Dynamic Energy Optimization in Chip Multiprocessors Using Deep Neural Networks

Milad Ghorbani Moghaddam, *Student Member, IEEE,* Wenkai Guan, *Student Member, IEEE,* and Cristinel Ababei, *Senior Member, IEEE*

Abstract—We investigate the use of deep neural network (DNN) models for energy optimization under performance constraints in chip multiprocessor systems. We introduce a dynamic energy management algorithm implemented in three phases. In the first phase, training data is collected by running several selected instrumented benchmarks. A training data point represents a pair of values of cores' workload characteristics and of optimal voltage/frequency (V/F) pairs. This phase employs Kalman filtering for workload prediction and an efficient heuristic algorithm based on dynamic voltage and frequency scaling. The second phase represents the training process of the DNN model. In the last phase, the DNN model is used to directly identify V/F pairs that can achieve lower energy consumption without performance degradation beyond the acceptable threshold set by the user. Simulation results on 16 and 64 core network-on-chip based architectures demonstrate that the proposed approach can achieve up to 55% energy reduction for 10% performance degradation constraints. In addition, the proposed DNN approach is compared against existing approaches based on reinforcement learning and Kalman filtering and found that it provides average improvements in energy-delay-product (EDP) of 6.3% and 6% for the 16 core architecture and of 7.4% and 5.5% for the 64 core architecture.

Index Terms—chip multiprocessors; energy optimization; Kalman filter; reinforcement learning; deep neural network

1 Introduction

The evolution of the internet and the emergence of mobile devices have created an environment where we interface computing continuously. Much of the computations (e.g., web searches, e-mail services, social networks, etc.) consumed by this emerging market are done by chip multiprocessors (CMP) in massive datacenters also called warehouse scale computers (WSCs). In 2013, U.S. datacenters consumed an estimated 91 billion kilowatt-hours of electricity, enough to power twice the households in New York City. By 2020, estimated consumption will increase to 140 billion kilowatt-hours, costing American businesses $13 billion per year in electricity bills and causing the emission of nearly 150 million metric tons of carbon pollution annually [1]. According to the U.S. Energy Information Administration, that is about 7% of total commercial electric energy consumption and it is projected that this number will increase [2]. Improving the efficiency of WSCs has been identified as one of the top priorities of web-service companies as it improves the overall total cost of ownership of WSCs. As noted by the Environmental Protection Agency, improving efficiency is not only important for the cost to companies, but for the environmental footprint of these WSCs as this computing domain rapidly expands [3].

Therefore, there is a strong motivation to seek new methods to reduce energy consumption in these WSCs. In this paper, we propose a new dynamic energy management (DEM) method to reduce energy consumption in future chip multiprocessors with 16 and 64 cores, that are projected to be increasingly used in WSC servers. The proposed method is based on dynamic voltage and frequency scaling (DVFS) and on machine learning theories that reveal unprecedented prediction success. Specifically, we propose to use DNN models and develop related self-adaptive supervised learning methods to identify optimal V/F pairs in chip multiprocessor systems. We see machine learning techniques, such as the one proposed here, as a potentially new enabler in pushing the frontier of energy optimization in CMP systems because they are known to have the ability to capture complex relations between input features and output labels. Researchers in machine learning attribute the immense success of DNN models in domains such as speech recognition, image processing, pattern recognition, etc. in the last decade to this ability.

The remainder of this paper is organized as follows. In the next section, we discuss related literature. Then, we present background information on neural networks as well as on a Kalman filtering based DVFS technique, which we use during the process of collection of training data. The proposed energy optimization method is then presented in section 5. In section 6, we report and discuss simulation results. We summarize our findings in the conclusion section 7.

2 Related Work

Energy optimization in single and multicore processors received a lot of attention in previous literature. The most popular techniques utilized by previous optimization solutions include DVFS and task migration. These techniques are used as primary control mechanisms to drive the operation of processors toward low energy consumption such

• *M.G. Moghaddam, W. Guan, and C. Ababei are with the Department of Electrical and Computer Engineering, Marquette University, Milwaukee WI, 53233.*
E-mail: milad.ghorbanimoghaddam; wenkai.guan; cristinel.ababei @marquette.edu

Manuscript received November 30, 2017; revised May 15, 2018.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2870438, IEEE Transactions on Multi-Scale Computing Systems

that performance is not significantly degraded. The control decisions are made based on estimations or predictions of the energy or other related variables in a reactive or proactive manner as part of the algorithm that implements the optimization solution. System monitoring and decision making are usually done periodically, at intervals called control periods or epochs. It is the prediction mechanism that differentiates the impact of a given energy optimization solution. For brevity, we limit ourselves to mainly discussing previous work that employed techniques that fall in the general area of machine learning.

The study in [4] proposes a multinomial logistic regression-based classification technique, that classifies the workload at runtime, into a fixed set of classes, which are then utilized to design a DVFS algorithm. In [5], a multinominal logistic regression classifier is built using a large volume of performance counters for offline workload characterization. This classifier is queried at run-time for a given application to predict the workload, and then selection of the frequency and thread packing are done to maximize performance under a given power budget. The techniques in [6], [7], [8] use online learning to select the most appropriate frequency for the processing cores based on the workload characteristic of a given application. The study in [10] uses supervised learning in the form of a Bayesian classifier for processor energy management. This framework learns to predict the system performance from the occupancy state of the global service queue. The predicted performance is then used to select the frequency from a pre-computed policy table. Reinforcement learning (RL) based optimization algorithms are proposed in [11], [12], [13], [14]. For example, the study in [13] used RL to learn the relationship between the mapping of threads to cores, clock frequencies, and temperatures, and employed that mapping information to develop better task mapping and DVFS solutions. The work in [11] used RL to learn the optimal control policy of the V/F levels in manycores and then exploited that to develop an efficient global power budget reallocation algorithm. The authors of [14] proposed an online DVFS control strategy based on core-level modular reinforcement learning to adaptively select appropriate operating frequencies for each individual core. Q-learning was used by the work in [15] to develop an algorithm that identifies V/F pairs for predicted workloads and given application performance requirements. In the context of dynamic VFI control in manycore systems with different applications running concurrently, the study in [16] investigated imitation learning and reported higher quality policies.

