

DSO: A GPU Energy Efficiency Optimizer by Fusing Dynamic and Static Information

Qiang Wang*, Laiyi Li*, Weile Luo†, Yijia Zhang‡, Bingqiang Wang‡

*Harbin Institute of Technology (Shenzhen)

†The Hong Kong University of Science and Technology (Guangzhou)

‡Peng Cheng Laboratory

*qiang.wang@hit.edu.cn, *23s151122@stu.hit.edu.cn, †wluo976@connect.hkust-gz.edu.cn, ‡{zhangyj01,wangbq}@pcl.ac.cn

Abstract—Increased reliance on graphics processing units (GPUs) for high-intensity computing tasks raises challenges regarding energy consumption. To address this issue, dynamic voltage and frequency scaling (DVFS) has emerged as a promising technique for conserving energy while maintaining the quality of service (QoS) of GPU applications. However, existing solutions using DVFS are hindered by inefficiency or inaccuracy as they depend either on dynamic or static information respectively, which prevents them from being adopted to practical power management schemes. To this end, we propose a novel energy efficiency optimizer, called DSO, to explore a light weight solution that leverages both dynamic and static information to model and optimize the GPU energy efficiency. DSO firstly proposes a novel theoretical energy efficiency model which reflects the DVFS roofline phenomenon and considers the tradeoff between performance and energy. Then it applies machine learning techniques to predict the parameters of the above model with both GPU kernel runtime metrics and static code features. Experiments on modern DVFS-enabled GPUs indicate that DSO can enhance energy efficiency by 19% whilst maintaining performance within a 5% loss margin.

Index Terms—GPU Modeling, Energy Efficiency, Dynamic Voltage and Frequency Scaling

I. INTRODUCTION

In recent years, there has been a significant increase in the utilization of graphics processing units (GPUs) as accelerators in high-performance computing (HPC). This trend has been driven by the growing demand for energy-efficient solutions, particularly due to the rise of artificial intelligence (AI) applications [1]–[3]. One prominent instance is the advanced language model GPT-3 [4], which was designed with over 150 billion parameters to generate human-like texts. The training cost for GPT-3 exceeds 4.6 million dollars, equivalent to nearly 120 years of electricity consumption by an average household. These staggering figures underscore the significance of implementing effective mechanisms to enhance the energy efficiency of these systems. Even a modest 5% reduction in energy consumption can have a substantial impact.

Dynamic voltage and frequency scaling (DVFS) is a promising technique for GPUs which enables the adjustment of devices to lower performance/power states. DVFS optimizes GPU performance and power by adjusting voltage and frequency levels, offering substantial energy savings with minimal performance impact [5], [6]. Recent investigations [3], [7] demonstrate that the utilization of DVFS techniques in graphics processing units (GPUs) engaged in deep neural network (DNN) applications resulted in energy savings of up to 26% especially in the common DNN inference scenario. Several existing studies [8]–[12] on energy conservation through DVFS rely on runtime information provided by GPU profiling tools, such as *nvprof* for Nvidia GPUs. These tools have proven to be effective in modeling the performance and power changes under different DVFS settings, given the high correlation between the performance counters of each GPU sub-component and the execution time/power. However, two drawbacks hinder their practical online usage. Firstly, the profiling overhead associated with these tools is typically significant because these profiling tools often require multiple replays of the target application, resulting in heavy computational costs. Secondly, some of these tools necessitate modifications to the application source code, which is not user-friendly and may not be available for online submitted jobs.

Another branch of analyzing the performance and power behavior of GPU applications is static information modeling, which involves examining GPU low-level assembly codes such as PTX¹ and SASS². This approach relies on using the GPU assembly of the kernels, which can be obtained at compile-time or by disassembler tools. One advantage of this approach is that it does not require modifying users' applications or pre-executing them to collect runtime information. Moreover, this type of static modeling introduces new usage scenarios, such as facilitating the evaluation of how changes in the source code can affect the DVFS behavior of applications. However, due to the lack of GPU runtime information such as cache hit rate and compute resource occupancy, the prediction errors for execution time are typically high.

