Performance-Aware Energy-Efficient GPU Frequency Selection using DNN-based Models

Ghazanfar Ali
Texas Tech University
Lubbock, USA
ghazanfar.ali@ttu.edu

Mert Side
Texas Tech University
Lubbock, USA
mert.side@ttu.edu

Sridutt Bhalachandra
Lawrence Berkeley National Laboratory
Berkeley, USA
sriduttb@lbl.gov

Nicholas J. Wright
Lawrence Berkeley National Laboratory
Berkeley, USA
njwright@lbl.gov

Yong Chen
Texas Tech University
Lubbock, USA
yong.chen@ttu.edu

ABSTRACT

Energy efficiency will be important in future accelerator-based HPC systems for sustainability and to improve overall performance. This study proposes a deep neural network (DNN)-based learning model for execution time and power consumption of workloads across GPUs DVFS design space. Micro-architectural data obtained by running SPEC-ACCEL, DGEMM, and STREAM benchmarks are used for model training. These features are consistent for a workload unaffected by frequency and input size reducing the data required significantly. For real-world applications - LAMMPS, NAMD, GROMACS, LSTM, BERT, and ResNet50 power and time models show 89% – 98% accuracy on NVIDIA Ampere. Multi-objective functions help select optimal frequencies that lower power and minimize performance impact showing maximum energy savings of 27% at a performance loss of 1.8%. The same models trained on Ampere showed an accuracy of greater than 93% on an NVIDIA Volta, thereby demonstrating model portability across architectures.

KEYWORDS

GPU, dynamic voltage frequency scaling, Ampere GPU, Volta GPU, energy-efficiency, high-performance computing

ACM Reference Format:
Ghazanfar Ali, Mert Side, Sridutt Bhalachandra, Nicholas J. Wright, and Yong Chen. 2023. Performance-Aware Energy-Efficient GPU Frequency Selection using DNN-based Models. In *52nd International Conference on Parallel Processing (ICPP 2023), August 07–10, 2023, Holladay, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605573.3605600

1 INTRODUCTION

While the first exascale system, Frontier [40], delivered on the 20MW power budget goal [13], future deployments are set to consume almost double this power [42]. These future HPC systems are estimated to achieve more than 90% compute capability from GPUs [30]; therefore, improving the energy efficiency of GPUs can significantly contribute to lowering the overall power budget of GPU-enabled HPC systems.

Three key questions drive our current research. First, as GPU workloads exhibit different computational intensities, what key features could uniquely identify their power and performance behaviors? Second, GPUs offer a wide array of performance and power controls. For example, NVIDIA GA100 [5] supports 81 *dynamic voltage frequency scaling (DVFS)* configurations in the range between 210 MHz and 1410 MHz. While these configurations favor saving power, they also make the power and performance control design space more complex. How do we go beyond traditional brute-force approaches that may not always be feasible? Third, an HPC center aspires to reduce power with little or no impact on performance. How to select a DVFS configuration that satisfies these criteria?

Limitations of state-of-art approaches: While several studies have explored the relationship between GPU utilization, power, and performance, such as [1, 7, 12, 16], little attention has been given to identifying fine-grain GPU's low-level utilization features that uniquely recognizes the power and performance signature of a workload. Furthermore, the features identified in previous research are not always portable across applications and agnostic to changes in input sizes and DVFS configurations. Numerous studies that have intended to model power and performance behaviors across different DVFS configurations using analytical and machine learning (ML)-based models have limitations too. For example, data acquisition of these existing methodologies is complex and costly, and often they require metrics collection at all supported DVFS configurations [4, 7, 41, 43]. Many proposed ML models are specific to optimizing power or performance and thus have higher performance overheads while optimizing for energy. On the other hand, existing models may use application-specific characteristics and hence are not versatile. The majority of the models are based on *traditional ML-based* learners that may not be suitable for making reliable predictions for workloads with different computational patterns. Last, the existing multi-objective optimal approaches provide a set of multiple optimal DVFS configurations rather than a single

ICPP 2023, August 07–10, 2023, Holladay, UT, USA

ACM ISBN 979-8-4007-0843-5/23/08.
https://doi.org/10.1145/3605573.3605600

DVFS configuration for an application [8, 11]. However, this flexibility may come at the cost of simplicity for an average user unaware of the performance implications of a particular DVFS setting.

In this study, we first characterized and selected GPU utilization metrics correlating power and execution time using the mutual information (MI) technique. Then, we developed power and performance models based on a DNN guided by the selected features. Finally, using a multi-objective function, the predicted values are used to select the optimal GPU frequency for a workload.

Key insights and contributions: This study make the following key contributions:

- *Selection of fine-grained features:* Using the Mutual Information, we analyzed different GPU utilization metrics for their relationship with power and performance. We identified features agnostic to changes in DVFS and input size. It is shown that the same set of features can model both power and time.
- *Deep learning-based power and performance models and selection of optimal frequency:* We developed models based on deep learning to improve prediction accuracy for GPU workloads that exhibit linear, non-linear, and other complex relationships between utilization metrics and power/ performance. These are efficient in learning complex power and performance behaviors of applications across different DVFS configurations. A multi-objective function (energy-delay product (EDP) or ED^2P) is used to select the optimal frequency.
- *Portability of the approach:* The portability of the proposed approach was validated on both application and architecture levels. The power and performance models, developed using metrics from general-purpose GPU benchmarks (SPEC ACCEL, DGEMM, STREAM), predicted the power and performance of real applications (which were not used in model training) on NVIDIA GV100 and GA100 architectures with an accuracy between 89% and 98%. The energy profiles were chosen using ED^2P achieved, on average, an energy saving of up to 27.1% with 1.8% performance loss for the real applications on GA100. Further, energy saving of up to 23.6% was achieved with less than 1% performance loss on GV100.

2 MOTIVATION

Understanding the impact of power controls on workload power and performance is an essential foundational step toward enabling power and energy-efficient computing. To understand these patterns, we analyzed the impact of 61 DVFS configurations, in the range of 510 MHz - 1410 MHz, of NVIDIA GA100 on the power, performance, and energy of DGEMM (i.e., representative of compute-intensive workload) and STREAM (i.e., representative of memory-intensive workload) micro-benchmarks, as shown in Figure 1. The workloads running below 510 MHz experience heavy performance degradation and are thus excluded from the design space. Figures 1 (a) and (e) show that the power is approximately a nonlinear function of GPU core frequency. The observation shows that at the maximum core frequency, a compute-intensive workload can use power up to 100% of GPU's thermal design power (TDP), and a memory-intensive workload can use up to ~50% of the TDP. Similarly, at the lowest frequency, the power for compute- and memory-intensive workloads can be reduced by up to one-fifth of the TDP.

Figures 1 (b) and (f) show that execution time and frequency exhibit an inversely nonlinear relationship. However, DGEMM and STREAM attain maximum performance at different frequencies. We can generalize that the performance of a memory-intensive workload is less impacted at lower frequencies than a compute-intensive workload. Energy (Figures 1 (c) and (g)) is a product of power and execution time. Thus, it shows a nonlinear relationship with the core frequency. Predictably, energy is higher at the lower and higher frequency ranges. This increase in energy at the extremes is due to higher execution time at the lower frequency ranges and higher power at higher frequency ranges.