The studies in [17], [18] predicted workload in CMPs using Kalman filtering and long short term memory (LSTM) models. The predictions are then used inside efficient heuristics to identify V/F pairs for each CMP core in order to reduce energy consumption under performance constraints. The authors of [19] develop an artificial neural network (ANN) based mechanism for network-on-chip (NoC) power management. The offline training of the ANN is augmented with a simple proportional integral (PI) controller as a second classifier. The ANN is used to predict the NoC utility, which is then used to make DVFS decisions that lead to improvements in the energy-delay product. A neural network (NN) based model with eight outputs for different

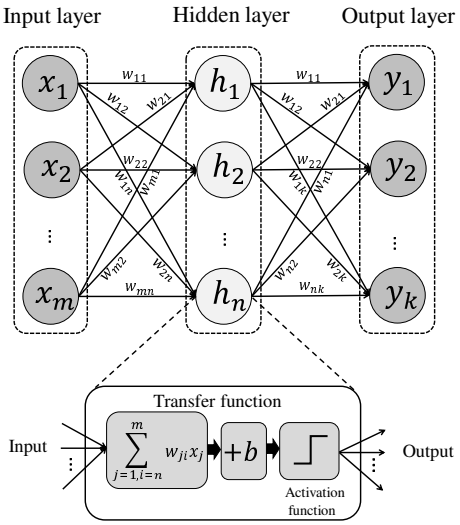


Fig. 1. Typical neural network architecture.

interface configurations of a mobile device was presented in [9] to do classification. Such classification is used as the basis for setting the mobile device into the configuration state that reduces energy consumption. It was reported that NN and support vector machine (SVM) models provided the best prediction accuracy. In particular, NN and k-nearest neighbor (KNN) based solutions outperformed the logistic regression based solution. The study in [20] proposed a DNN model to model plant performance and to predict power usage effectiveness (PUE) in datacenters. Testing and validation at Google's datacenters showed that the DNN model can be an effective approach to exploit existing sensor data to model datacenter performance and to identify operational parameters that improve energy efficiency and reduce the PUE [20].

While there has been significant work, it is not clear how far the existing DVFS based energy optimization techniques are from the optimal solutions. We believe there is still room for improvement, and generally, we see this as the only limitation of previous works. As such, our main motivation for this work is the need to investigate whether DNN models can be of any help in pushing the frontier of energy optimization in chip multiprocessors. This idea in turn is motivated by the immense success that DNN models had in the last decade in many application domains including speech and pattern recognition, image processing, and datacenter operation. Our comprehensive simulation experiments on sixteen benchmarks show that DNN models can indeed provide improvements over existing approaches. That is the main contribution of this paper.

3 Background on Neural Networks

The simplest and most popular NN architecture is the feed-forward neural network, which is illustrated in Fig. 1. The information in this network is transferred from one layer to the next in the forward direction only and no cyclic connections exist between layers. Each node represents a neuron that receives its weighted inputs from the nodes on the previous layer and calculates the output (i.e., decision)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2870438, IEEE Transactions on Multi-Scale Computing Systems

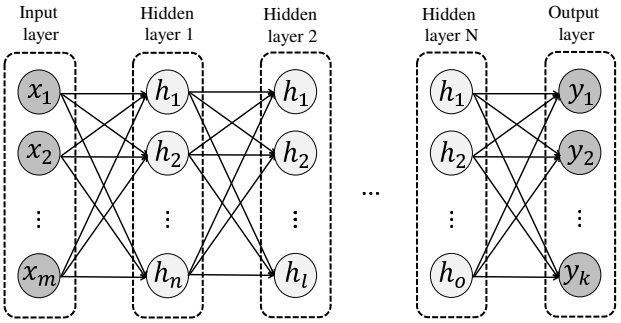


Fig. 2. A deep neural network is a neural network with many hidden layers.

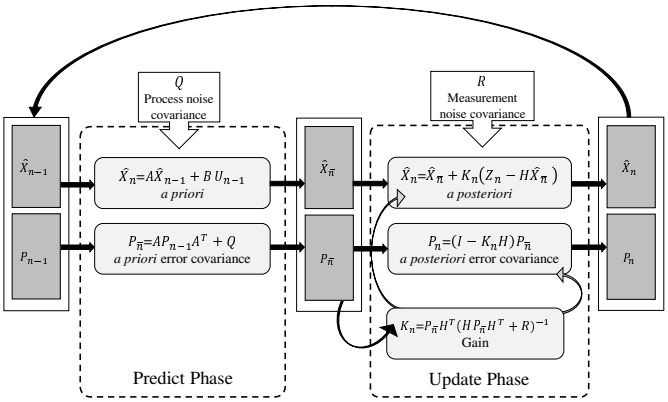


Fig. 3. Block diagram of the Kalman filtering control loop used to evaluate and reduce the estimation error.

that is passed to the next layer. The transfer function of the node sums together all the decisions from the nodes in the previous layer and adds them to a bias value. The result then is passed through an activation function to generate the output. This process takes place in the forward direction through all layers up to the output layer, which produces the final output decisions. The values of weights and biases are crucial as they affect the accuracy of the final decision. These values are determined during the training process of the network. In supervised training, for a set of known features and labels (i.e., inputs and their corresponding output decisions), the final decisions produced by the NN model are compared to the labels by means of a cost function. Then, an optimizer is employed to minimize the generated cost by updating the weights through the network going in the backward direction as a backpropagation process. Usually, the optimizer uses a gradient descent optimization approach [21]. The training process is repeated on different sets of features and labels, thereby determining the optimized weights and biases. Once trained, the NN model can be utilized to provide estimations on new data of interest. That is, the outputs of the final layer can be used directly for classification purposes. Structurally, a DNN model is just a feed-forward neural network with many hidden layers [22], as illustrated in Fig. 2. The main difference compared to traditional NNs is that DNNs have more hidden layers. That helps DNNs to capture more complex nonlinear relationships [23].

