SSMDVFS: Microsecond-Scale DVFS on GPGPUs with Supervised and Self-calibrated ML

Minqing Sun
The University of Michigan-Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University
Shanghai, China
sun-minqing@sjtu.edu.cn

Ruiqi Sun
The University of Michigan-Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University
Shanghai, China
srq916@sjtu.edu.cn

Yingtao Shen
The University of Michigan-Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University
Shanghai, China
doctorcoal@sjtu.edu.cn

Wei Yan
The Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China
yanwei@ict.ac.cn

Qinfen Hao
The Institute of Computing Technology
Chinese Academy of Sciences
Beijing, China
haoqinfen@ict.ac.cn

An Zou
The University of Michigan-Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University
Shanghai, China
an.zou@sjtu.edu.cn

***Abstract*—Over the past decade, as GPUs have evolved to achieve higher computational performance, their power density has also accelerated. Consequently, improving energy efficiency and reducing power consumption has become critically important. Dynamic voltage and frequency scaling (DVFS) is an effective technique for enhancing energy efficiency. With the advent of integrated voltage regulators, DVFS can now operate on microsecond (µs) timescales. However, developing a practical and effective strategy to guide rapid DVFS remains a significant challenge. This paper proposes a supervised and self-calibrated machine learning framework (SSMDVFS) to guide microsecond-scale GPU voltage and frequency scaling. This framework features an end-to-end design that encompasses data generation, neural network model design, training, compression, and final runtime calibration. Unlike analytical models, which struggle to accurately represent GPU architectures, and reinforcement learning approaches, which can be challenging to converge during runtime, the SSMDVFS offers a practical solution for guiding microsecond-scale voltage and frequency scaling. Experimental results demonstrate that the proposed framework improves energy-delay product (EDP) by 11.09% and outperforms analytical models and reinforcement learning approaches by 13.17% and 36.80%, respectively.**

I. Introduction

Dynamic Voltage and Frequency Scaling (DVFS) is a key technique for improving performance and energy efficiency in Graphics Processing Units (GPUs). As GPUs are increasingly used in high-performance computing, AI, and data-intensive applications, their energy consumption and heat dissipation are major concerns. DVFS addresses these by adjusting voltage and frequency based on workload demands, reducing power consumption and thermal output during low loads without sacrificing performance during high-demand tasks. However, applying DVFS to GPUs presents two challenges: first, the complex execution dynamics, including parallelism and overlapping computation and memory operations, complicate workload assessment and performance prediction; second, with advances in integrated voltage regulators enabling fine-grained, microsecond-scale DVFS, predicting workload fluctuations at such fine resolutions becomes more difficult, complicating optimization of power and performance.

Corresponding authors: Wei Yan (yanwei@ict.ac.cn), Qinfen Hao (haoqinfen@ict.ac.cn), An Zou (an.zou@sjtu.edu.cn).

Analytical methods and machine learning-based techniques, especially reinforcement learning (RL), are key for optimizing dynamic voltage and frequency scaling (DVFS). Both face the tradeoff between performance and power efficiency: reducing frequency saves power but can increase execution time, potentially raising total energy consumption and degrading performance for time-sensitive applications. The goal is to optimize DVFS by selecting the best voltage/frequency (V/f) settings for each program phase while minimizing power.

However, each approach has limitations. Analytical methods often oversimplify the complex execution behavior of GPUs, such as parallelism and warp transitions. Despite advancements incorporating frequency sensitivity and iterative workload characteristics [1], they still struggle to find optimal solutions for DVFS on GPUs. On the other hand, reinforcement learning (RL) offers a more flexible and adaptive solution by modeling the complex behavior of GPUs. In theory, a well-trained RL model with sufficient data can predict and optimize GPU performance efficiently. However, the high computational cost and time required to train RL models, coupled with the challenge of balancing exploration and exploitation, limit their practical use for fine-grained DVFS in fast-evolving environments like GPUs. RL methods typically require extensive iterations to converge on an optimal strategy, introducing delays and resource overhead. Even hierarchical RL-based models [2], designed to combine coarse-grained policy updates with fine-grained DVFS predictions, suffer from slow model update times, often taking hundreds of microseconds to make the first well-founded decision, leading to massive performance loss.