Figure 1 (d) shows a measure of performance in terms of floating point operations per second (FLOPS), which is a direct linear function of core frequency, and the highest FLOPS were attained at the maximum frequency. The FLOPS count was reduced by about one-fourth at the lowest frequency. Thus, lower frequencies significantly degrade the performance of the compute-intensive kernels.

Figures 1 (h) showed the change in bandwidth in relation to the core frequency, which initially increases with the frequency but flattens at ~ 900 MHz. In alignment with the execution time behavior of the STREAM, the bandwidth does not significantly improve after ~900 MHz. Thus, memory-intensive kernels can be clocked to the lower frequency ranges without a significant impact on performance.

The impact of DVFS on an application is highly dependent on its computational intensity. In Figure 1, DGEMM shows optimal energy and run time at 1080 MHz and 1410 MHz (maximum frequency), respectively. Likewise, STREAM provides optimal energy and run time at 1005 MHz and 1260 MHz, respectively. Thus, there is no universally optimal DVFS configuration.

3 EXPERIMENTAL SETUP

In this study, we collected the utilization metrics for SPEC ACCEL, DGEMM, and STREAM, real applications (LAMMPS, NAMD, GROMACS, and LSTM) from an NVIDIA Ampere A100 GPU node located at the National Science Foundation (NSF)'s Chameleon CHI@UC site [20]. The Volta V100 GPU measurements were obtained from a node at High Performance Computing Center, Texas Tech University [15]. Table 1 lists the configuration of the NVIDIA GPU used in this study. We used CUDA 11.2 (NVIDIA driver version 450) and CUDA 11.5 (driver version 495) for GV100 and GA100, respectively. All our experiments were performed with exclusive access to the node to avoid any interference from other applications.

Table 1: Specifications of the GPUs used in this study.

	GA100	GV100
Core Frequency Range (MHz)	[210:1410]	[135:1380]
Default Core Frequency (MHz)	1410	1380
Used DVFS Configurations	61 out of 80	117 out of 167
Memory Frequency (MHz)	1597	877
GPU Memory (HBM2e) (GB)	80	40
Peak Memory Bandwidth (GB/s)	2039	900
TDP (W)	500	250

We used DGEMM [26] and STREAM [6] as our micro-benchmarks since they are representative of compute- and memory-intensive

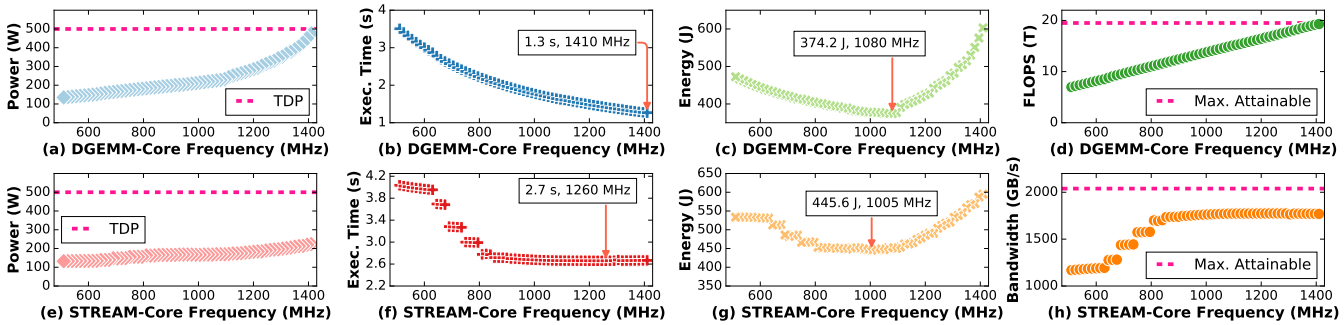


Figure 1: Power, execution time, energy, and FLOPS variations across different frequency configurations of NVIDIA A100 GPU for DGEMM (upper) and STREAM (lower).

workloads, respectively. The proposed models were trained using DGEMM, STREAM, and SPEC ACCEL® benchmark suite [18]. The SPEC ACCEL benchmark suite consists of 19 parallel workloads specifically designed to test the performance of the accelerators, such as GPUs. The portability of the proposed models was evaluated using four GPU real-world applications on NVIDIA GV100 and GA100 GPUs. Table 2 lists the applications used for the training and evaluations of our models.

Table 2: List of applications used in this study.

Category	Applications
SPEC ACCEL [Training]	`TPACF`, `STENCIL`, `LBM`, `FFT`, `SPMV`, `MRIQ`, `HISTO`, `BFS`, `CUTCP`, `KMEANS`, `LAVAMD`, `CFD`, `NW`, `HOTSPOT`, `LUD`, `GE`, `SRAD`, `HEARTWALL`, `BPLUSTREE`
Micro-Benchmarks [Training]	`DGEMM`, `STREAM`
Real-world [Evaluation]	`LAMMPS`, `NAMD`, `GROMACS`, `LSTM`, `BERT`, `ResNet50`

4 METHODOLOGY

Our goal is to devise a systematic method that reduces energy for applications with minimal performance degradation. Figure 2 illustrates the key components involved in finding an optimum DVFS configuration. Our approach consists of two phases: (1) an offline training phase; and (2) an online prediction phase.

Each training workload was executed three times across all supported DVFS frequencies. The execution time of some workloads was small, so a 20-ms interval is used to collect metrics to develop a statistically significant dataset. These metrics are analyzed (see Section 4.2 for more details) to identify features that affect a workload's power and execution time. The deep learning-based power and performance models are then constructed using these features.

In the prediction phase, an application is executed only at the maximum frequency to acquire the model features. This is because the features do not change significantly across DVFS configurations (see Section 4.2) and are used across all other frequencies to create the dataset for the workload. This is one of the key distinctions between our methodology and other prior works involving utilization metrics. This dataset is then used to predict power and execution time across all DVFS configurations for the workload. The energy for the workload is computed using the predicted power and execution time. The predicted execution time and power at each frequency are used to select the optimal frequency.