We argue that an ideal energy efficiency optimizer for DVFS-based GPUs should be efficient and accurate, which cannot tolerate the extremely high overhead of those existing profiling tools. Recently, data center GPU manager (DCGM) [13] published by Nvidia is a lightweight tool to manage and monitor the GPUs in data center environments. It provides a

¹<https://docs.nvidia.com/cuda/parallel-thread-execution/index.html>

²<https://docs.nvidia.com/cuda/cuda-binary-utilities/>

set of powerful tools and APIs that allow system administrators and operators to monitor the health, performance, and utilization of the GPUs in a non-intrusive manner with negligible cost. However, the metrics provided by DCGM is just a subset of those by *nvprof*, which may decrease the model accuracy. To this end, we finally come to the optimization framework, called DSO, that leverages both the DCGM metrics (dynamic information) and the PTX codes (static information). We summarize the contributions of DSO as follows.

- We propose a novel parameterized theoretical model of GPU energy efficiency considering both the effects of DVFS and the tradeoff between performance and energy consumption. The optimization solution for a GPU kernel is also explicitly derived to tune the best DVFS setting.
- We design a machine learning based scheme to predict the parameters of the proposed theoretical model leveraging both the runtime metrics from the lightweight DCGM profiling tool and the static features from the PTX codes.
- Validated on 20 real applications (not used during training) among two contemporary GPUs, the model trained on only micro-benchmarks shows considerably low errors (mostly within 5%) for both performance and power prediction. The average energy conservation observed in our optimization results achieves 19% on Tesla V100 on average compared to the default setting with no more than 5% performance loss, all without heavy offline profiling.

II. BACKGROUND AND MOTIVATION

A. GPU DVFS

Performance and power modeling is essential for energy conservation in different DVFS settings, dictating the total energy consumption of GPU applications. Recent studies [5]–[7] have shown that GPU DVFS behaviors are more complex than CPUs when altering voltages and frequencies, sometimes even proving contrary to conventional CPU DVFS. As modern GPUs have two primary frequency domains—core frequency for stream multiprocessors (SMs) speed and memory frequency for GPU memory bandwidth, our efforts centralize on optimizing frequency control based on application behaviors.

B. The Input Sources for GPU Modeling

Two input types are generally used in previous works for GPU performance, power modeling, and DVFS management: dynamic and static information. Dynamic information, also referred to runtime information, is hardware-dependent, collected during one execution of the target application, usually through profiling tools like *nvprof* [14]. Despite high prediction accuracy, the profiling overhead may inhibit practical DVFS energy optimization due to the lack of real-time measurements. Static information is hardware-independent, referring to GPU code features obtained before kernel execution. The NVIDIA Parallel Thread Execution (PTX), an intermediate assembly language for NVIDIA GPUs, is often utilized. It can be extracted from CUDA binary files using the Nvidia disassembler tool *cuobjdump*, allowing analysts to link each instruction with the GPU components involved in its

execution. Moreover, lightweight monitoring tools like *nvidia-smi* [15] and DCGM [13] are available for tracing GPU status. As they relate to GPU hardware runtime, they fall under the category of runtime information.

C. Why Dynamic and Static Information Fusion

As stated above, these two input source types have their own advantages and drawbacks. A practical energy management scheme with GPU DVFS should be efficient and accurate in terms of modeling and optimization. Notice that DCGM can monitor the runtime utilization of different GPU components with negligible overhead. We consider it as the substitute of the heavy profiling tool such as *nvprof*. To fulfill the information gap between DCGM and *nvprof*, we further utilize the PTX code features as complement when designing our energy efficiency optimizer, which finally comes to the scheme of dynamic and static information fusion.

III. RELATED WORK

There have been many research studies about understanding the impacts on GPU performance and energy consumption brought by DVFS as well as optimizing them solely or jointly. They generally take advantage of either physical runtime metrics (also referred as dynamic information) and GPU kernel codes (also referred as static information).

Dynamic information-based methods initially used micro-benchmarks to profile GPU hardware components [16]–[18]. DVFS-aware prediction models were developed by Wang et al. [12] to quantify each GPU component’s performance contribution. Guerreiro et al. employed similar tactics for power contributions [19], [20]. ML-based methods, such as one developed by Wang et al. [21], encapsulated interplay influences of diverse instructions and achieved below 10% average errors. The high accuracy of the above methods indicates the importance of runtime information, especially for contemporary GPU architectures with rich sub-components.