To overcome these challenges, we propose a supervised and self-calibrated machine learning framework (SSMDVFS) to

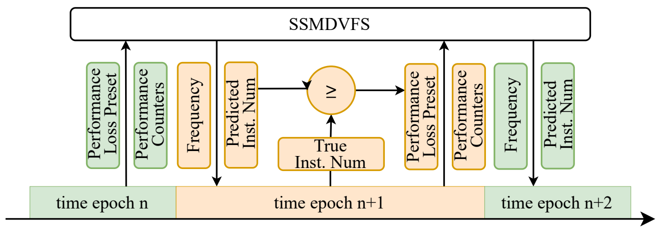


Fig. 1. SSMDVFS Workflow Diagram

guide microsecond-scale GPU voltage and frequency scaling. Instead of relying on direct optimization problem modeling, we utilize a custom data generation process to train a neural network offline. This approach enables real-time inference during program execution, allowing the selection of the lowest possible frequency that minimizes energy consumption while keeping performance loss within acceptable limits. Our contributions are as follows:

- We propose a novel data generation technique that transforms the DVFS optimization problem into a supervised learning task, using real-time performance counters to construct a dataset that captures power-performance trade-offs. This enables the model to learn optimal V/f settings across varying workloads, enhancing energy efficiency and performance.
- We develop a high-performance DVFS prediction module based on a multi-layer perceptron (MLP), achieving fine-grained tuning at a 10-microsecond resolution. This improves adjustment granularity, stabilizes system reactions, and maximizes power savings.
- We introduce an adaptive, self-calibrated tuning mechanism that continuously monitors system performance, dynamically adjusting the preset performance loss. This ensures that performance loss remains within a predefined threshold, minimizing manual intervention.

II. Problem Formulation

In DVFS (Dynamic Voltage and Frequency Scaling) decision-making, both reinforcement learning (RL) [2] and analytical methods [1], [3] typically involve a well-defined optimization process. Analytical methods focus on constructing an objective function, identifying key variables, and using latency and power comsuption as constraints to find the optimal DVFS decisions. RL methods, on the other hand, explore action-state spaces and define reward functions based on latency and power comsuption, learning to maximize rewards to make DVFS decisions. A close examination of these approaches reveals inherent limitations. Analytical methods often result in low-complexity models that fail to capture the full complexity of GPU operations, while RL models, although more sophisticated, suffer from online learning and long converge times, may fail when applied to fine-grained DVFS decisions, especially at microsecond timescales.

Given these challenges, we propose a more practical supervised learning based microsecond-scale framework SSMDVFS that can effectively guide the GPU voltage and frequency scaling at microsecond timescales.

We start by establishing a key assumption for the optimization process: during GPU operation, the program runs in a linear, forward-moving manner, meaning predictions must be based on outputs from previous time periods. This implies that the operating characteristics at the current DVFS decision point can be predicted from the characteristics of the previous time period. Since these characteristics are closely tied to power efficiency, parameters obtained after the previous period can directly inform the current DVFS decision. Based on this assumption, we propose a classification model Which is called Decision-maker to directly generate DVFS decisions. The model's classification criterion is to select the minimum frequency that satisfies a given performance loss preset. We also introduce a regression model which is called Calibrator to assess whether the chosen frequency meets the performance loss preset and to adjust the threshold for subsequent periods, ensuring that performance loss remains within acceptable limits.

The workflow of SSMDVFS framework during a processor core's runtime is illustrated in Fig. 1. In the current DVFS time epoch n, the processor core runs at a clock frequency f, collecting various performance counters at the end of the epoch. These values, along with the performance loss preset, are fed into Decision-maker, which outputs the frequency for the next time epoch $n+1$. Simultaneously, Calibrator takes Decision-maker's inputs and output frequency for time epoch $n+1$, predicting the total number of instructions to be executed in the upcoming period. At the end of the next time epoch $n+1$, the actual instruction number is compared with the predicted number from Calibrator. This comparison informs the adjustment of the performance loss preset for the following time epoch $n+2$. If the predicted number exceeds the actual number, it indicates that the core is running too slowly at the current frequency, potentially exceeding the performance loss preset. In response, we reduce the preset slightly, guiding Decision-maker toward predicting a higher execution speed. This iterative process of desicion-making and calibrating allows us to effectively resolve the DVFS optimization problem using straightforward classification and regression models.

III. Prediction Model

A. Data Generation

The objective of the DVFS optimization problem is to select the minimal operating frequency that minimizes energy consumption while adhering to a maximum allowable performance loss. In a classification context, this corresponds to input a set of features, including the performance loss preset, into a neural network that classifies the inputs into the minimal voltage/frequency (V/f) operating point that satisfies the constraint.