4 Energy Optimization using Kalman Filtering for Workload Prediction and DVFS

In this section, we briefly describe the dynamic energy management algorithm from [17] because it serves as the basis for implementing the first phase (described later) of the proposed energy optimization approach in this paper.

4.1 Kalman Filtering as Prediction Technique

The Kalman filter is an algorithm applied to predict the state x in a discrete-time controlled process. It uses a set of recursive equations as well as a feedback control mechanism to minimize the variance of the estimation error [24]. The process can be described by the following state and output equations, using the notation from [25]:

$$x_n = Ax_{n-1} + Bu_{n-1} + w_{n-1} \tag{1}$$

$$z_n = Hx_n + v_n \tag{2}$$

where A is the state transition model applied to the previous state x_{n-1} at time steps $n-1$ and n, in the absence of control input or process noise. B is the optional control input model applied to the control vector u, and the matrix H relates the state x to the measurement or observation z. The random variable w_{n-1} models the process noise assumed to be a white Gaussian noise with zero mean and covariance Q, $w \sim N(0,Q)$. Similarly, the random variable v_n is the measurement noise also assumed to have a Gaussian distribution with zero mean and covariance R, that is independent from Q, $v \sim N(0,R)$.

A Kalman filter is constructed in two phases (see Fig. 3). The first phase is called the *predict phase* and also called the time update phase. Here, the state x is predicted a priori as $\hat{x}_n^-$. The second phase is called the *update phase* and also called the measurement update phase. This is where the predicted $\hat{x}_n^-$ is updated a posteriori as $\hat{x}_n$. In the predict phase, the filter uses the previous state $\hat{x}_{n-1}$ and the input u_{n-1} to project the state. It also uses the error covariance of the a posteriori error P_{n-1} and the process noise covariance Q to project the error covariance P_n^- for the a priori error. The two equations used in this phase are:

$$\hat{x}_n^- = A\hat{x}_{n-1} + Bu_{n-1} \tag{3}$$

$$P_n^- = AP_{n-1}A^T + Q \tag{4}$$

The update phase begins after the predict phase with the measurement of the actual state value at time n. It first computes the Kalman gain K_n. K_n is chosen to minimize P_n. Then, the current state matrix $\hat{x}_n$ and P_n are updated. The three equations utilized in this phase are:

$$K_n = P_n^- H^T (HP_n^- H^T + R)^{-1} \tag{5}$$

$$\hat{x}_n = \hat{x}_n^- + K_n(z_n - H\hat{x}_n^-) \tag{6}$$

$$P_n = (1 - K_n H)P_n^- \tag{7}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2870438, IEEE Transactions on Multi-Scale Computing Systems

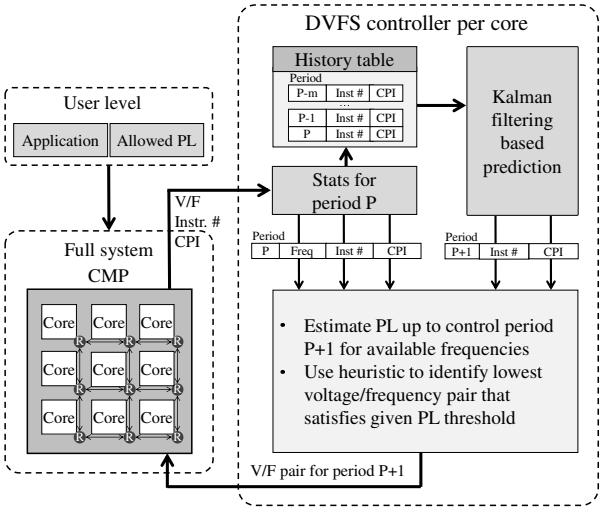


Fig. 4. Diagram of the energy optimization algorithm from [17] constructed as a combination of DVFS and Kalman filtering based prediction techniques.

where R is the measurement noise covariance. H relates the observation or measurement z to the state x.

In the context of dynamic voltage scaling for MPEG applications, the study in [25] proposed an extended Kalman filter to estimate the processing time of workloads. Also in the context of high performance processors, the authors of [26] proposed a sparse Kalman filter to estimate the states of a dynamical network system. They then applied their solution to the thermal model network of manycore processors to solve the problem of finding the minimum number of in-situ sensors that can be used for both thermal profile estimation and tracking of hotspots in dynamic thermal management solutions. Recently, we used a similar Kalman filtering technique to estimate the average cycles per instruction (CPI) and the instruction count inside a method for dynamic energy management for chip multiprocessors with 16 and 64 core architectures [17]. We found that the Kalman filtering based predictions are very accurate and allow the proposed energy reduction heuristic to provide consistent energy savings under a given performance constraint for all benchmarks that we investigated. We describe this in the next subsection.