4.1 Data Collection Framework

This study has designed and developed a transparent, extensible framework (no compiling or linking needed) for GPU data (metrics) acquisition. The data acquisition is performed using three modules. The `launch` module initiates data acquisition and allows the orchestration of the data collection process. It enables the specification of the desired DVFS configurations, executables' names with their arguments and paths, the results path, the number of runs, and the sampling interval. The `control` module applies the desired operating frequency to the GPU cores and memory to regulate the GPU power. The NVIDIA Data Center GPU Manager (DCGM) interface [29] is used for frequency control and metric collection. The `profile` module runs the application and collects GPU metrics using DCGM throughout the execution of the application. Finally,

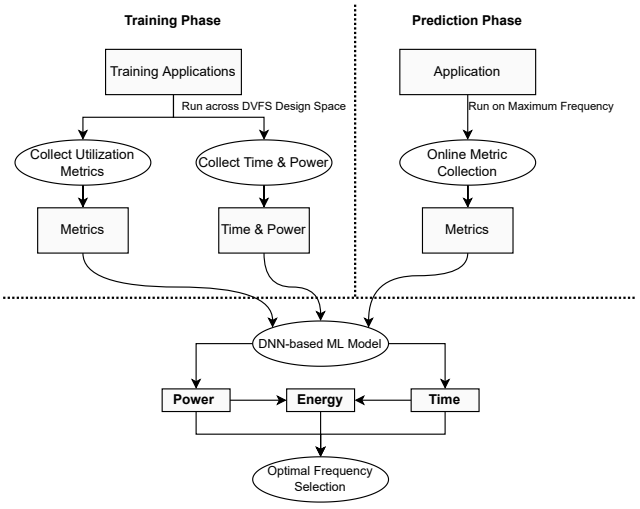


Figure 2: Overview of the methodology for selecting the DVFS configuration with optimal energy and performance.

the `launch` module saves output metrics of each run into a comma-separated values format file.

We collected 12 GPU utilization metrics ((1) `fp64_active`, (2) `fp32_active`, (3) `sm_app_clock`, (4) `dram_active`, (5) `gr_engine_active`, (6) `gpu_utilization`, (7) `power_usage`, (8) `sm_active`, (9) `sm_occupancy`, (10) `pcie_tx_bytes`, (11) `pcie_rx_bytes`, and (12) `exec_time`) for DGEMM, STREAM, and SPEC ACCEL benchmarks. For the four real applications (LAMMPS, NAMD, GROMACS, and LSTM), only model features, power, and execution time are collected at the maximum frequency for prediction and validation.

4.2 Feature Characterization

We identified the relationship between power, execution time, and the GPU's utilization features. Furthermore, we investigated the impact of DVFS and input size on them.

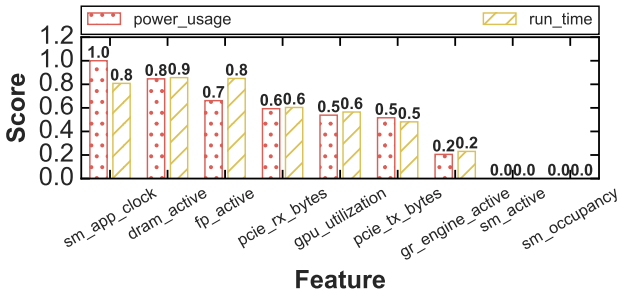


Figure 3: Feature dependency for predicting power and time.

4.2.1 Selection of the Fine-Grained Features. Feature selection was performed to choose features that directly impact an application's power usage and execution time. These features are critical in developing accurate and reliable ML models. We used MI [32] technique to identify the features closely related to predictands (i.e., power_usage and execution_time). MI is a nonparametric procedure based on estimating distances using k-nearest neighbors [22, 35]. The advantage of this approach is that it is neutral and independent of machine learning algorithms. We used the dataset for DGEMM and STREAM to discover the relationship between the predictands and features, as shown in Figure 3. The feature with a higher mutual correlation (close to 1) value indicates higher dependency with the predictand. Out of 10 features, we selected the top three features (i.e., `fp_active`, `sm_app_clock`, `dram_active`) that showed the highest association with power usage and execution time.

The impact of *sm_app_clock* on power and execution time is shown in Figure 1 for supported frequencies. For both micro-benchmarks, a decrease in frequency causes a decrease in power and an increase in execution time. However, the rate of change for DGEMM is higher than STREAM for both cases. It is well known that frequency changes have a more prominent effect on compute-bound kernels like DGEMM compared to STREAM, which is memory-bound. Therefore, knowing the computational intensity of a workload is necessary, and the `fp_active` and `dram_active` metrics help with this.

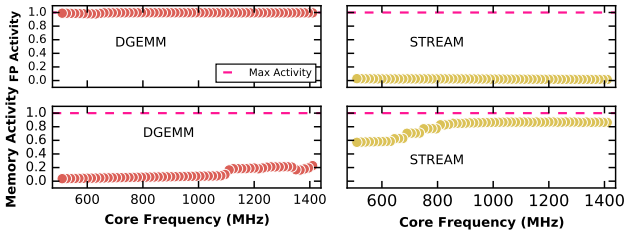


Figure 4: Impact of DVFS on the computational activities (i.e., `fp_active` and `dram_active`) of STREAM and DGEMM.

4.2.2 Impact of DVFS on Selected Features. We further investigated the impact of changes in DVFS configurations on `fp_active` and `dram_active` for memory- and compute-intensive workloads. DGEMM and STREAM were tested by changing the frequency at the maximum input size supported. As demonstrated in Figure 4, the floating-point activity is almost unaffected by the change in frequency for both compute- and memory-intensive applications; however, memory-activity shows variations to some extent for both applications. Nevertheless, this change in memory activity did not show any noticeable impact on our model prediction.

4.2.3 Impact of Input Size on Selected Features. We investigated the impact of changes in input sizes on `fp_active` and `dram_active`. DGEMM and STREAM were tested using different input sizes at maximum frequency (i.e., 1410 MHz), as depicted in Figure 5. As in the case of frequency, floating-point activity is also unaffected by the change in the input size. For STREAM, memory activity is largely unaffected by the change in input sizes; however, it increases with the increase in input size for DGEMM. Again, the impact of input size change on memory activity showed little effect on DGEMM power/time prediction.

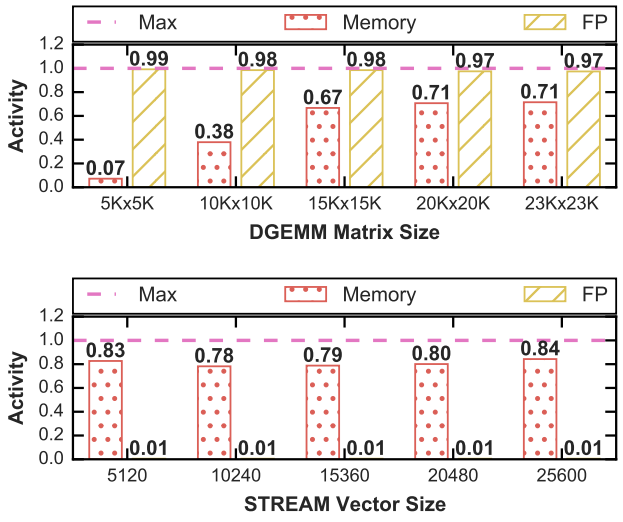
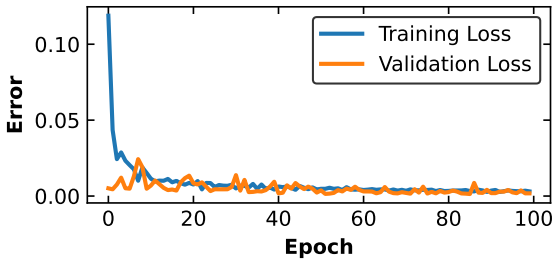
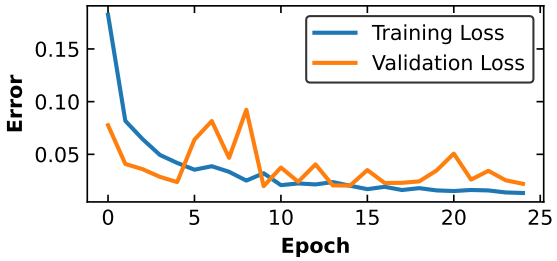


Figure 5: Impact of input sizes on the computational activities (i.e., `fp_active` and `dram_active`) of STREAM and DGEMM.