To this end, we have devised the following data generation methodology. We use over 20 benchmarks from Rodinia [4], Parboil [5] and PolyBench [6]. These benchmarks are executed under Nvidia GeForce GTX Titan X architecture using GPGPU-Sim simulator [7] with 6 operating points of V/f

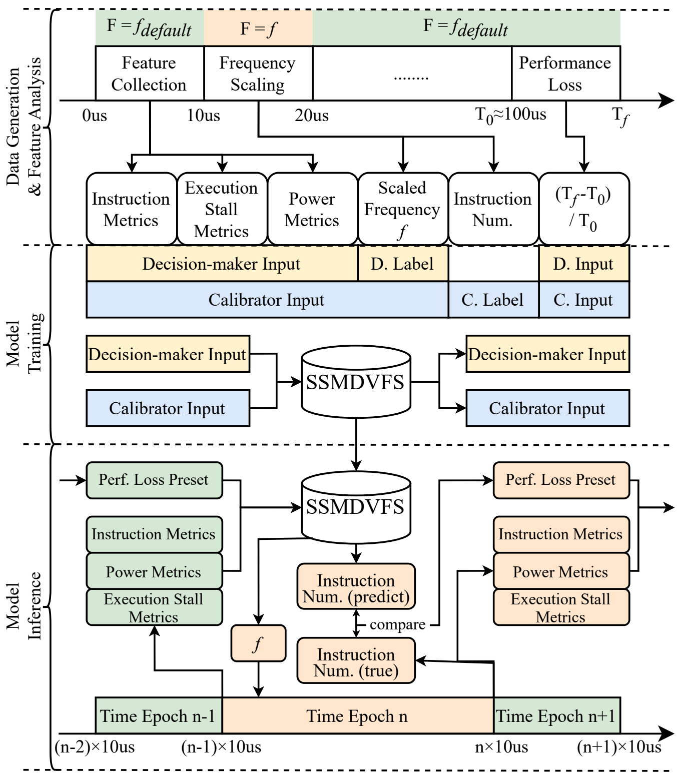


Fig. 2. Overall Process of SSMDVFS Build-up

selections, ranging from default V/f operating point (1.155 V, 1165 MHz) to minimal frequency (1.0 V, 683 MHz) [8].

As shown in Fig. 2 data generation part, initially, a benchmark is executed at the default V/f operating point. For about every 100 µs, a breakpoint is established, and the total execution time T_0 is recorded, marking one data point generation cycle. Subsequently, the program is executed for 10 µs at the default V/f operating point, referred to as the feature collection window. In the following 10 µs, the operating frequency is varied across the available V/f operating points, referred to as the frequency scaling window. In our case, there are six V/f operating points, including the default V/f operating point, requiring the program to be executed six times. After each frequency scaling window, the total instruction number within the 10 µs period is recorded. The operating frequency is then reverted to the default value, and the program continues until the breakpoint, ensuring the total workload remains constant. At this point, we measure the total execution time T_f for all six V/f operating points. The performance loss is then computed as $(T_f - T_0)/T_0$, reflecting the impact of frequency changes on execution performance.

It is crucial to note that data collection spans 100 µs, not only the 20 µs containing the feature collection window and frequency change window. This is due to the nonlinear execution dynamics of GPUs, where the effects of frequency changes may manifest over an extended execution process. For instance, a warp of instructions stalled due to a frequency change may not resume for several time epochs. If the collection period were limited to 20 µs, the calculated execution time might erroneously exclude delayed effects, leading to inaccuracies. Therefore, a 100-microsecond collection period is chosen to mitigate errors in performance loss calculation.

B. Feature Analysis

The data generation process yields several types of data: performance counters from the feature collection window, total instruction number during the frequency scaling window, frequency applied to the frequency scaling window, and the resulting performance loss.

We analyze performance counters obtained during the feature collection window, which can be categorized into three types: instruction metrics, execution stall metrics, and power consumption metrics. Instruction metrics include the total instruction count and counts of various instruction types. Execution stall metrics cover control hazards, memory hazards, and cache hit/miss rates. Power consumption metrics represent the total power consumed by the processor cores during the epoch.

Given our goal to select the lowest frequency that satisfies the performance loss preset while minimizing power consumption, we classify the performance counters into direct and indirect features. Direct features include power consumption metrics, as they directly correlate with the desired minimization of current time epoch power consumption. Lower power in the prior time epoch may indicate under-utilization of core computational resources, suggesting a frequency reduction. However, relying solely on direct features can lead to sub-optimal frequency selection. For example, low power may result from either low computational instruction density or increased stalls in pipelines, each requiring different frequency adjustments. Indirect features, such as instruction and stall metrics, provide context to the power metrics, helping to predict processor behavior in the subsequent epoch and indirectly guide optimal frequency selection.