4.2 Energy Optimization using DVFS

We assume that the execution of a given benchmark is split into consecutive control periods and that the energy optimization algorithm is applied at the end of each such period. This is the case of the study in [17], which relies on performance loss estimations that are calculated at the end of each control period for each core of the CMP. A Kalman filtering based approach is employed to predict the workload in the next control period for which V/F pairs must be selected and set. This selection is done with a DVFS based heuristic algorithm whose objective is to reduce energy consumption but without degrading performance beyond a user set threshold.

The idea behind the optimization method in [17] is to predict the workload in the next control period and then find the lowest V/F pair for each core so that the execution of the predicted workload will not violate a predetermined performance degradation threshold. To facilitate that, the concept of the performance loss (PL) that is incurred over all the control periods was introduced, which can be calculated by the following expression [17]:

$$PL = \sum_{P=1}^{N} \frac{I_{P_{Done}} \times (\frac{CPI_P(\frac{f_H}{f_P}-1)}{f_H})}{T} \quad (8)$$

where:

N: Total number of control periods
f_H: Highest available CPU frequency
$I_{P_{Done}}$: Number of instructions done in period index P
CPI_P: Average CPU cycles per instruction in period P
f_P: CPU clock frequency in period P
T: Duration of the control period

The expression in equation (8) provides an estimation technique for the performance loss incurred due to the application of DVFS compared to the case when no DVFS were applied at all and the CMP would be kept running at the highest V/F level.

The overall energy optimization algorithm is constructed as a heuristic algorithm that uses DVFS in combination with the Kalman filtering based prediction. The block diagram is shown in Fig. 4 as implemented inside a custom Sniper based system simulator. The algorithm is fed by the periodic statistics (i.e., number of instructions executed by each core and CPI) during a regular simulation of a given application. The statistics are recorded for a moving window of m past control periods and utilized to make predictions about the next control period instruction count and CPI using the Kalman filtering technique. Then, the algorithm uses the predictions to estimate the performance loss using the expression from equation (8) for available frequencies and to decide the best V/F pairs for all cores for the next control period. The V/F pairs are selected to maximize energy savings but without violation of the performance loss constraint set by the user.

To summarize, essentially, the Kalman filtering makes predictions of the average cycles per instruction and the instruction count (these represent the workload of each core in the next control period), based on which then, equation (8) estimates the performance loss, which finally, at its turn is used to decide about V/F pairs for the next control period.

5 Energy Optimization Using DVFS and DNN Based Prediction

5.1 Top-level Description

The main idea of the dynamic energy optimization approach proposed in this paper is to use DNN models for prediction or classification. Note that, as in the case of many other application domains including speech recognition, pattern recognition, and recommending systems that have been revolutionized lately by the use of DNN models, the merit of this work lies in the application of the DNN model to a specific practical problem rather than the DNN model itself, which has been known for decades already.

The system level block diagram of the proposed optimization framework is shown in Fig. 5, as it is implemented in our Sniper based full system simulation tool

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2870438, IEEE Transactions on Multi-Scale Computing Systems

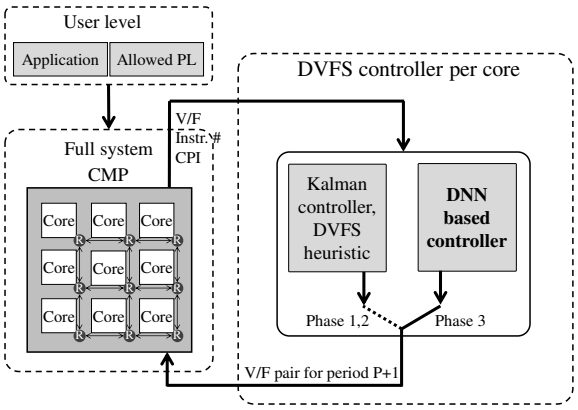


Fig. 5. The proposed dynamic energy optimization algorithm switches to DNN based prediction once the DNN model has been constructed. The Kalman filtering based controller block operates similarly to that in Fig. 4.

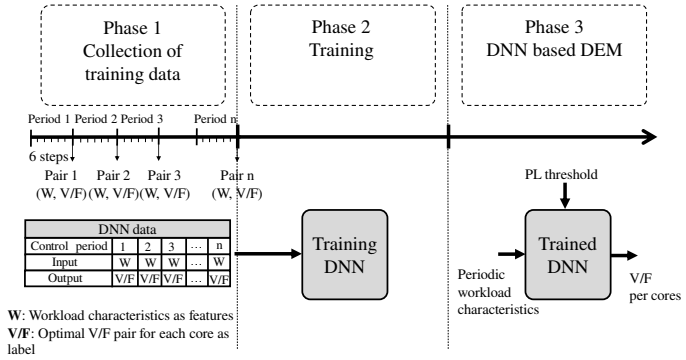


Fig. 6. Illustration of the three phases of the implementation and usage of the DNN model.

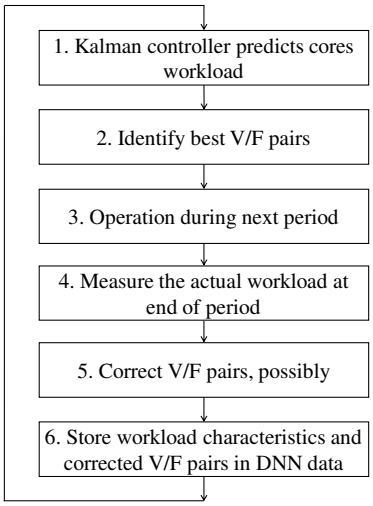


Fig. 7. Steps of the procedure to generate one training data point during one control period in Phase 1.

discussed later in the paper. The proposed framework is implemented mainly in software and is responsible for managing all related activities, including creating, maintaining, and storing specific information about the *DNN controller*. The information about the DNN topology, related weights, as well training data represents what is denoted as *DNN data*. The primary objective of the proposed dynamic energy optimization approach is to reduce the energy consumption of the CMP. This can be achieved by throttling CMP core frequencies to the lowest possible V/F levels while meeting as much as possible the execution deadline of all executed tasks. Two of the key elements of the optimization framework include 1) the *DNN controller* with its associated *Kalman controller* and self-learning technique and 2) the DVFS algorithm that decides the V/F levels for each of the cores for the next control period.

