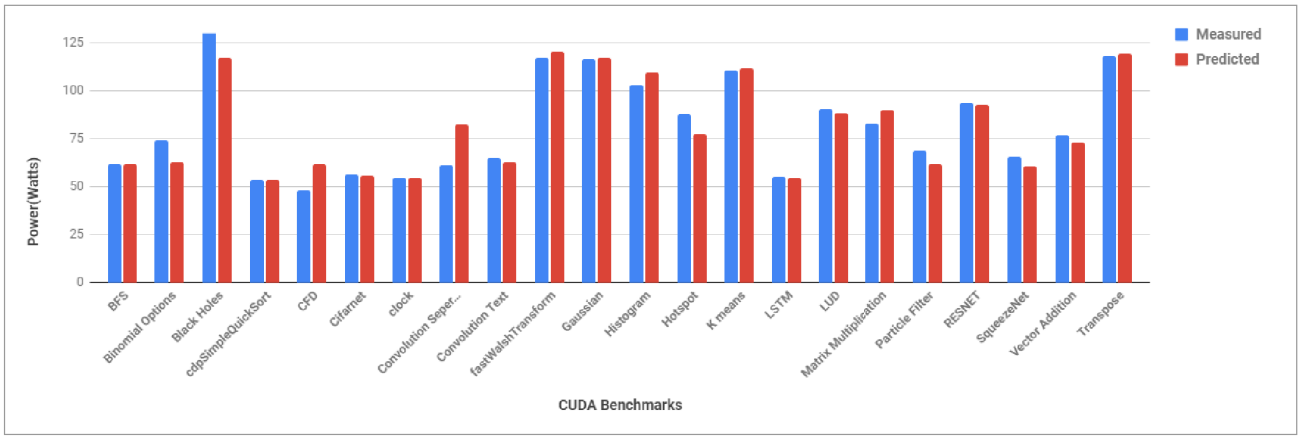


Fig. 2: Measured Vs Predicted Power Consumption

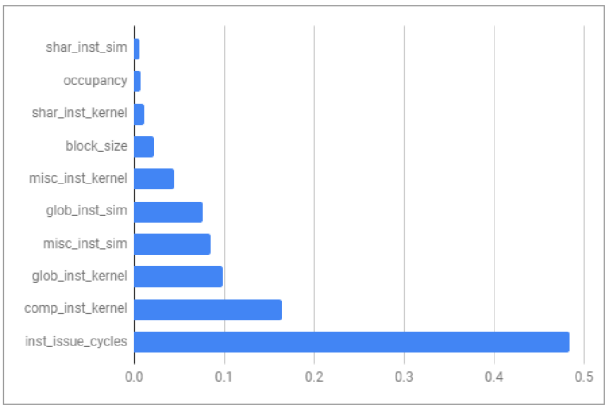


Fig. 3: Relative feature importance for Tesla K20

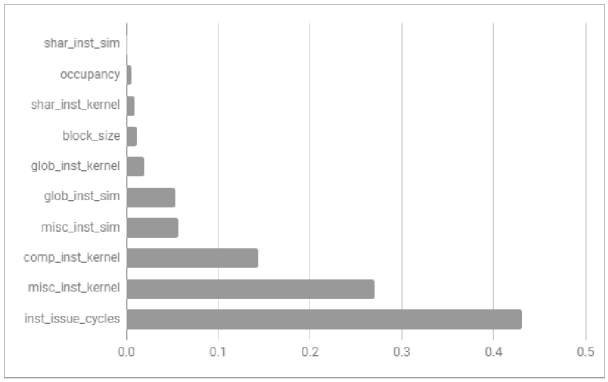


Fig. 5: Relative feature importance for Tesla V100

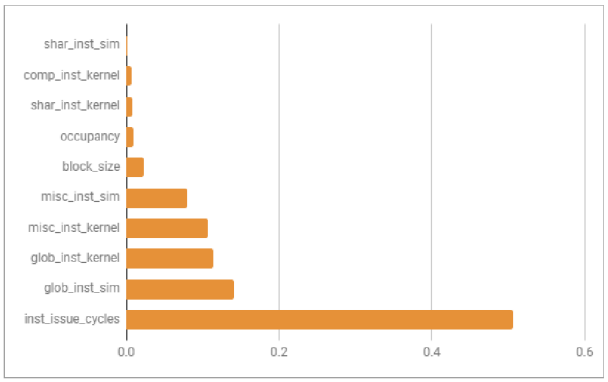


Fig. 4: Relative feature importance for Tesla M60

VII. Conclusion

We propose a preliminary power prediction model purely based on static analysis of CUDA kernel and its launch configuration using a machine learning technique. We validate this model using decision tree regression across three GPU architectures: Kepler, Maxwell, and Volta. Observed R^2 score for these respective architectures is 0.8199, 0.8533, and 0.8973. These results suggest that power prediction is feasible using program analysis details of CUDA code and launch configuration without executing it on GPU. We also analyse the relationship between program features and its effect on power consumption using the feature importance of decision tree regression. Future work can involve including some hardware-specific features which can be computed using vendor-supplied information. We would also like to explore further other machine learning techniques which can improve the observed prediction error.

References

[1] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding gpu power: A survey of profiling, modeling, and simulation methods," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 41:1–41:27, Sep. 2016. [Online]. Available: http://doi.acm.org/10.1145/2962131

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, 2000, pp. 83–94.

[3] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing

modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[4] S. Hong and H. Kim, "An integrated gpu power and performance model," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 280–289. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815998

[5] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of gpu kernels using performance counters," in *Green Computing Conference, 2010 International*, ser. GREENCOMP '10. IEEE, 2010, pp. 115–122.

[6] Y. Zhang, Y. Hu, B. Li, and L. Peng, "Performance and power analysis of ati gpu: A statistical approach," in *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*, 2011, pp. 149–158.

[7] J. Chen, B. Li, Y. Zhang, L. Peng, and J. Peir, "Tree structured analysis on gpu power study," in *2011 IEEE 29th International Conference on Computer Design (ICCD)*, 2011, pp. 57–64.