In our initial experiments, we collected 47 performance counters. In Section IV, we will refine the feature set based on these three categories of performance counters.

C. Model Training and Inference

Model Training Logic: As shown in Fig. 2 model training part, the input for the Decision-maker consists of the performance counters collected during the feature collection window and the actual performance loss calculated after adjusting the V/f operating point during the frequency change window. The label for each data point is the frequency level applied during the frequency change window. For the Calibrator, the input includes both the features used for decision-making and the label from the Decision-maker, with the output being the total number of instructions executed during the frequency change window.

Model Inference Logic: In real-time inference, every 10 µs, we first compare the total instructions number in the current epoch with the predictions from the Calibrator. This comparison helps determine whether the performance loss preset should be adjusted. Subsequently, the performance counters collected during the current epoch, along with the calibrated performance loss preset, are used as inputs to determine the frequency decision for the next 10-microsecond interval. Additionally, the current epoch's performance counters, combined with the originally set performance loss preset and the frequency decision from the Decision-maker, serve as inputs to the Calibrator to predict the expected total instructions number of the subsequent time epoch.

A key point to consider is the setting of the performance loss preset. During training, the performance loss input reflects the actual performance loss incurred due to the frequency change, indicating that this level of performance loss was achieved by switching to the corresponding frequency. During inference, the performance loss preset becomes a manually set input, which can be adjusted by Calibrator using regression model to help maintain the desired performance loss preset. However, the Calibrator consistently uses the originally set performance loss preset, implying that under the initial performance loss expectation, it predicts the expected total instructions executed in the following time epoch.

D. *Model Architecture*

We developed a lightweight neural network (NN) to perform both classification and regression tasks. Given the substantial overlap in input features between the two models, they are combined into a single network, including five fully connected layers dedicated to Decision-maker output, followed by four fully connected layers for Calibrator output. Each layer is activated using the ReLU function, with the initial number of neurons per layer set to 20.

IV. Model Combination and Compression

A. *Feature Selection*

In Section III, all features derived from performance counters are categorized into instruction information, execution stall information, and power information. Power information serves as a direct feature, while the other two are indirect features that help interpret the execution dynamics of the processor. We employed Recursive Feature Elimination (RFE) [9] to refine the two indirect feature categories. RFE works by measuring the impact on model accuracy when a specific feature's values are shuffled. A significant decrease in accuracy indicates the importance of that feature to the model. Refer to Table I, three features were selected from the original set of 47: instructions per core (instruction information), memory hazards, memory hazards from other than load and L1 cache read misses (execution stall information). The refined model resulted in only a 0.48% reduction in classification accuracy and a 0.65% increase in regression MAPE, compared to using all 47 features.

B. *Layer-wise Compression*

The initial network architecture consists of nine fully connected layers, each with 20 neurons. Five layers are dedicated to Decision-maker, and the remaining four are used for Calibrator. However, this architecture can be significantly compressed. We first explored the relationship between FLOPs (floating point operations) and model accuracy by adjusting the number of layers and hidden neurons per layer. In Fig. 3, it was observed that reducing the network's FLOPs below a certain threshold led to a sharp drop in accuracy. The original model had approximately 6960 FLOPs, which could be compressed down to 912 FLOPs. To minimize the model's complexity while maintaining accuracy, we selected the architecture with the fewest layers that did not massively sacrifice accuracy. The compressed model consists of three fully connected layers for Decision-maker and two layers for Calibrator, making a total of five layers, each with 12 hidden neurons.

TABLE I
Metrics and Performance Counters

Metrics	**Performance Counters**
Instruction Metrics	IPC (Instruction per Core)
Power Metrics	PPC (Power per Core)
Execution Stall Metrics	MH (Memory Hazard)
	MH\L (Memory Hazard From other than Load)
	L1CRM (L1 Cache Read Miss)