Static information-based methods began with Hong et al. [22], [23], who modelled GPU kernel performance and power consumption using static CUDA and PTX code analysis. Guerreiro et al. [24] expanded this by including GPU assembly instruction sequences and deploying recurrent neural networks to capture dependency features. Braun et al. [25] proposed a simple model for swift predictions across GPUs relying solely on PTX instruction statistics. Fan et al. [26] developed DVFS-aware static models based on a vector of 10 instruction types. However, these methods achieved high power prediction accuracy, but their execution time predictions were lacking due to the absence of hardware runtime information, such as cache hit rate or register spilling.

IV. METHODOLOGY

A. Overview

Figure 1 outlines our DSO optimizer framework. Initially, we present a parameterized optimization model, referred to as the “GPU DVFS Model” rectangle, which incorporates the effects of GPU DVFS and balances the tradeoff between

performance and energy efficiency. Our model aims to enhance energy efficiency by considering these factors. To accurately and efficiently predict the model parameters, we employ machine learning techniques that leverage hardware status information from DCGM and GPU kernel details from the PTX parser. By utilizing these inputs, we can make informed predictions about the model parameters. Once these parameters are determined, we can theoretically derive the optimal DVFS configuration. For implementing the DVFS configuration, the GPU DVFS controller utilizes the APIs provided by NVML [27]. These APIs enable us to set the desired voltage and frequency targets, allowing us to implement the optimal DVFS configuration based on the calculated parameters.

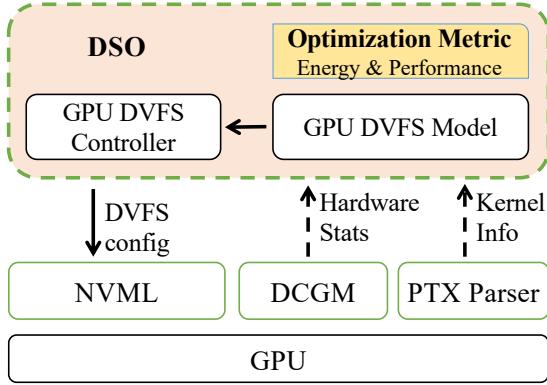


Fig. 1: DSO Workflow.

B. Problem Formulation

Previous studies have showcased the effectiveness of statistical models trained using dynamic and static GPU features to accurately represent performance and power data samples. These models have achieved a remarkable level of accuracy and confidence. However, compared to learning-based approaches, the utilization of parameterized models provides the advantage of interpreting the unique characteristics of GPU hardware and comprehending the impact of DVFS on performance and power. Expanding upon the research presented in [2], [28], we adopt a similar approach to model the runtime power of the GPU, as shown in Equation (1).

$$P(V^c, f^c, f^m) = \underbrace{(P^0 + \kappa V^c)}_{P_{static}} + \underbrace{(\gamma f^m + c(V^c)^2 f^c)}_{P_{dynamic}} \quad (1)$$

V^c , f^c , and f^m represent the GPU supply core voltage, GPU core frequency, and GPU memory frequency, respectively. The power is consist of two parts, the static part P_{static} and the dynamic part $P_{dynamic}$. P_{static} includes P^0 , which denotes the constant power consumption of the GPU system that is not related to GPU voltage/frequency scaling, and κV , which denotes the power that maintains the supply voltage for executing the GPU application. κ denotes the coefficient related to the hardware characteristics, such as the number of transistors in the chip design and the leakage current for a single transistor. As for the dynamic part, the coefficients γ

and c are constant values that rely on both the hardware characteristics and the specific application being considered. These coefficients indicate the sensitivity of power consumption to memory frequency scaling and core voltage/frequency scaling, as explained in [23].