[8] V. Jatala, J. Anantpur, and A. Karkare, "Greener: A tool for improving energy efficiency of register files," 2017.

[9] J. Lucas, S. Lal, M. Andersch, M. Alvarez-Mesa, and B. Juurlink, "How a single chip causes massive power bills," in *IEEE International Symposium on Performance Analysis of Systems and Software, 2009*, ser. ISPASS '13. IEEE, 2013, pp. 97–106.

[10] J. Lim, N. B. Lakshminarayana, H. Kim, W. Song, S. Yalamanchili, and W. Sung, "Power modeling for gpu architecture using mcpat," vol. 19. ACM, 2014.

[11] G. Wang, Y. Lin, and W. Yi, "Kernel Fusion : an Effective Method for Better Power Efficiency on Multithreaded GPU," in *2010 IEEE/ACM International Conference on Green Computing and Communications*, 2010, pp. 344–350.

[12] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software, 2009*, ser. ISPASS 2009. IEEE, 2009, pp. 163–174.

[13] Zhe Fan, Feng Qiu, A. Kaufman, and S. Yoakum-Stover, "Gpu cluster for high performance computing," in *SC '04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004, pp. 47–47.

[14] M. Chadha, A. Srivastava, and S. Sarkar, "Unified power and energy measurement api for hpc co-processors," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, 2016, pp. 1–8.

[15] NVIDIA, "Nvml api, reference manual," (accessed on 04/12/2019). [Online]. Available: https://docs.nvidia.com/pdf/NVML_API_Reference_Guide.pdf

[16] N. Corporation, "Cuda occupancy calculator," (accessed on 20/12/2019). [Online]. Available: https://docs.nvidia.com/cuda/cuda-occupancy-calculator/index.html

[17] S. Song, C. Su, B. Rountree, and K. Cameron, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," in *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, May 2013, pp. 673–686.

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 44–54.

[19] NVIDIA, "Cuda samples," (accessed on 02/10/2019). [Online]. Available: https://docs.nvidia.com/cuda/cuda-samples/index.html

[20] A. Karki, C. Palangotu Keshava, S. Mysore Shivakumar, J. Skow, G. Madhukeshwar Hegde, and H. Jeon, "Tango: A deep neural network benchmark suite for various accelerators," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 137–138.

[21] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone, "Classification and regression trees," 1983.

[22] G. Alavani, K. Varma, and S. Sarkar, "Predicting execution time of cuda kernel using static analysis," in *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2018, pp. 948–955. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/BDCloud.2018.00139