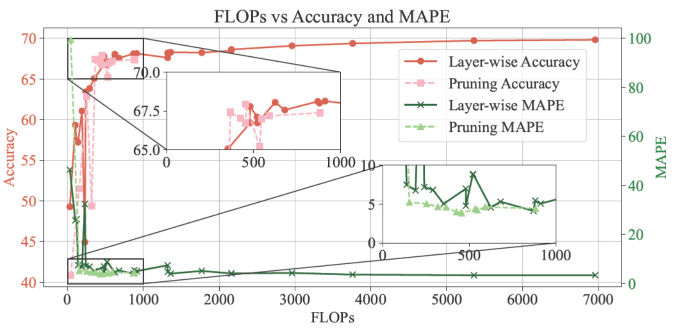


Fig. 3. FLOPs vs Accuracy and MAPE for Layer-wise Compression and Pruning

C. *Pruning*

Pruning was applied to further reduce model size by eliminating unnecessary weights and neurons. The pruning process is conducted in two stages. First, fine-grained pruning zeros out a portion $x_1 \in [0, 1]$ of the smallest weights, reducing computational cost without affecting accuracy. Second, neuron-level pruning is performed (analogous to vector-level pruning in convolutional neural networks). If a neuron's weight vector contains a high proportion $x_2 \in [0, 1]$ of zero-valued weights after the first step, the neuron is considered redundant and removed entirely. This adaptive method adjusts the number of neurons per layer, complementing the layer-wise compression performed earlier. By tuning the pruning parameters, we obtained models of various sizes with corresponding FLOPs. As demonstrated in Fig. 3, when total FLOPs fall below a critical threshold, model accuracy declines sharply. But the pruning curve of accuracy and MAPE outperform the results of layer-wise pruning which indicates a finer level of compression. Finally, we selected $(x_1, x_2) = (0.6, 0.9)$ as the optimal pruning parameter, by which 60% of total weights are pruned and neurons with over 90% zeros are discarded, yielding the final compressed model.

TABLE II
FINAL MODEL INFORMATION

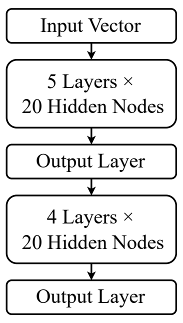
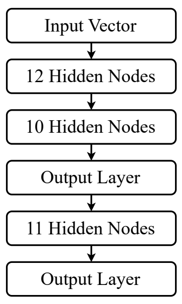
Model Information	Before Compression	After Compression
Model Structure	 Input Vector 5 Layers × 20 Hidden Nodes Output Layer 4 Layers × 20 Hidden Nodes Output Layer	 Input Vector 12 Hidden Nodes 10 Hidden Nodes Output Layer 11 Hidden Nodes Output Layer
FLOPs	6960	366
Accuracy(%)	69.82	67.42
MAPE(%)	3.43	4.61

Table II shows the final compressed model informatioin. Model size with respect to FLOPs has been compressed by 94.74%, while accuracy and MAPE are degraded only by 2.4% and 1.18% respectively.

V. Full System Performance

A. System Setup and Implementation

We conduct our simulations using GPGPU-Sim [7] and employ the McPAT [10] power model to perform power consumption analysis. Our setup is based on the Nvidia GeForce GTX Titan X architecture, which comprises 24 clusters. DVFS is applied at the per-cluster level, with six available V/f operating points: (1.0V, 683MHz), (1.0V, 780MHz), (1.0V, 878MHz), (1.0V, 975MHz), (1.1V, 1100MHz), (1.155V, 1165MHz) [8]. The default V/f operating point is set to (1.0 V, 1165MHz). We randomly select benchmarks from the Rodinia [4], Parboil [5] and PolyBench [6]. To ensure the generalization capability of the SSMDVFS model, more than 50% of the selected programs are not included in the training set. Additionally, we limit the execution time of programs to approximately 0.0003 s, ensuring that short duration tasks can also benefit from microsecond-scale SSMDVFS mechanism.

B. Comparison Methodology

The primary objective of our SSMDVFS mechanism is to identify the minimal V/f operating point that maintains performance loss below a predefined threshold. To accurately assess the effectiveness of different methods, the consistency between optimization objectives must be ensured. Thus, we modify both the analytical method PCSTALL and the RL-based approach F-LEMMA to fit into similar objectives as ours.

PCSTALL [1] takes advantage of the linear additivity of frequency sensitivity metrics and exploits the iterative computational patterns typically seen in GPGPUs to reformulate the optimization problem. Traditionally, PCSTALL employs a control mechanism aimed at minimizing the Energy-Delay Product (EDP). In the adapted version, while the frequency sensitivity-based prediction model is retained, the objective function has been modified. Frequency sensitivity is used to estimate the number of executed instructions, which in turn is used to predict performance loss. The minimal frequency that satisfies performance requirements is then selected, ensuring consistency between the original and revised optimization objectives.