Compared to power modeling, performance modeling of GPU DVFS is rather complex [12], [29]. Inspired by the DVFS-aware roofline observations in [12], we innovatively design a piecewise mathematical model with a concise form to simplify the subsequent analysis of GPU energy conservation. We formulate the performance function $T(f^c, f^m)$ of a GPU-accelerated application as shown in Eq. (2). t^0 represents the constant component in GPU application execution time. α is a constant factor that indicates the sensitivity of this application to GPU memory frequency scaling, and β is a constant factor that indicates the sensitivity to GPU core frequency scaling. With t^0 , α and β set to different values, the model is capable of simulating the various DVFS effects of a variety of applications. Our experiments on real GPU applications indicate that this time model effectively captures the performance effects of DVFS on all the tested applications. It provides a coherent explanation for the observed performance variations and accurately represents the impact of DVFS across different application scenarios.

$$T(f^c, f^m) = t^0 + \max\left(\frac{\alpha}{f^m}, \frac{\beta}{f^c}\right) \quad (2)$$

Notice that f^c and V^c are correlated. For a fixed V^c , the maximum core frequency (f_{max}^c) is determined by V^c . We apply the function in [30] to denote this relationship: $f^c \leq g_1(V^c) = \sqrt{(V^c - \kappa)/2} + \kappa$, and κ is the function parameter.

With the above models, the GPU energy (E_J) consumed to process one task is the product of the runtime power and the execution time, as shown in Eq. (3).

$$E = \underbrace{(P^0 + \kappa V^c + \gamma f^m + c(V^c)^2 f^c)}_{P(V^c, f^c, f^m)} \times \underbrace{(t^0 + \max\left(\frac{\alpha}{f^m}, \frac{\beta}{f^c}\right))}_{T(f^c)} \quad (3)$$

We propose a simple objective cost function to tradeoff the performance and energy consumption as Eq. (4).

$$\begin{aligned} C(V^c, f^c, f^m) &= \eta E + (1 - \eta) P_{max} T \\ &= \eta P T + (1 - \eta) P_{max} T \\ &= (\eta P + (1 - \eta) P_{max}) T \end{aligned} \quad (4)$$

Here η is the parameter specified by the user to express the relative importance of energy efficiency and training performance (throughput). When $\eta = 0$, we are only optimizing for time consumption, whereas when $\eta = 1$, we are only optimizing for energy consumption. P_{max} is the maximum power limit supported by the GPU, a constant introduced to unify the units of measure in the cost metric.

The parameters to be determined are (P^0, κ, γ, c) related to power and (t^0, α, β) related to performance. While they can be fitted with the data points sampled from different DVFS

settings, in the paper, we attempt to directly predict them by the DCGM metrics and static code information, which do not need to pre-execute the target GPU application with different DVFS settings.

C. Best Configuration for Target Cost

As a first step, we consider the following how to derive the solution to achieve the best energy efficiency modeled by Eq. (4). Eq. (5) shows the mathematical formulation of the problem. Notice that V^c and f^c are correlated variables, and f^c is upper bounded by a function of V^c , denoted by $g_1(V^c)$.

$$\begin{aligned} \text{argmin } C = & \text{argmin}_{V^c} \{ (\eta(P^0 + \kappa V^c + \gamma f^m + c(V^c)^2 f^c) \\ & + (1 - \eta)P_{max}) \times (t^0 + \max(\frac{\alpha}{f^m}, \frac{\beta}{f^c})) \} \\ \text{s.t. } & V_{min}^c \leq V^c \leq V_{max}^c, f_{min}^m \leq f^m \leq f_{max}^m, \\ & f_{min}^c \leq f^c \leq g(V^c) \end{aligned} \quad (5)$$

Theorem 1. With a fixed memory frequency, the cost function of a GPU kernel is minimum when the GPU core frequency is maximum corresponding to the GPU core voltage, and $f^c \leq \frac{\beta}{\alpha} f^m$, i.e.,

$$C_{min}(f^m) = \arg \min_{V^c} C(V^c, g(V^c), f^m).$$

Proof. We firstly discuss the case of $f^c < \frac{\beta}{\alpha} f^m$ (equivalent to $\frac{\alpha}{f^m} < \frac{\beta}{f^c}$), which indicates that the kernel is compute-bound. The cost function then becomes $C = (\eta(P^0 + \kappa V^c + \gamma f^m + c(V^c)^2 f^c) + (1 - \eta)P_{max}) \times (t^0 + \frac{\beta}{f^c})$. We obtain the first-order partial derivatives as: $\frac{\partial C}{\partial V^c} = \eta(\kappa + 2cV^c f^c)(t^0 + \frac{\beta}{f^c})$ and $\frac{\partial C}{\partial f^c} = c\eta(V^c)^2(t^0 + \frac{\beta}{f^c}) - \beta\eta(P^0 + \kappa V^c + \gamma f^m + c(V^c)^2 f^c) + (1 - \eta)P_{max}$. Because $\frac{\partial C}{\partial V^c} > 0$, C cannot attain its minimum on the interior of the domain, and E_J is a monotonically increasing function of V^c . The minimum is on the boundary of $g(V^c)$.