4.2.4 Features Portability Across GPU Architectures. We analyzed the portability of the selected features on the NVIDIA GV100 GPU architecture. We observed that patterns of `fp_active` and `dram_active` for DGEMM and STREAM were similar to their patterns on GA100 as depicted in Figures 4 and 5. The selected features and the proposed approach are thus portable to Volta.



(a) Power model loss



(b) Performance model loss

Figure 6: The training loss and validation loss for our models.

Summary: We confirm that the FP activity and memory activity directly influence power and execution time. They can uniquely identify power patterns for the applications regardless of their input sizes and DVFS configurations. Thus, we consider the feature values obtained at default as constant, thereby making our models portable to unseen applications and GPU architectures.

4.3 Power and Performance Modeling

The non-linear behaviors of an application's power and execution time with respect to frequency change led our decision to use a deep learning-based standard feedforward neural network (FNN) [17, 36] to model the power and execution time of applications across different DVFS configurations. These models were constructed using input, hidden, and output layers, and trained on the same dataset. The dataset consists of the selected features for 21 benchmarks – DGEMM, STREAM, and 19 benchmarks in SPEC ACCEL across 61 DVFS configurations on NVIDIA GA100 GPU. As shown in Equation 1, feature vector $\vec{X}$ consists of three features, `fp_active`, `dram_active`, and `sm_app_clock`, for both power and time models.

$$\vec{X} = (sm\_app\_clock_{\mathbf{1}}, fp\_active_{\mathbf{1}}, dram\_active_{\mathbf{1}}), ..., (sm\_app\_clock_{\mathbf{N}}, fp\_active_{\mathbf{N}}, dram\_active_{\mathbf{N}}) \quad (1)$$

The optimal configurations for the proposed model architectures were selected after extensive testing with various numbers of layers, optimizers, activation functions, and epochs. We ran experiments with activation functions including rectified exponential linear unit (`ReLU`), exponential linear unit (`elu`), Leaky ReLU, scaled exponential linear unit (`SELU`), `sigmoid`, `tanh`, `softmax`, `softplus`, `softsign` coupled with various learning optimizers including `Adam`, `Adamax`, `Nadam`, `RMSprop`, and `AdaDelta`. Based on our experiments, `SELU` is the most efficient and provides robust inference for unseen applications. Furthermore, it almost eliminates the risk of gradients vanishing. Thus, we selected `SELU` as an activation function [21] and `RMSprop` [39] for training both models. The `SELU` function is defined as shown in Equation 2.

$$selu(x) = \begin{cases} scale \cdot x, & \text{if } x > 0 \\ scale \cdot alpha \cdot (exp(x) - 1), & \text{if } x < 0 \end{cases} \quad (2)$$

where `alpha` and `scale` are pre-defined constants (i.e., `alpha=1.67326324` and `scale=1.05070098`).

The target variable for the power model was `power_usage` and is represented as shown in Equation 3.

$$Y_{power} = (power\_usage_{\mathbf{1}}, ..., power\_usage_{\mathbf{N}}) \quad (3)$$

The dataset is represented as

$$(\vec{X}, Y_{power}) = (\vec{x_{\mathbf{1}}}, y_{power_{\mathbf{1}}}), ..., \vec{x_{\mathbf{N}}}, y_{power_{\mathbf{N}}})) \quad (4)$$

where $\vec{x_{\mathbf{i}}}$ (see Equation 1) is a feature vector and y_{power_i} is the target value. The training of the power model is initiated by feeding the dataset in Equation 4 as input. We used three hidden layers, 64 neurons in each layer, and the same was used as the batch size. The functional behavior of each neuron is shown in Equation 5. Each neuron performs three key tasks: (1) receiving input signals, i.e., x_1, x_2, ..., x_N multiplied with their corresponding weights; (2) computing the neuron's input `s` by adding weighted signals and bias b; and (3) activating the neuron's output using Equation 5.

$$s = \sum_{i=1}^{N} w_i x_i + b \quad (5)$$

In this way, all neurons learn about the training dataset, and the output layer predicts the final outcome. An error is computed by comparing measured and predicted values using MSE. Based on the error, new weights are propagated using the backpropagation mechanism. This procedure continues until the network is stabilized to a minimum error. We selected the number of epochs by considering training loss and overfitting. The training dataset was split into training (80%) and validation (20%) sets to compute the losses. The training loss and validation loss were observed to fit up to 100 epochs optimally, as shown in Figure 6 (a).

Likewise, to build the performance model, the same dataset (Equation 4) was used as input, and `execution_time` was used as the target variable. The target variable for the performance model was `execution_time` and is represented as shown in Equation 6.

$$Y_{time} = (execution\_time_{\mathbf{1}}, ..., execution\_time_{\mathbf{N}}) \quad (6)$$

The dataset is represented as

$$(\vec{X}, Y_{time}) = (\vec{x_{\mathbf{1}}}, y_{time_{\mathbf{1}}}), ..., \vec{x_{\mathbf{N}}}, y_{time_{\mathbf{N}}})) \quad (7)$$

where $\vec{x_{\mathbf{i}}}$ (see Equation 1) is a feature vector and y_{time_i} is the target value. The training of the time model is initiated by feeding the dataset in Equation 7 as input. We used three hidden layers, with

64 neurons in each layer, and the same was used as the batch size. The functional behavior of each neuron is the same as discussed in the power model above. For the performance model, the training loss and validation loss converged after 25 epochs, as shown in Figure 6 (b). After 25 epochs, slight overfitting was observed, and we stopped training here to avoid overfitting. The execution time for training power and performance models were 6.5 and 2.6 seconds, respectively. The variations in time are due to each model's varying numbers of epochs to converge. The power and performance prediction took only about 0.2 seconds.

4.4 Optimal Frequency Selection

As discussed in Section 2, choosing an optimal frequency is to reduce power with minimal performance degradation. The optimal frequency for an application is selected using a multi-objective function. Our framework allows a user to define this objective function. However, in the current study, EDP and ED^2P [10, 23, 25, 31] were used. These approaches require energy and execution time across different DVFS configurations. The energy is computed using the power usage and the execution time predicted via the proposed power and time models across 61 DVFS configurations, as shown in Equation 8.

$$E_{f_{predicted}} = P_{f_{predicted}} \times T_{f_{predicted}} \tag{8}$$

The algorithm for selecting an optimal frequency among supported DVFS configurations is straightforward and shown in Algorithm 1. This algorithm takes performance degradation threshold (in %) and three lists: energy (`E`), execution time (`T`), and frequency (`F`) as input. It outputs an optimal `f` setting based on the EDP score. The algorithm involves two major steps: First, the EDP score for each set of energy and time is computed by multiplying the energy by the execution time. Second, the lowest score decides the optimal energy-delay profile out of the given sets of energy and time for the given workload. The frequency (`f`) corresponding to the lowest score is selected as the EDP-based optimal frequency. The frequency is the desired configuration when the performance degradation is less than the threshold degradation. A higher frequency configuration is selected when the performance loss is greater than the threshold degradation. This step is repeated until the performance degradation is less than the threshold degradation. Note that our evaluation selected optimal frequency without using the performance degradation threshold. Thus, performance degradation is solely determined by EDP. The optimal frequency selection using ED^2P is similar to this algorithm, and the only difference is that the ED^2P score is calculated instead of the EDP score, where the energy is multiplied by the square of the execution time to give more emphasis to the execution time).