F-LEMMA [2] utilizes a hierarchical actor-critic reinforcement learning framework combined with a linear classifier, enabling fine-grained prediction and coarse-grained training for DVFS decision-making. The original reward function balances normalized power consumption and normalized instruction count by linearly combining both metrics and adjusting their respective weights. In the modified version, the baseline value for the instruction count normalization reward function is reduced by a certain percentage to allow for performance degradation. Furthermore, faster F-LEMMA is applied by reducing the actor-critic model update cycle, enabling it to better accommodate fine-grained DVFS and short-duration GPU programs.

C. Experiment Result

We compare SSMDVFS with PCSTALL and F-LEMMA using program information from the default V/f operating point as the baseline. Additionally, we test SSMDVFS with and without Calibrator to validate the calibration mechanism. The Energy-Delay Product (EDP) is used as the primary metric, along with program execution time data. In Fig. 4, the two left plots use a 10% performance loss preset, and the two on the right use a 20% preset. Across both presets, SSMDVFS achieved a 7.85% EDP reduction compared to the baseline, 9.91% compared to PCSTALL, and 29.19% compared to the RL-based approach. Fully compressed SSMDVFS models using GPGPU-Sim yielded similar results to the pre-compressed model. On average, the compressed SSMDVFS achieved an 11.09% EDP reduction compared to the baseline, 13.17% compared to PCSTALL, and 36.80% compared to the RL-based method.

Moreover, the latency data in Fig. 4 reveal that both PCSTALL and SSMDVFS successfully kept performance loss within the specified limits. For cases where certain programs exceeded the preset thresholds, adding Calibrator reduced latency, bringing it back under control. In contrast, the RL-based method performed poorly for short-duration programs, both in terms of controlling latency and minimizing EDP through V/f operating point selection. This suboptimal performance is largely due to the RL method's need for an extensive warm-up period to explore the state-action space and gather reward values for subsequent learning. As a result, it struggles to reduce power consumption in short-duration GPU programs, where the overhead of the initial exploration outweighs the potential benefits.

D. Hardware Implementation

We implemented the proposed SSMDVFS module in ASIC (Application Specific Integrated Circuit) by writing Verilog code on VIVADO 2019.2, and we got the synthesis result using Cadence RTL Compiler and Innovus with 65 nm TSMC library. Besides, we scale the power and area consumption to 28 nm

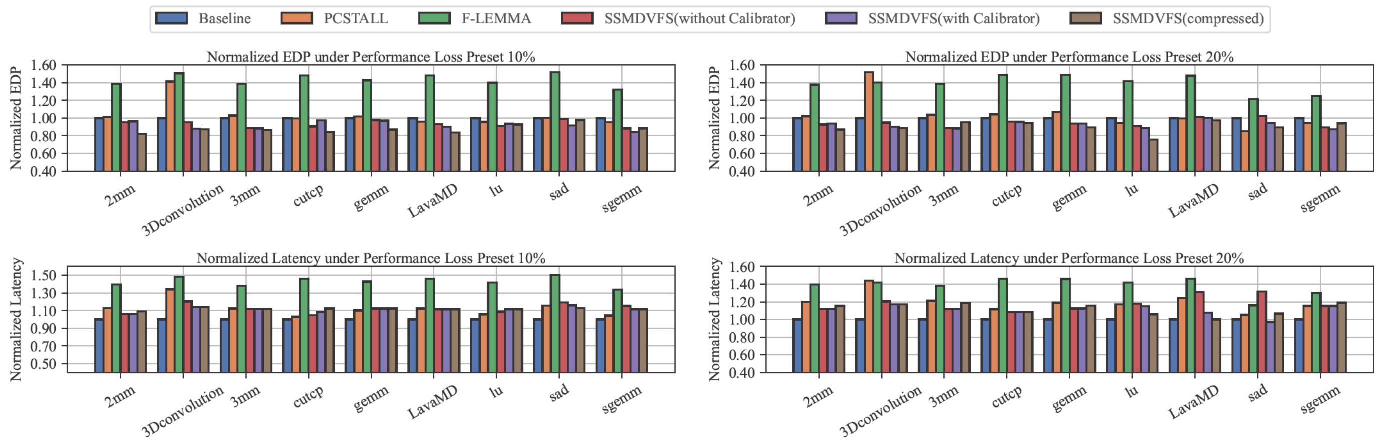


Fig. 4. Normalized EDP and Latency for Different DVFS Mechanism with Performance Loss Preset of 10% and 20%

using DeepScale Tool [11] which is the same technology node as our GPU, and the module is designed for FP32 calculation. After scaling, the module occupies only 0.0080 mm², which is negligible compared to the GPU's die area of hundreds of square millimeters. Additionally, the power consumption per inference is calculated at 0.0025 W, a negligible fraction of the Nvidia GTX Titan X GPU's 250 W TDP [8]. Simulations show that the module requires 192 clock cycles per inference, which is calculated to 0.16 µs at the default frequency 1165 MHz, constituting only 1.65% of one 10 µs DVFS predicting period, thereby imposing minimal latency on the GPU's overall operation.