The *Kalman controller* is implemented with the help of a series of Kalman filters and works with a sliding window of m previous control periods. Thus, in generating a training data pair, we consider past history covering the last m control periods. We choose to use the Kalman technique due to both the ease of implementation and the very good performance demonstrated in workload estimation. In our previous work, the Kalman technique provided excellent results for workload estimation. In addition, having in place an existing approach for dynamic energy optimization provides a way to achieve energy reductions also during the first two phases of the proposed approach, when we collect training data and train the DNN model.

The implementation of the DNN model based energy optimization algorithm includes three phases as illustrated in Fig. 6. In the first phase, we collect input samples (i.e., input features) and their corresponding outputs (i.e., labels) as the initial training data set. The features capture the benchmark behavior and the labels represent the V/F pairs identified to lead to energy reduction. In the second phase, the training data is used to train the DNN model. Lastly, in the last phase, the DNN model is employed to directly predict optimal V/F pairs for each CMP core for given workload at runtime. These phases are described in more details next.

5.2 Phase 1: Collection of Training Data

One of the main challenges of working with DNN models is training. This is a two-faceted challenge: first, labeled data is necessary for training and second, the training process may become computationally intensive and require long training times for increasingly large training data sets. In addition, workloads can vary greatly and developing a representative training data set is very difficult because a DNN well trained for certain workloads may perform very poorly on different workloads. To address the lack of training data when it comes to chip multiprocessors, we propose to use a new self-adaptive supervised training technique. We develop the ability to generate training data automatically in three phases as illustrated in Fig. 6.

Phase 1 begins when a new CMP system starts to be used in a datacenter. This is the time when the CMP operation starts to be monitored for the purpose of generating input-output training data pairs in Fig. 6. The generation of training data is done for each control period. For each such control period we generate, at the end of the period, training data pairs by recording input values and the *corrected* outputs (as V/F levels) which would have been better if set at the beginning of the control period. The six steps followed to generate the corrected V/F pairs in a given control period

are shown in Fig. 7.

1) The *Kalman controller* is used to make predictions about the workload of each CMP core in the next control period.
2) Then, equation (8) is used to estimate the performance loss and to identify the lowest available V/F pairs at which energy could be saved without violating the performance loss threshold in the next control. The selection of these V/F pairs is done with the efficient DVFS heuristic algorithm from [17].
3) Proceed with the execution of the next control period at selected V/F pairs for all CMP cores.
4) At the end of the just executed control period, measure the actual just executed workload.
5) Repeat Step 2 but use the actual measured workload to find possible corrections to the just used V/F pairs. The corrected V/F pairs are the ones that ideally should have been identified earlier in Step 2. These corrected V/F pairs are used as outputs in the training data set because they would have helped reduce energy without violation of the performance degradation threshold.
6) The above steps are done for a moving window of m control periods in order to generate one training data point of input features and corrected V/F pairs added to the DNN data. These will be used later for the actual training of the DNN model.

It is important to note that, in theory, one could conduct the whole process of collecting training data with a set-up that does not use the Kalman filtering based prediction combined with the DVFS heuristic. Instead, one could just run the selected benchmark testcases at the default (highest frequency and voltage pair) all the time during Phase 1. While this would eliminate the need for the Kalman filtering and simplify the overall implementation of the proposed framework, the issue would be that the training data would not be diverse and would always have as input features values that would characterize core operations at the maximum frequency all the time. When the Kalman filtering based technique is used, training data points are generated also for input features that characterize core operations at throttled frequencies as well. Therefore, the training data resulting from the proposed approach is more diverse and characterizes better the operations of all cores (among the 16 or 64) at many different clock frequencies. In addition, as already mentioned, the Kalman filtering based technique provides an alternative way to achieve energy reductions also during the first two phases of the proposed approach.

5.3 Phase 2: Training of the DNN Model

By this time, we have collected runtime statistics and constructed the training data set. Instruction count and average CPI values together with the corrected V/F pairs have been recorded as the *features* and the *labels* of the DNN data characterizing all the control periods of Phase 1. Note that the labels are transformed into the *one hot* format before actual use. In the *one hot* format, for each class in the output we consider a digit which can be zero or one. If the label belongs to class number k, the k-th digit is set to "1" while the other digits are set to "0". The collected training data set is now used for supervised training of the proposed DNN model. The input features are passed to the feed-forward model. At each node, the weights and biases are applied to the given inputs and then the result gets activated through an activation function. We use the *RELU* function (like that shown in Fig. 1) as the activation function because it helps to mitigate the vanishing gradient problem described in [27]. The final result of the output layer is used to calculate the cost. That is, the *cross entropy cost function* compares the generated output with the stored labels (recall, these are the corrected V/F pairs) to calculate the cost based on the prediction error. The *gradient decent optimizer* uses this cost to optimize the weights and biases in the backward direction. Specifically, as the gradient decent optimizer algorithm we use the *AdaGrad* method, which was shown to give the best results [21]. This method adapts the learning rate to the model parameters and performs larger updates for infrequent parameters and smaller updates for frequent ones. Thus, it is well suited for dealing with sparse data, which we see in our case.