We then discuss the case of $f^c \geq \frac{\beta}{\alpha} f^m$, which indicates that the kernel is memory-bound. $T(f^c)$ is reduced to be $t^0 + \frac{\beta}{f^m}$. As P is a monotonically increasing function of f^c , the minimum of C is achieved when $f^c = \frac{\beta}{\alpha} f^m$.

To be concluded, f^c can be eliminated such that finding the minimum of C is only related to V^c , and the condition of getting the minimum C is $f^c \leq \frac{\beta}{\alpha} f^m$. \square

Theorem 1 transforms a three-variable optimization problem into a two-variable optimization problem. It implies that when we scale the GPU core alone to conserve energy, we only need to find an appropriate core voltage and set the core frequency to the largest allowed value.

We then consider GPU memory frequency scaling alone. If the core voltage and frequency settings are fixed as V_o^c and f_o^c , we can easily compute the optimal memory frequency by setting $\frac{\partial C}{\partial f^c} = 0$. We denote it as f_o^m . Since the minimum C is obtained when $f^c \leq \frac{\beta}{\alpha} f^m$, the time model can be simplified to $(t^0 + \frac{\beta}{f^c})$, which eases the calculation of f_o^m .

Based on the above analysis, the original three-variable problem is transformed into an one-variable optimization problem. Reducing the problem dimension is vital to speeding up the computation. Since the GPU voltage usually has a narrow range, we can conduct a grid search on it and derive the optimal core and memory frequencies in practice.

D. Modeling the performance and power of GPU DVFS

Theorem 1 allows the solution of Eq. (5) via a single-variable optimization problem. The next step is estimating parameter values in the cost model for a GPU application. Unlike prior work [8], [10], [24], [25] directing predictions towards performance and power or the scaling ratio compared to the default DVFS setting, we suggest estimating model parameters using both DCGM metrics (dynamic information) and PTX instructions (static information) with a machine learning algorithm.

1) *Feature Processing:* We selected eight key metrics from the DCGM profiling tool closely tied to GPU kernel activities (Table I), the values of which range between 0 and 1. As discussed in [12], these metrics encompass the crucial factors essential for GPU performance modeling.

Regarding PTX instructions, we consider three categories: instruction type, data type, and memory space (Table II). These include instruction types defined in the PTX ISA, basic instruction operand types, and all the types in the GPU memory hierarchy. We parse the PTX source code to count each instruction type and normalize each value by their category's total instructions.

TABLE I: DCGM Features used in our GPU DVFS model

Name	Descriptions
SMACT	The ratio of cycles an SM has at least one warp assigned
SMOCC	The ratio of number of warps resident on an SM
TENSO	The ratio of cycles the any tensor pipe is active
DRAM	The ratio of cycles the device memory interface is active sending or receiving data
FP64A	Ratio of cycles the fp64 pipe is active
FP32A	Ratio of cycles the fp32 pipe is active
FP16A	Ratio of cycles the fp16 pipe is active
INTAC	Ratio of cycles the integer pipe is active

TABLE II: PTX Features used in our GPU DVFS model

Type	Count	Examples
instruction	101	add, sub, fma, bra
data	17	.u16, .f32, .f64
memory	8	.global, .shared, .local, .reg

2) *Training Scheme:* Contrasting previous methods [8], [12], [24], [26] that predict absolute values or scaling factors, we propose to estimate the parameters in Eq. (3), which can be used by Theorem 1 to derive the best configuration. This approach leverages efficient machine learning algorithms like Randomized Trees Regression [25] and shallow neural networks [24], with an explicit formulation simplifying the learning process. We utilize a multi-layer perceptron for estimating the energy model parameters, consisting of five layers

including one input, three hidden layers (using a sigmoid activation function) with empirically set neurons (100, 50, 25), and one output layer. Hyper-parameters like batch size and learning rate are optimized with grid search and overfitting is prevented through three-fold cross-validation. Practically, parameters in Eq. (3) are obtained by collecting data samples under all frequency settings for each GPU. Linear regression is then used to fit the power model and piecewise linear regression for the performance model, with the average regression absolute percentage error within 2%. Our neural network is trained to estimate these model parameters, providing accurate predictions for realistic benchmarks.