Data: $th, E_1 \dots E_N, T_1 \dots T_N, F_1 \dots F_N$ // perf. loss threshold, list of energies, execution times, and frequencies
Result: f // optimal frequency

$EDP \leftarrow E \times T$ // compute list of EDP scores
$min \leftarrow 0$
$index \leftarrow 0$
$N \leftarrow length(F)$ // number of frequencies
for $k \leftarrow 1$ **to** N // loop through all freqs
do
  if $EDP_k < min$ **then**
    $min \leftarrow EDP_k$
  end
end
for $i \leftarrow k$ **to** N // loop k to N freqs
do
  $perfDeg_i \leftarrow \frac{maxPerf - perf_i}{maxPerf}$
  if $perfDeg_i < th$ **then**
    $index \leftarrow i$
  end
end
$f \leftarrow F_{index}$ // optimal frequency

Algorithm 1: Optimal frequency determination using EDP

5 EVALUATION

This section evaluates the portability of the proposed models and the efficacy of the selected optimal frequency using real-world applications on NVIDIA GV100 and GA100 GPUs.We used six GPU-enabled real-world applications, including (1) Nanoscale Molecular Dynamics (NAMD) [27, 33], a large biomolecular systems simulation program; (2) Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [28, 38], a particle simulator that models solid-state, soft matter, and coarse-grained materials; (3) GROMACS [14, 34], a software package that performs molecular dynamics simulation; It is fundamentally designed to study bonded interactions in biochemical molecules. (4) Long short-term memory (LSTM) [37] algorithm, a TensorFlow-based implementation of binary sentiment classification of large movie review dataset [24]. (5) BERT (Bidirectional Encoder Representations from Transformers) is a language representation model trained on the movie review dataset, which is the same dataset used for LSTM. (6) ResNet50 (Residual Networks) is a deep learning architecture trained on the CIFAR-10 dataset. For our experiments with NAMD, we used the standard NAMD Apolipoprotein A1 (ApoA1) dataset, which is based on a protein-coding gene with 92,224 atoms [19]. For tests with LAMMPS, we ran a standard Leanard-Jones 3D melt experiment. For GROMACS, we have used a standard water molecule motion simulation, where the algorithm simulates a protein (lysozyme) solution in a box of water.

5.1 Power and Time Models

The DNN-based power and time models, trained using the data from DGEMM, STREAM, and SPEC ACCEL benchmarks, were evaluated using real applications on GV100 and GA100. Figure 7 compares measured and predicted power for each real application across 61 DVFS configurations of GA100. To measure the prediction accuracy of models, we used mean absolute percentage error (MAPE) [32]. The prediction accuracy of the power model for each application was greater than 96% and 94% on GA100 and GV100, respectively,

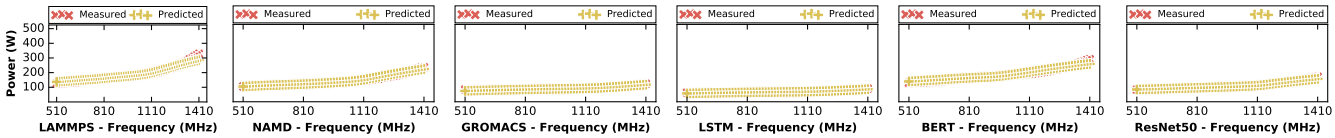


Figure 7: Predicted and measured power consumption for real applications on NVIDIA A100 GPU.

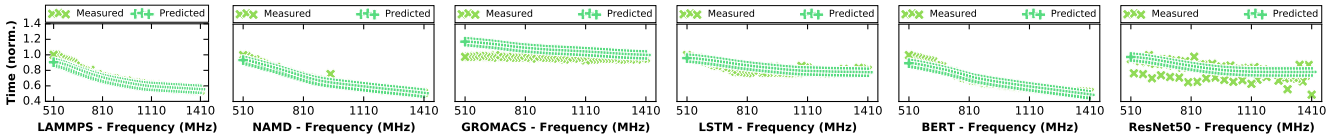


Figure 8: Normalized predicted and measured execution time for real applications on NVIDIA A100 GPU.

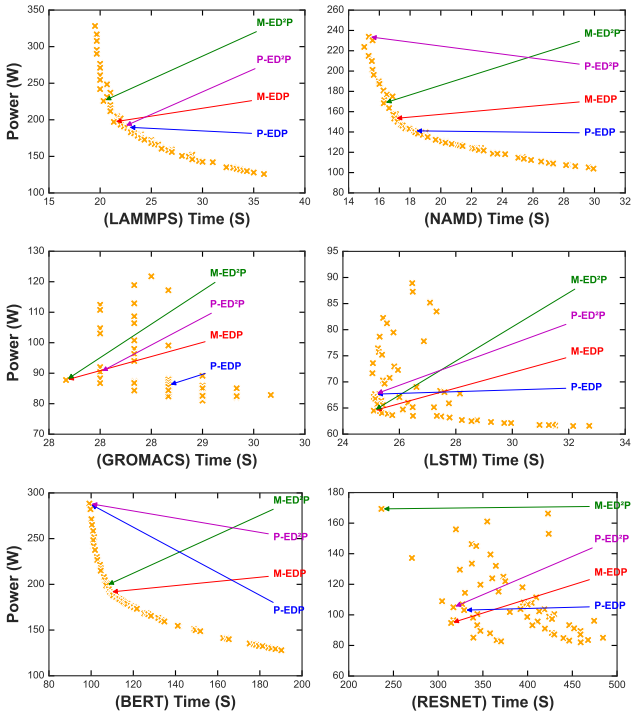


Figure 9: Optimal DVFS configurations selected via EDP approach using measured data (M-EDP), EDP approach using predicted data (P-EDP), ED^{2}P approach using measured data (M-ED^{2}P), and ED^{2}P approach using predicted data (P-ED^{2}P) for the applications shown along with the power usage and execution time for different DVFS configurations on GA100.

as shown in Table 3. We observed that deep learning-based models *outperformed* multi-learners-based models (see Figure 11)

Figure 8 compares measured and predicted execution time for each real application across 61 DVFS configurations of NVIDIA GA100. The prediction accuracy of the performance model for each application was greater than 88% and 90% on GA100 and GV100, respectively, as shown in Table 3. While execution time for GROMACS was predicted with reasonably good average accuracy (i.e., 88.7%),

Table 3: Accuracy of power and performance models for each real application on NVIDIA GA100 and GV100.