5.4 Phase 3: Prediction Using the DNN Model

Now, that we have trained the DNN model as the *DNN controller* from Fig. 5, we can use it in realtime to identify V/F pairs at any time. Note that the same trained DNN model is replicated as many times as cores in a given architecture and used individually. This is the phase where the role of making predictions and deciding the V/F pairs is switched from the *Kalman controller* and the DVFS heuristic to the *DNN controller*. Collection of training data can still be performed in parallel, in order to prepare for future periodic retraining of the DNN model to address application variability. However, we do not do this in this paper.

6 Simulation Results

6.1 Experimental Setup

We leverage existing simulation tools and develop an in-house full system simulation framework inside which we have implemented all the algorithms described in this paper. Specifically, we use the Sniper system simulator [28] integrated with the McPAT power estimator [29]. Our simulation framework has implemented three energy optimization approaches: the reinforcement learning (RL) approach described in [14], the Kalman filtering approach from [17], and the proposed DNN model based approach. This makes the collection of simulation results easier and all the comparisons consistent because all simulations are done within the same simulation tool and on exactly the same benchmarks. The simulation framework includes all the functions to implement the three phases of the proposed energy optimization approach. We implemented the RL algorithm as it is described in Algorithm 1 of [14]. The number of modules is equal to the number of cores of the CMP architecture. Similarly to [14], each state is defined as a 2D tuple $s_t = (h_t, \mu_t)$, where $h_t = num\_busy\_cycles_t / time\_elapsed_t$ is the core throughput in terms of busy-cycle-count per unit time and $\mu_t = num\_cycles\_stalled_t / num\_busy\_cycles_t$ is

TABLE 1
Architectural configuration parameters.

Parameter	Value
Technology node	45nm
Core	Intel X86 Gainstown
Core CPU model	Out of order (Detailed CPU)
Frequencies(f)	2GHz downto 1GHz, with 100MHz step
VDDs	[f>=1.8G:1.2V],[1.8G>f>=1.5G:1.1V],[1.5G>f>=1G=1V]
Cores/socket	1
Transition latency	2000 ns
Branch predictor	2 bit counter
Reorder buffer	80-entries
L1ICache/1core	32KB
L1DCache/1core	64KB
L2/1core	256KB
L3/4cores	8MB
Network	2D regular mesh, 1 router per core
Link bandwidth	64 bits

the so-called CPU intensiveness. As in [14], the reward was selected so that improvement in the energy-delay-product is encouraged. For training the DNN model, we employ Google's Tensorflow machine learning library [30]. All simulations are conducted on a Linux Ubuntu 16.04 machine that runs on an Intel Xeon eight core processor equipped with a K40c Tesla GPU.

We conduct simulations on Parsec and Splash2x benchmarks [31] and provide comparisons against both RL and Kalman filtering based approaches. We test the proposed energy optimization algorithm for two different CMP architectures with 16 and 64 cores. Communication between cores is facilitated via 4x4 and 8x8 regular mesh networks-on-chip. All the measurements and reported results are for the region of interest (ROI) portion of the execution of each application benchmark because that is the region where most of the calculations take place. The default architectural configuration parameters utilized in our custom Sniper based simulations are shown in Table 1.

As discussed, the objective of the optimization algorithm is to minimize energy under user set performance constraints. The user can set such constraints based on known or assumed application criticality levels, which translate into acceptable performance losses. For example, a video streaming application could be categorized as high criticality while an email application can be treated as a rather low criticality level in the sense of expected response or execution time. In this context, a certain criticality level can be assigned a performance loss (PL) threshold or constraint. In this paper, for simplicity, we assume the same criticality for all simulated benchmarks, by setting the PL threshold to $PL = 10\%$. This threshold can easily be changed in our framework; we do not report here results for different thresholds due to lack of space. A PL threshold of $PL = 10\%$ means that the user wants the proposed algorithm to save as much as energy possible but without degrading the performance of the application with more than 10% compared to the case when no energy optimization were done. Thus, the proposed DNN model should ideally be able to suggest the best set of V/F pairs for all cores to ensure energy reduction but within the acceptable performance degradation. To achieve that, Phase 1 discussed earlier in the paper must first be done for the given PL threshold such that the training data is collected for that PL. Then, in Phase 2, the training data is used to train the DNN model, which will then be plugged in into the DNN controller used to proactively provide V/F settings to all cores during all control periods within the execution time of the application. In cases where applications are highly critical and could not tolerate any performance degradation, the energy optimization scheme could be turned off and all cores run at the highest clock frequency available. Otherwise, as an alternative scheme, we could construct multiple DNN models, each trained for a specific PL value, and then, build in an enhanced scheme that could switch between models.

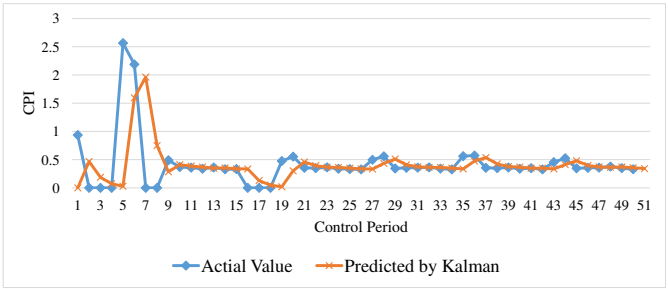
All algorithms of the proposed framework are implemented in C++. Some tasks however such as the use of the Google's Tensorflow are done in Python. Specific details on how these development tasks were done cannot be described in detail here. That is because of lack of space and because this would be description of coding/programming, which cannot be presented as technical contributions. Moreover, we note that we will make the entire implementation of this project publicly available to facilitate replication of results as well as investigations for different PL values. Details of the code architecture can be seen directly in the implementation.