V. EXPERIMENTS

A. Experimental Setup

We validate the proposed DSO model on the Tesla V100 GPU. Experiments are performed on a Linux Ubuntu 18.04, with CUDA 11.5 and Nvidia Driver v515. Notice that the memory frequency of V100 is fixed since they cannot be tuned under our experimental environments. We change the GPU operating DVFS settings by *nvidia-smi* [15] with the flag “-lfc” (for core frequency) and “-lmc” (for memory frequency). In our experiments, we tune no less than 10 frequency options, from 705 to 1380 MHz, to collect sufficient data samples to train the model.

To obtain all the data samples to train the machine learning algorithms for predicting the GPU DVFS model parameters, we execute each benchmark in [16], [17] on the GPUs at all the available frequency configurations. By tuning the ratio of different instructions in each application, we obtain totally 138 GPU benchmarks of different operational intensity values. After that, the accuracy of the estimated models is tested on 20 realistic benchmarks from CUDA SDK 11.5 and Rodinia. These benchmark applications cover a wide range of execution patterns, such as DRAM intensive, L2 cache intensive, shared memory intensive and computation intensive. Notice that the testing set is not used to train the models. Our model can perform well generalization on unseen GPU applications.

To build the DSO model, we obtain the necessary PTX instructions, DCGM metrics, and power samples for each benchmark. The “-ptx” flag of the nvcc compiler is used for PTX instructions, adjusting the PTX ISA version for different GPU architectures with “-gencode=arch=compute_XX,code=compute_XX” (in the case for the Tesla V100, which has a compute capability of 7.0, “XX” is 70). To gather DCGM metrics, a daemon thread runs “*dcmi dmon -e \${METRIC_LIST}*” in loop during benchmark execution. A second daemon thread using NVML [27] gathers power consumption data. Each benchmark’s average power consumption is calculated from these samples. These efficient, non-intrusive methods demonstrate DSO’s practical flexibility and efficiency.

B. Model Accuracy

Table III showcases our DSO in comparison to other studies, highlighting the modeling overhead and accuracy. Relying

solely on static PTX code analysis proves challenging for execution time prediction, highlighting the importance of dynamic tools like nvprof and DCGM for accurate estimates. The DSO integrates DCGM for practical, high-quality, and low overhead estimation. The performance and power estimation accuracy of DSO on the V100 GPU are manifested in Figure 2. To emphasize the benefit of fusing dynamic and static information, DSO is compared with methods using either PTX features or DCGM metrics alone. DSO consistently yields low prediction errors for execution time (0.5%-9.5%) and power consumption (1.3%-7%). It outperforms the others with an average MAPE of 4.6% (vs. PTX’s 7.8% and DCGM’s 5.7%) for performance and 4.9% for power (vs. PTX’s 8.3% and DCGM’s 6.9%). It’s observed that combining PTX and DCGM improves the accuracy drastically for several GPU applications, confirming the utility of runtime utilization information for capturing DVFS effects. This underlines the rationale behind DSO’s design, using PTX features as a supplement to DCGM metrics for enhanced prediction.

TABLE III: Comparison against the existing studies on Volta GPUs. The MAPE is measured for time (T) and power (P).

Paper	Input Source	Modeling Overhead	Mean Average Precision Error (MAPE)
J. Guerreiro [24]	PTX	low	16.7(T)/6.7(P)
Q. Wang [12]	nvprof	high	8.8%(T)/4.2%(P)
L. Braun [25]	PTX	low	11%(T)/2.3%(P)*
DSO (ours)	DCGM & PTX	low	4.6%(T)/4.9%(P)

*168 realistic kernels are used for cross-validation model training, which results in much better accuracy than others.