GPU	Application	Power	Performance
GA100	LAMMPS	96.5 %	96.2 %
	NAMD	96.8 %	98.1 %
	GROMACS	97.5 %	88.7 %
	BERT	95.7 %	95.9 %
	ResNet50	98.5 %	88.4 %
	LSTM	98.2 %	95.4 %
GV100	LAMMPS	94.9 %	93.4 %
	NAMD	96.5 %	96.5 %
	GROMACS	95.1 %	93.5 %
	BERT	94.5 %	95.9 %
	ResNet50	95.7 %	97.1 %
	LSTM	98.6 %	90.7 %

in Figure 8 (c) execution time is seen to be slightly overpredicted at lower frequencies and underpredicted at higher frequencies. In reality, we observed that the change of DVFS configurations for GROMACS does not impact execution time. We plan to further address applications whose power or performance are not affected by DVFS in future work.

Table 4: The measured ED^{2}P, predicted ED^{2}P measured EDP, and predicted EDP optimal frequencies for real applications.

GPU	Application	M-ED^{2}P	P-ED^{2}P	M-EDP	P-EDP
GA100 (Optimal Frequency (MHz))	LAMMPS	1215	1065	1110	1050
	NAMD	1215	1410	1155	1050
	GROMACS	1110	1140	1110	930
	LSTM	810	1065	810	1065
	BERT	1155	1410	1125	1410
	ResNet50	1410	1020	795	975

5.2 Optimal Frequency Selection

We determined optimal frequencies using EDP and ED^{2}P functions for measured and predicted data (by the proposed deep learning-based models). Figure 9 depicts optimal DVFS configurations selected with EDP for measured (M-EDP) and predicted data (P-EDP), and ED^{2}P for measured (M-ED^{2}P) and predicted data (P-ED^{2}P) for LAMMPS, NAMD, GROMACS, LSTM, BERT, and RESNET50 on

GA100. The power usage and execution time for other DVFS configurations are also shown. Table 4 provides the optimal frequencies for these applications selected via M-EDP, P-EDP, M-ED^{2}P, and P-ED^{2}P on GA100.

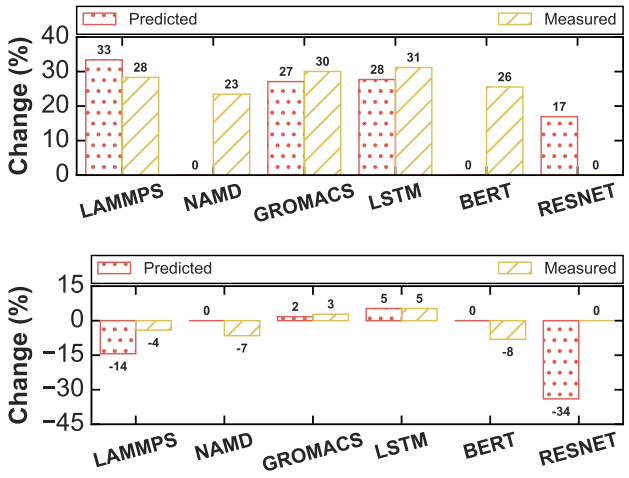


Figure 10: Percentage change in energy (upper) and change in execution time (lower) related to predicted-ED^{2}P, and measured-ED^{2}P for each real application on NVIDIA GA100.

The predicted optimal frequencies selected via P-ED^{2}P for GROMACS, and LSTM on GA100 were higher than their corresponding measured optimal frequencies selected via M-ED^{2}P. These higher values are due to the underprediction of the execution time for these applications at higher frequencies. This underprediction causes the objective functions to choose a higher frequency as optimal. An unintended but favorable consequence of the chosen higher frequencies is less performance degradation, as discussed in Section 5.3. Overall, optimal frequencies for each benchmark's measured and predicted data were less than the maximum core frequency. This observation validates our hypothesis that GPU maximum frequency is not always optimal. We also observed that estimated ED^{2}P optimal frequencies were consistently higher than the estimated EDP optimal frequencies, as expected.

5.3 Performance-Aware Energy Savings

Figures 10 - (a) and (b) show the percentage change in energy and execution time (performance) with ED^{2}P as the objective function on the Ampere GPU. The change in execution time and energy savings for the optimal frequency is calculated with reference to the GPU's maximum setting. It is pertinent to mention that performance degradation can occur even with the measured data using EDP and ED^{2}P approaches (see performance degradation of LAMMPS, NAMD, and BERT on GA100). Overall, the real applications showed that predicted changes in energy savings and execution time closely matched the measured changes in energy savings and time.

The energy performance trade-off using P-EDP, M-EDP, P-ED^{2}P, and M-ED^{2}P approaches for each application is listed in Table 5. The average for each approach is also shown. The results in negative indicate a performance degradation. While P-ED^{2}P predicted maximum frequency for NAMD on GA100 that resulted in no change in execution time and energy savings, P-EDP predicted lower frequency (i.e., 1050 MHz) as shown in Table 4. This trade-off allows saving of 28% energy at the loss of 19.6% performance as shown in Table 5. We observed that the optimal frequency selected for ResNet50 is an outlier. Overall, we observed that greater than 27% energy reduction is possible with a minimal performance degradation of 1.8% for real applications on GA100.

It is seen that LAMMPS and ResNet50 showed higher performance penalties at their optimal frequencies in Table 5. Table 6 shows that performance degradation thresholds can be used to improve the performance penalty. While thresholds limit the DVFS exploration space and can yield no energy savings, it is shown that the performance loss is greatly reduced.

6 RELATED WORK

There have been numerous efforts to improve GPU performance, power, and energy efficiency by developing models and techniques using analytical, machine learning, and static code analysis approaches. Analytical modeling approaches depend on acquiring low-level GPU performance metrics (e.g., voltage).The acquisition of these features is costly and not always feasible in real-world scenarios. Much work focuses on establishing a relationship between performance, power, and energy through static code analysis on GPU assembly instructions [4, 9, 11]. Guerreiro et al. [11] and Fan et al. [8] use Parallel Thread Execution (PTX) assembly code and leverage the Pareto-optimal mechanism to model across DVFS configurations without requiring a prior execution of an application. PTX-based code modeling, while only applicable to PTX workloads, cannot account for memory access patterns and do poorly with model memory-intensive workloads. Further, they are slow and not energy-efficient. While the principal benefit of our approach is the estimation of an application's profile before its execution, there are several other advantages: (1) reliable for both compute- and memory-intensive workloads; (2) portable to CUDA, heterogeneous interface for portability (HIP) and (3) training speed (~7 seconds).

Wu et al. [43] uses performance counters to analyze power and performance. Their study requires an offline collection phase for each application to predictions on subsequent runs, and it does not consider the trade-off between power and performance in their optimal solution. Laros et al. [23] require the acquisition of energy and execution time across different performance states of CPUs and developed a methodology to combine energy and time into a single fused metric called EDP. The energy and performance were combined using a weighting mechanism, which helps design acceptable trade-offs between energy and time metrics. In contrast, our approach selects an optimal frequency using a model-driven approach that requires feature acquisition at the maximum frequency.