6.2 Collection of Training Data

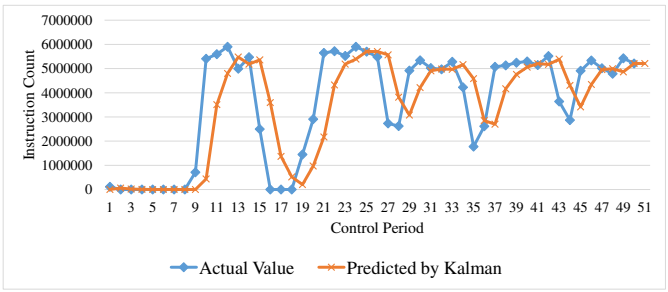
We have implemented all six steps discussed earlier in the paper in the custom Sniper simulator, which is paused during each control period for the purpose of collecting training data points. During Phase 1, we use the Kalman filtering based prediction, as illustrated in Fig. 5. We used half of the benchmarks (i.e., *fmm*, *lu.cont*, *ocean.cnt*, *radiosity*, *raytrace*, *facesim*, *freqmine*, and *swaptions*), selected arbitrarily, for collection of training data. But, only 70% of that training data is actually used for training; the remaining 30% is used for model testing and validation. The Kalman filtering technique is used to predict the instruction count and the average CPI for each core of the CMP architecture during each control period. Because we have the implementation from [17], we use the same values for the filter parameters: $A = 5$, $H = 1$, $Q = 1$, $R = 0.5$ and $B = 0$. These Kalman parameters were found to provide good results.

For example, Fig. 8 shows the values of the CPI and the instruction count predicted by the Kalman filter as well as their actual values for a sample core while running the *fmm* benchmark with 16 threads on a 16 core CMP architecture and 10% PL constraint. These are values predicted during each control period in step 1 above. Note that, Kalman filtering provides excellent prediction accuracy, which is the reason we use it for collection of training data as well as for comparison. The Kalman filtering prediction does not perform very well though during abrupt changes of the predicted variable. The corresponding frequency values calculated in step 2 from Fig. 7 are plotted in Fig. 9, which also shows the adjusted or corrected frequency values. It is the corrected values that are then used as labels together with performance counters of the cores, caches, memory, and NoC to create training data points. Recall that a training data point is constructed with input features for a moving window of m past control periods. In our simulations, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2870438, IEEE Transactions on Multi-Scale Computing Systems



(a)



(b)

Fig. 8. CPI and instruction count values collected during step 1.

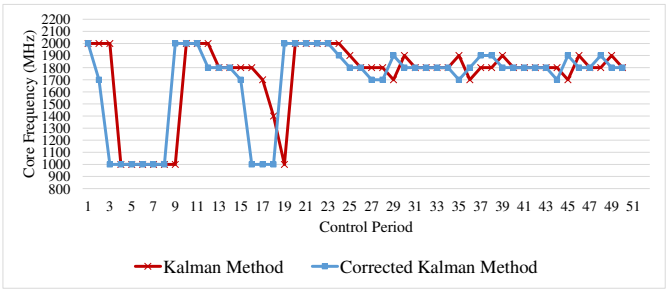


Fig. 9. Frequency values calculated in step 2 from Fig. 7.

use a value of $m = 5$ to always capture the workload behavior of the 5 past control periods. However, this parameter can be changed by the user. In this example, during each control period we collected 62 performance counters plus the frequency value for which the counters values were generated. These performance counters include statistics from CPU performance counters, different levels of caches, stalls, uncore memory accesses, TLBs, branch predictors, ALU activities including int, fp, mul/div operations and others that are available inside the Sniper simulator. We do not list them all here, but, the documentation that we will release with the complete source code of our implementation describes this to aid in the use of the simulation framework. In summary, each training data point (saved in the DNN data) includes a vector of 5x63 values as the input feature plus one value as the output label; that is a total of 316 values that required roughly 2.5KB per core. Thus, we need 16x2.5KB=40KB and 64x2.5KB=160KB of memory for the 16 core and 64 core CMP architecture, respectively.

Note that, in our simulations, the length of each control period is 1 ms of actual region of interest (ROI) application benchmark simulated execution time. Note that typical ROI duration for Parsec and Splash2x benchmarks is in the order of tens of ms, which are simulated by full system simulators for much longer durations (sometimes up to a few hours) as wallclock simulator runtime. We are forced to work with such a small control period because the total length of the region of interest (ROI) of the benchmarks that we use in simulations is relatively short. However, this parameter would be changed to larger values in real-life deployment where workload benchmarks are executed continuously or for very long times and not for just tens or hundreds of ms that is the typical length of the ROI in full system simulators like Sniper and Gem5. This phase took 2 days to complete for both 16 core and 64 core architectures, due to fact that the Sniper framework was instrumented to pause to be able to collect DNN data. Also, full system simulators, while very accurate, are inherently relatively slow compared to the case when benchmarks would be executed on real systems rather than inside simulators.

6.3 Training of the DNN Model

At this point, we needed to decide about the exact topology of the DNN model. Previous literature does not provide helpful recipes in terms of how one should size-up a DNN model. Most often, previous literature just reported the exact DNN topology without further elaborations. In our case, we conducted a design space exploration type of search to identify the topology for the DNN model that provided the best results for a few selected benchmarks. We started with one hidden layer and increased the number of layers until no further improvement was noticed. For a given number of layers, we varied the number of units per layer as 300, 400, or 500. At the end of this search, we have found empirically that an eight layer DNN model was a good topology that provided good results, yet it is manageable in terms of training times and required storage. The final selected DNN models are shown in Fig. 10. While not necessary, we found that conducting a separate search to identify the topology for the DNN model for any new CMP architecture, leads to slightly better results.