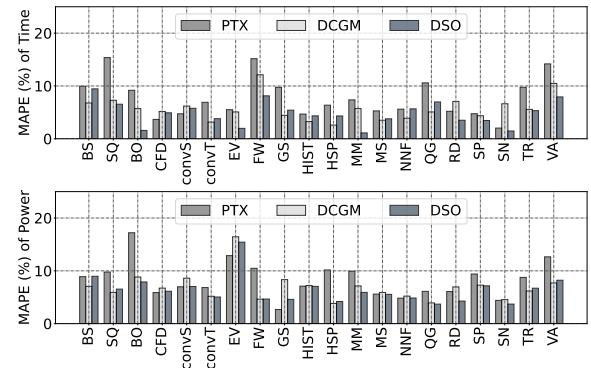


Fig. 2: The prediction accuracy on Tesla V100.

C. Energy Efficiency

Once predictive model parameters are garnered, the optimal DVFS configuration for minimal energy consumption is derived (Section IV-C). Figure 3 displays these results as average values over 20 real applications. We examined the impact of varying η values, which signal the preference level for energy efficiency. An η of 1 indicates maximum preference for low energy consumption. As η increases, execution time expands, but energy consumption decreases. This results from additional opportunities for energy efficiency allowed by a higher η . Within performance loss limits (e.g., 5%), our DSO offers a

suitable η selection according to these requirements. For the V100 GPU architecture, we advise using $\eta = 0.8$, conserving energy consumption by approximately 19%.

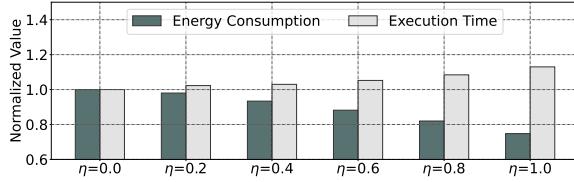


Fig. 3: The energy and performance compared to the default setting on V100.

VI. CONCLUSION

We introduce DSO, an innovative framework for modeling and optimizing GPU performance, power, and energy with DVFS. DSO uses a parameterized theoretical model considering DVFS effects to simultaneously optimize performance and energy efficiency. Combining static PTX and dynamic DCGM information, DSO accurately estimates model parameters, showing improved prediction accuracy over using either source independently. Tested on the Volta GPU, DSO models yield accurate results for unencountered real GPU applications. Furthermore, DSO facilitates balancing performance and energy efficiency. Leveraging DSO’s optimal configurations can enhance GPU energy efficiency by approximately 20% with no more than a 5% performance loss.

ACKNOWLEDGMENTS

This research was supported by the National Natural Science Foundation of China (No. 62302126), the Shenzhen Science and Technology Program (No. RCBS20221008093125065, No. JSGGKQTD20221101115655027) and the Peng Cheng Laboratory Project (Grant PCL2021A13).