To our knowledge, this work is the first to evaluate the efficacy of deep learning-based models to predict power and performance on the state-of-the-art NVIDIA Ampere architecture. This work differentiates from the previous studies on several aspects shown in Table 7. This is also the first to demonstrate the portability of DNN models across architectures with high accuracy. Moreover, unlike the majority of the prior works that have extensively used benchmarks, we use real-world workloads in our evaluation to further demonstrate the effectiveness of our approach.

Table 5: Change in energy and execution time for each application on NVIDIA GA100. Negative values indicate a performance loss, while positives indicate a performance gain.

		Energy (%)				Time (%)			
		M-ED^{2}P	P-ED^{2}P	M-EDP	P-EDP	M-ED^{2}P	P-ED^{2}P	M-EDP	P-EDP
LAMMPS	GA100	28.3	33.4	34.3	32.76	-4.1	-14.4	-9.2	-16.4
NAMD	GA100	23.4	0.0	27.3	28.0	-6.5	0.0	-11.1	-19.6
GROMACS	GA100	30.0	27.1	30.0	28.9	2.8	1.8	2.8	-0.7
LSTM	GA100	31.2	27.7	31.2	27.7	5.3	5.3	5.3	5.3
BERT	GA100	25.5	0.0	27.03	0.0	-8.1	0.0	-9.8	0.0
ResNet50	GA100	0.0	16.9	25.6	15.3	0.0	-34.0	-32.9	-39.0
Average	GA100	28.2	17.5	29.2	22.1	-1.8	-6.9	-9.1	-11.7

Table 6: Change in execution time and energy GA100 with different performance thresholds. At Threshold=Nil, all the frequencies are determined using EDP.

	Threshold=Nil		Threshold=5%		Threshold=1%	
	Time (%)	Energy (%)	Time (%)	Energy (%)	Time (%)	Energy (%)
LAMMPS	-16	33	-4	28	-0.8	10
ResNet50	-39	15	0	0	0	0

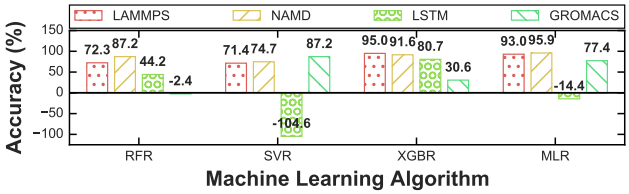


Figure 11: Prediction accuracy for power consumption across different ML algorithms.

Table 7: Comparison with state-of-the-art.

Study	Static	Machine Learning	Real Apps	Multi-Objective
Guerreiro et al. [11]	✓	✓	✗	✗
Fan et al. [8]	✓	✓	✗	✗
Wu et al. [43]	✗	✓	✗	✗
Ali et al. [2, 3]	✗	✗	✓	✓
Our Work	✗	✓	✓	✓

7 DISCUSSION

Modern GPU architectures support a large number of DVFS configurations. This makes designing an energy-efficient solution a non-trivial task requiring understanding performance and power at all supported frequencies. However, data acquisition across all supported GPU frequencies is costly and tedious. One approach to solving this challenge is developing predictive ML models for forecasting power, performance, and energy for unseen (not used in the model training) applications across all supported GPU frequencies. Further, limiting the amount of data required by a model is also desirable. Our feature characterization study helped us to identify low-level GPU utilization metrics that impact the power and execution time of a workload. It is shown that both power and execution time can be accurately modeled using only three features. Also, the identified features are shown not to be largely affected by DVFS and changes in the input size. This dramatically reduces data requirements and allows feature collection to be made only at the default setting. We further confirmed these feature characteristics on the NVIDIA GV100 GPU making our approach portable across architectures. The prediction accuracy of an ML method is also critical in choosing the optimal frequency. To further validate the quality of our models, we compared the accuracy of the power model to several other multi-learner methods – Random Forest Regressor (RFR), eXtreme Gradient Boosting Regressor (XGBR), Support Vector Regressor (SVR), and Multiple Linear Regressor (MLR). These models were again trained with the metrics from DGEMM, STREAM, and SPEC ACCEL benchmarks. Figure 11 shows their power prediction accuracy across different frequencies. In comparison to our model (Table 3), the accuracy is shown to be much lower. Further, the multi-learner approach is inefficient because it requires handling a plethora of different individual learners. Therefore, we decided to use the deep learning method to model the behavior of applications with complex utilization patterns and varying computational intensities.

Workloads with low utilization (e.g., LSTM) showed more energy saving with little to no performance loss. Some workloads (e.g., LAMMPS, NAMD) showed performance degradation using EDP and ED^{2}P even with measured data. Such applications are best left untouched to run at the maximum frequency in scenarios where performance is important. Ultimately, the quality of the objective function determines the power-performance trade-off. Compared to EDP, ED^{2}P showed significant improvement in performance with minimal impact on energy savings, making it better suited for HPC centers where performance is paramount.

8 CONCLUSIONS AND FUTURE WORK

Future supercomputer scale and performance will be determined by how efficiently power budgets are managed. Consequently, it is imperative to develop effective power management solutions for GPUs where a large portion of performance will reside. The practicality concerns and infeasibility of brute-force approaches in dealing with the large DVFS exploration space motivated us to explore DNN-based models for predicting power and execution time on GPUs. GPU utilization metrics that influence power and time were identified using the mutual information technique. These features are largely unaffected by frequency and input size changes, allowing for their acquisition at the default setting, significantly reducing the data required by the models. Our models (power and time) achieved accuracies between 89% and 98% for the four real-world applications – LAMMPS, NAMD, GROMACS, LSTM, BERT, and ResNet50 on the state-of-the-art NVIDIA GA100 as well as the previous generation GV100 GPU. It is shown that the power-performance trade-off depends on the quality of the objective functions (i.e., EDP, ED^{2}P). The objective functions used

in this study can achieve energy savings of up to 27.1% on GA100 with negligible performance degradation of 1.8%. Ultimately, we have demonstrated the viability of using DNN-based models for effective GPU power control. Moreover, we show that DNN-based models are a better fit than multi-learner models for modeling GPU power and time of complex applications. In the future, we plan to evaluate the voltage design space using the proposed methodology on GPUs supporting change of voltage configuration.

ACKNOWLEDGMENTS

The National Energy Research Scientific Computing Center (NERSC) is a U.S. Department of Energy Office of Science User Facility operated under Contract No. DEAC02-05CH11231. Results presented in this paper were obtained using the Chameleon testbed supported by the NSF. This research is supported in part by the National Science Foundation under grants CNS-1817094, OAC-1835892, and CNS-1939140 (A U.S. National Science Foundation Industry-University Cooperative Research Center on Cloud and Autonomic Computing). The authors thank Mathew Colgrove (NVIDIA) for his assistance in SPEC ACCEL, Alan Sill (TTU) for discussing, and Victor Sheng (TTU) for their comments on modeling. We are also very grateful to the High Performance Computing Center of Texas Tech University for providing HPC resources for this research.