VI. Related Work

As GPGPU applications continue to expand across various domains, the demand for higher computational density has led to increased power consumption and heat generation. This rise in power and thermal output can significantly hinder performance, making techniques like DVFS and power gating essential for regulating power consumption and ensuring sustainable operation. Also, with the development of integrated voltage regulators (IVRs), per-core microsecond level fast DVFS has become practical. Toprak-Deniz et al. [12], Meinerzhagen et al. [13], Kim et al. [14], and Keller et al. [15] designed IVRs that can support sub-microsecond level dynamic voltage scaling. Kim et al. [16] and Eyerman et al. [17] studied energy benefits from microsecond-scale dynamic voltage scaling supported by on-chip IVRs.

With relative hardware support, analytical methods and machine learning-based techniques, particularly reinforcement learning (RL), become two prominent approaches to optimizing DVFS in microsecond-scale. In terms of analytical DVFS methodology, Rahmani et al. [18], Ebi et al. [19], Lai et al. [20] and Kanduri et al. [21] explored reliability/variability, thermal, latency or accuracy aware solutions. Winter et al. [22] presented a thread scheduling and global power management co-design for a heterogeneous many-core processor. Nath et al. [3] applied a detailed analytical model to GPUs, based on which, Srikant et al. [1] leveraged an iterative pattern to formulate a DVFS prediction mechanism. DVFS has degraded to a millisecond scale, thus the analytical model is built up with extra complexity.

Meanwhile, in terms of machine learning-based DVFS methodology, Jung et al. [23], Shen et al. [24], [25], Chen et al. [26], [27], Rapp et al. [28] and Yu et al. [29] used a learning-based predictor and controller to find optimal power and performance. Zou et al. [2] applied RL learning to build one hierarchical power management framework. However, these methods were designed for manycores only and fine-grained DVFS was constantly missing in terms of machine learning for GPUs. Also, there are others researches [30]–[33] solving power and computational performance trade-off in both hardware and system levels.

VII. Conclusion

This paper presents the supervised and self-calibrated machine learning framework (SSMDVFS) designed for effective microsecond-scale GPU voltage and frequency scaling. The end-to-end architecture encompasses data generation, neural network model design, training, compression, and runtime calibration, offering a comprehensive solution. Unlike traditional analytical models, which often fail to accurately capture GPU architectures, and reinforcement learning methods that may struggle with convergence, SSMDVFS provides a robust approach for dynamic scaling. Experimental results confirm that our framework enhances the energy-delay product (EDP) by 11.09%, surpassing analytical models and reinforcement learning techniques by 13.17% and 36.80%, respectively. These findings highlight the practical advantages of SSMDVFS in optimizing GPU performance and efficiency.

Acknowledgment

This research was supported by the National Natural Science Foundation of China (NSFC) under grant number `NSFC62202287` and the CLQ Project under grant number `CLQ202302`.

References

[1] S. Bharadwaj, S. Das, K. Mazumdar, B. M. Beckmann, and S. V. Kosonocky, "Predict; don't react for enabling efficient fine-grain dvfs in gpus," *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:248496909

[2] A. Zou, K. Garimella, B. Lee, C. Gill, and X. Zhang, "F-lemma: Fast learning-based energy management for multi-/many-core processors," in *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (ML-CAD)*, 2020, pp. 43–48.

[3] R. Nath and D. Tullsen, "The crisp performance model for dynamic voltage and frequency scaling in a gpgpu," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 281–293.

[4] 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)*, 2009, pp. 44–54.

[5] J. A. Stratton, S. S. Stone, and W. m. W. Hwu, "Mcuda: An efficient implementation of cuda kernels on multicore cpus," Center for Reliable and High-Performance Computing, UIUC, Tech. Rep., 2010.

[6] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite," http://web.cse.ohio-state.edu/ pouchet/software/polybench, accessed: September 15, 2024.

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

[8] 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.

[9] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, pp. 389–422, 01 2002.

[10] 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.

[11] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[12] Z. Toprak-Deniz, M. Sperling, J. Bulzacchelli, G. Still, R. Kruse, S. Kim, D. Boerstler, T. Gloekler, R. Robertazzi, K. Stawiasz *et al.*, "5.2 distributed system of digitally controlled microregulators enabling per-core dvfs for the power8 tm microprocessor," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2014.