REFERENCES

- [1] J. You, J.-W. Chung, and M. Chowdhury, “Zeus: Understanding and optimizing gpu energy consumption of dnn training,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023.
- [2] Q. Wang, X. Mei, H. Liu, Y.-W. Leung, Z. Li, and X. Chu, “Energy-aware non-preemptive task scheduling with deadline constraint in dvfs-enabled heterogeneous clusters,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4083–4099, 2022.
- [3] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, “Characterization and prediction of deep learning workloads in large-scale gpu datacenters,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’21, 2021.
- [4] L. Floridi and M. Chiratti, “Gpt-3: Its nature, scope, limits, and consequences,” *Minds and Machines*, vol. 30, no. 4, pp. 681–694, 2020.
- [5] R. A. Bridges, N. Imam, and T. M. Mintz, “Understanding gpu power: A survey of profiling, modeling, and simulation methods,” *ACM Computing Surveys*, vol. 49, no. 3, p. 27, Sep. 2016.
- [6] X. Mei, Q. Wang, and X. Chu, “A survey and measurement study of gpu dvfs on energy conservation,” *Digital Communications and Networks*, vol. 3, no. 2, pp. 89 – 100, 2017.
- [7] Z. Tang, Y. Wang, Q. Wang, and X. Chu, “The impact of gpu dvfs on the energy and performance of deep learning: An empirical study,” in *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, ser. e-Energy ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 315–325.
- [8] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, “Gpgpu performance and power estimation using machine learning,” in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 564–576.
- [9] Q. Wang and X. Chu, “Gpgpu performance estimation with core and memory frequency scaling,” in *Proceedings of the 24th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2018, pp. 417–424.
- [10] J. a. Guerreiro, A. Ilic, N. Roma, and P. Tomás, “Dvfs-aware application classification to improve gpgpus energy efficiency,” *Parallel Computing*, vol. 83, p. 93–117, apr 2019.
- [11] Y. Huang, B. Guo, and Y. Shen, “Gpu energy consumption optimization with a global-based neural network method,” *IEEE Access*, vol. 7, pp. 64 303–64 314, 2019.
- [12] Q. Wang and X. Chu, “Gpgpu performance estimation with core and memory frequency scaling,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2865–2881, 2020.
- [13] NVIDIA, “Data Center GPU Manager (DCGM),” [Online] <https://developer.nvidia.com/dcgm>.
- [14] ———, “NVIDIA Profiler,” [Online] <http://docs.nvidia.com/cuda/profiler-users-guide>.
- [15] ———, “NVIDIA System Management Interface (nvidia-smi),” [Online] <https://developer.nvidia.com/nvidia-system-management-interface>.
- [16] E. Konstantinidis and Y. Cotronis, “A quantitative performance evaluation of fast on-chip memories of gpus,” in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, 2016, pp. 448–455.
- [17] ———, “A quantitative roofline model for gpu kernel performance estimation using micro-benchmarks and hardware metric profiling,” *Journal of Parallel and Distributed Computing*, vol. 107, pp. 37–56, 2017.
- [18] X. Mei and X. Chu, “Dissecting GPU memory hierarchy through microbenchmarking,” *IEEE TPDS*, vol. 28, no. 1, pp. 72–86, Jan 2017.
- [19] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, “Gpgpu power modeling for multi-domain voltage-frequency scaling,” in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 789–800.
- [20] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, “Modeling and decoupling the gpu power consumption for cross-domain dvfs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 11, pp. 2494–2506, 2019.
- [21] Q. Wang, C. Liu, and X. Chu, “Gpgpu performance estimation for frequency scaling using cross-benchmarking,” in *Proceedings of the 13th Annual Workshop on General Purpose Processing Using Graphics Processing Unit*, ser. GPGPU ’20, 2020, p. 31–40.
- [22] S. Hong and H. Kim, “An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA ’09, 2009, p. 152–163.
- [23] ———, “An integrated gpu power and performance model,” in *Annual International Symposium on Computer Architecture*, vol. 38, no. 3. ACM, 2010, pp. 280–289.
- [24] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, “Gpu static modeling using ptx and deep structured learning,” *IEEE Access*, vol. 7, pp. 159 150–159 161, 2019.
- [25] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, “A simple model for portable and fast prediction of execution time and power consumption of gpu kernels,” *ACM Transactions Architecture and Code Optimization*, vol. 18, no. 1, dec 2021.
- [26] K. Fan, B. Cosenza, and B. Juurlink, “Predictable gpus frequency scaling for energy and performance,” in *Proceedings of the 48th International Conference on Parallel Processing*, ser. ICPP 2019, 2019.
- [27] NVIDIA, “NVIDIA Management Library ,” [Online] <https://developer.nvidia.com/nvidia-management-library-nvml>.
- [28] J. Butts and G. Sohi, “A static power model for architects,” in *Proceedings 33rd Annual IEEE/ACM International Symposium on Microarchitecture. MICRO-33 2000*, 2000, pp. 191–201.
- [29] R. Nath and D. Tullsen, “The CRISP performance model for dynamic voltage and frequency scaling in a GPGPU,” in *Proceedings of the 48th MICRO*, 2015.
- [30] X. Mei, L. S. Yung, K. Zhao, and X. Chu, “A measurement study of gpu dvfs on energy conservation,” in *Proceedings of the Workshop on Power-Aware Computing and Systems*, ser. HotPower ’13, 2013.