REFERENCES

[1] Yuki Abe et al. 2014. Power and performance characterization and modeling of GPU-accelerated systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 113–122.

[2] Ghazanfar Ali, Sridutt Bhalachandra, Nicholas J Wright, Mert Side, and Yong Chen. 2022. Optimal GPU Frequency Selection using Multi-Objective Approaches for HPC Systems. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.

[3] Ghazanfar Ali, Mert Side, Sridutt Bhalachandra, Nicholas J Wright, and Yong Chen. 2023. Optimal GPU Frequency Selection using Multi-Objective Approaches for HPC Systems, In Future Generation Computer Systems. *Future Generation Computer Systems*. ACCEPTED.

[4] Lorenz Braun et al. 2020. A simple model for portable and fast prediction of execution time and power consumption of GPU kernels. *ACM Transactions on Architecture and Code Optimization (TACO)* 18, 1 (2020), 1–25.

[5] Jack Choquette et al. 2021. 3.2 the A100 datacenter GPU and Ampere architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 48–50.

[6] Tom Deakin et al. 2016. GPU-STREAM v2. 0: Benchmarking the achievable memory bandwidth of many-core processors across diverse parallel programming models. In *International Conference on High Performance Computing*. Springer, 489–507.

[7] Bishwajit Dutta et al. 2018. GPU power prediction via ensemble machine learning for DVFS space exploration. In *Proceedings of the 15th ACM International Conference on Computing Frontiers*. 240–243.

[8] Kaijie Fan et al. 2019. Predictable GPUs frequency scaling for energy and performance. In *Proceedings of the 48th International Conference on Parallel Processing*. 1–10.

[9] TJ Florindo et al. 2018. Application of the multiple criteria decision-making (MCDM) approach in the identification of Carbon Footprint reduction actions in the Brazilian beef production chain. *Journal of Cleaner Production* 196 (2018), 1379–1389.

[10] Ricardo Gonzalez and Mark Horowitz. 1996. Energy dissipation in general purpose microprocessors. *IEEE Journal of solid-state circuits* 31, 9 (1996), 1277–1284.

[11] João Guerreiro et al. 2019. GPU static modeling using PTX and deep structured learning. *IEEE Access* 7 (2019), 159150–159161.

[12] João Guerreiro et al. 2019. Modeling and decoupling the GPU power consumption for cross-domain DVFS. *IEEE Transactions on Parallel and Distributed Systems* 30, 11 (2019), 2494–2506.

[13] Stijn Heldens et al. 2020. The Landscape of Exascale Research: A Data-Driven Literature Analysis. *ACM Computing Surveys (CSUR)* 53, 2 (2020), 1–43.

[14] Berk Hess and other. 2008. GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of Chemical Theory and Computation* 4, 3 (2008), 435–447.

[15] HPCC. 2020. *High Performance Computing Center*. Retrieved May, 2020 from http://www.depts.ttu.edu/hpcc/

[16] Shadi Ibrahim et al. 2014. Towards Efficient Power Management in MapReduce: Investigation of CPU-Frequencies Scaling on Power Efficiency in Hadoop. In *Adaptive Resource Management and Scheduling for Cloud Computing*. Springer International Publishing, Cham, 147–164.

[17] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. 1996. Artificial neural networks: A tutorial. *Computer* 29, 3 (1996), 31–44.

[18] Guido Juckeland et al. 2014. SPEC ACCEL: A standard application suite for measuring hardware accelerator performance. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 46–67.

[19] Neha Kashyap. 2016. HPC Benchmarks and Applications Performance Study on Broadwell-EP 4S Processor. https://downloads.dell.com/manuals/all-products/esuprt_software/esuprt_it_ops_datcentr_mgmt/high-computing-solution-resources_white-papers59_en-us.pdf.

[20] Kate Keahey et al. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.

[21] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. *Advances in neural information processing systems* 30 (2017).

[22] Alexander Kraskov et al. 2004. Estimating mutual information. *Physical review E* 69, 6 (2004), 066138.

[23] James H Laros III et al. 2013. Energy delay product. In *Energy-Efficient High Performance Computing*. Springer, 51–55.

[24] Andrew L. Maas et al. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150. http://www.aclweb.org/anthology/P11-1015

[25] Vishwas Mishra and Shyam Akashe. 2015. Calculation of Power Delay Product and Energy Delay Product in 4-Bit FinFET Based Priority Encoder. In *Advances in Optical Science and Engineering*. Springer, 283–289.

[26] NVIDIA Corporation. 2013. *CUDA Samples*. https://docs.nvidia.com/cuda/cuda-samples/index.html#matrix-multiplication--cublas-

[27] NVIDIA Corporation. 2020. NGC NAMD Container. https://ngc.nvidia.com/catalog/containers/hpc:namd.

[28] NVIDIA Corporation. 2021. NGC LAMMPS Container. https://ngc.nvidia.com/catalog/containers/hpc:lammps.

[29] NVIDIA Corporation. 2021. NVIDIA DCGM. https://developer.nvidia.com/dcgm

[30] Oak Ridge National Laboratory. 2022. Exascale, Project. https://www.exascaleproject.org/wp-content/uploads/2021/12/webinar-WrongWay-220216.pdf.

[31] Junyoung Park and Jacob A Abraham. 2011. A fast, accurate and simple critical path monitor for improving energy-delay product in dvs systems. In *IEEE/ACM International Symposium on Low Power Electronics and Design*. IEEE, 391–396.

[32] F. Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[33] James C Phillips et al. 2020. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *The Journal of Chemical Physics* 153, 4 (2020), 044130.

[34] Sander Pronk et al. 2013. GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* 29, 7 (2013), 845–854.

[35] Brian C Ross. 2014. Mutual information between discrete and continuous data sets. *PloS one* 9, 2 (2014), e87357.

[36] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. 1997. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems* 39, 1 (1997), 43–62.

[37] TensorFlow. 2021. Long Short-Term Memory layer - Hochreiter 1997. https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM.

[38] Aidan P. Thompson et al. 2021. LAMMPS - A flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications* (2021), 108171. https://doi.org/10.1016/j.cpc.2021.108171

[39] Tijmen Tieleman and Geoffrey Hinton. 2012. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn* (2012).

[40] TOP500.org. 2022. Top500, June 2022 Ranking. https://www.top500.org/lists/top500/2022/06/.

[41] Qiang Wang and Xiaowen Chu. 2020. GPGPU performance estimation with core and memory frequency scaling. *IEEE Transactions on Parallel and Distributed Systems* 31, 12 (2020), 2865–2881.

[42] HPC Wire. [n. d.]. AMD's MI300 APUs to Power Exascale El Capitan Supercomputer. <https://www.hpcwire.com/2022/06/21/amds-mi300-apus-to-power-exascale-el-capitan-supercomputer. (Accessed on 10/06/2022).

[43] Gene Wu et al. 2015. GPGPU performance and power estimation using machine learning. In *21st International Symposium on High Performance Computer Architecture*. IEEE, 564–576.