[13] P. Meinerzhagen, C. Tokunaga, A. Malavasi, V. Vaidya, A. Mendon, D. Mathaikutty, J. Kulkarni, C. Augustine, M. Cho, S. Kim *et al.*, "An energy-efficient graphics processor featuring fine-grain dvfs with integrated voltage regulators, execution-unit turbo, and retentive sleep in 14nm tri-gate cmos," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 38–40.

[14] S. T. Kim, Y.-C. Shih, K. Mazumdar, R. Jain, J. F. Ryan, C. Tokunaga, C. Augustine, J. P. Kulkarni, K. Ravichandran, J. W. Tschanz *et al.*, "Enabling wide autonomous dvfs in a 22 nm graphics execution core using a digitally controlled fully integrated voltage regulator," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 18–30, 2015.

[15] B. Keller, M. Cochet, B. Zimmer, Y. Lee, M. Blagojevic, J. Kwak, A. Puggelli, S. Bailey, P.-F. Chiu, P. Dabbelt *et al.*, "Sub-microsecond adaptive voltage scaling in a 28nm fd-soi processor soc," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016.

[16] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *14th International Symposium on High Performance Computer Architecture*. IEEE, 2008.

[17] S. Eyerman and L. Eeckhout, "Fine-grained dvfs using on-chip regulators," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 1, pp. 1–24, 2011.

[18] A. M. Rahmani, M.-H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-aware runtime power management for many-core systems in the dark silicon era," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 2, pp. 427–440, 2016.

[19] T. Ebi, M. A. Al Faruque, and J. Henkel, "Tape: Thermal-aware agent-based power econom multi/many-core architectures," in *2009 IEEE/ACM International Conference on Computer-Aided Design-Digest of Technical Papers*. IEEE, 2009, pp. 302–309.

[20] Z. Lai, K. T. Lam, C.-L. Wang, and J. Su, "Latency-aware dvfs for efficient power state transitions on many-core architectures," *The Journal of Supercomputing*, vol. 71, no. 7, pp. 2720–2747, 2015.

[21] A. Kanduri, M.-H. Haghbayan, A. M. Rahmani, P. Liljeberg, A. Jantsch, H. Tenhunen, and N. Dutt, "Accuracy-aware power management for many-core systems running error-resilient applications," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 10, 2017.

[22] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *2010 19th International Conference on Parallel Architectures and Compilation Techniques*. IEEE, 2010, pp. 29–39.

[23] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1395–1408, 2010.

[24] H. Shen, J. Lu, and Q. Qiu, "Learning based dvfs for simultaneous temperature, performance and energy management," in *Thirteenth International Symposium on Quality Electronic Design*. IEEE, 2012.

[25] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Transactions on Design Automation of Electronic Systems*, pp. 1–32, 2013.

[26] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2015.

[27] Z. Chen, D. Stamoulis, and D. Marculescu, "Profit: priority and power/performance optimization for many-core systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 2064–2075, 2017.

[28] M. Rapp, A. Pathania, T. Mitra, and J. Henkel, "Prediction-based task migration on s-nuca many-cores," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1579–1582.

[29] Z. Yu, P. Machado, A. Zahid, A. M. Abdulghani, K. Dashtipour, H. Heidari, M. A. Imran, and Q. H. Abbasi, "Energy and performance trade-off optimization in heterogeneous computing via reinforcement learning," *Electronics*, vol. 9, no. 11, p. 1812, 2020.

[30] A. Zou, J. Leng, Y. Zu, T. Tong, V. J. Reddi, D. Brooks, G.-Y. Wei, and X. Zhang, "Ivory: Early-stage design space exploration tool for integrated voltage regulators," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[31] A. Zou, J. Leng, X. He, Y. Zu, C. D. Gill, V. Janapa Reddi, and X. Zhang, "Voltage-stacked gpus: A control theory driven cross-layer solution for practical voltage stacking in gpus," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 390–402.

[32] A. Zou, J. Li, C. D. Gill, and X. Zhang, "Rtgpu: Real-time gpu scheduling of hard deadline parallel tasks with fine-grain utilization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1450–1465, 2023.

[33] R. Sun, Y. Ni, X. He, J. Zhao, and A. Zou, "One-sa: Enabling nonlinear operations in systolic arrays for efficient and flexible neural network inference," in *2024 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2024, pp. 1–6.