Once the DNN models were selected, training was done using the training data set collected as described in the previous section. Tensorflow generated the DNN model (i.e., information about the network topology, number of hidden layers, number of units on each layer, and all weights), which used about 2 MB of memory. During training, we used a learning rate of 0.001 and a number of training steps of 2000. The trained DNN model provided about 80% accuracy on the testing and validation data set, which contained 30% out of the collected training data set. This phase required about 15 minutes for the 16 core CMP architecture and 1.5 h for the 64 core CMP architecture.

6.4 Runtime Prediction using the DNN Model

Once the DNN model is trained, we are ready to evaluate its performance. This corresponds to Phase 3 in Fig. 6. Essentially, the DNN model is used directly to identify V/F pairs for all cores during each control period during the execution of a given benchmark. Evaluation of the DNN model is pretty fast on the machines we used in our simulations. V/F pairs for all cores are found in about 10 ms of wallclock runtime of the simulator. This should not be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2870438, IEEE Transactions on Multi-Scale Computing Systems

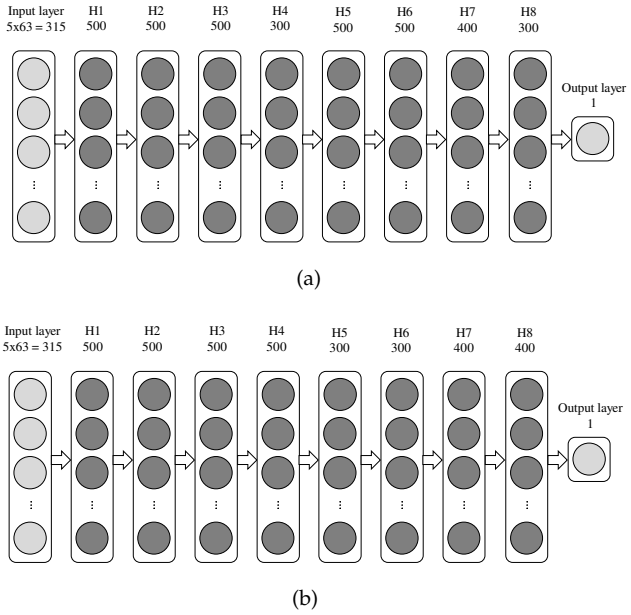


Fig. 10. Topologies of the DNN models for a) 16 core CMP architecture and b) 64 core CMP architecture.

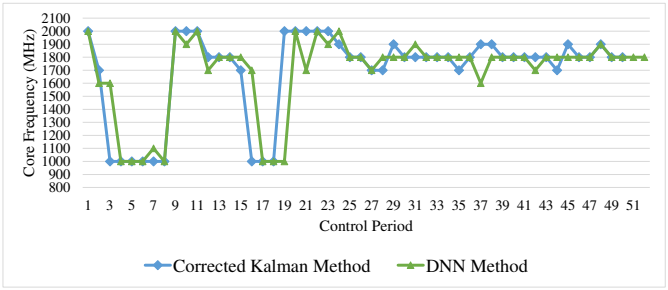


Fig. 11. Comparison of the predicted frequencies by the DNN model to those calculated by the Kalman filtering technique.

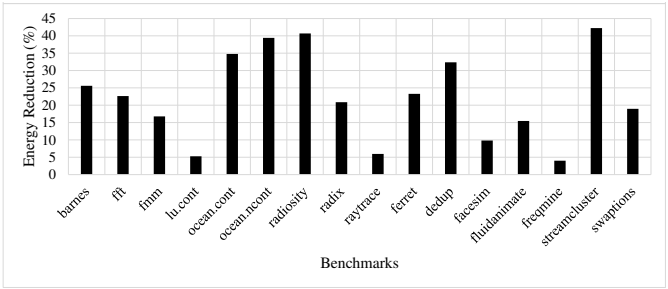


Fig. 12. Comparison of the proposed DNN model based energy optimization algorithm vs. no optimization at all for 16 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.

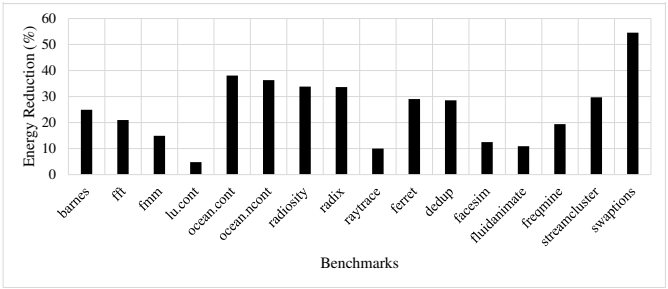
an issue in practice where applications run for much longer times (or even continuously) in datacenter servers. In such cases, the control period would be selected much longer too, compared to the region of interest (ROI) duration that system simulators like the one used in this paper focus on.

As an initial test of the DNN model, we compare its V/F predictions to those computed using the Kalman filtering based heuristic for the *fmm* benchmark testcase. Fig. 11 shows some of the results of this comparison, corresponding to only one of the cores of the 16 core CMP architecture. We observe that, the DNN model is pretty good at predicting the right frequencies (i.e., V/F pairs).

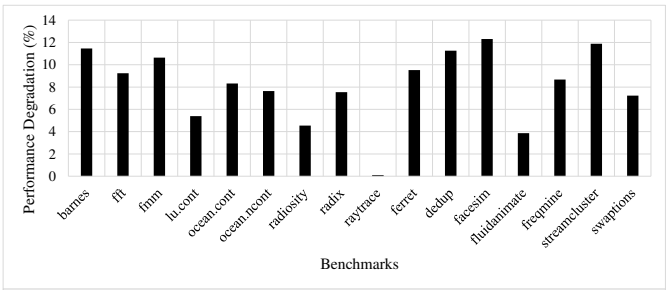
In a first set of simulations, we compare the proposed DNN model based dynamic energy management algorithm to the case when no DEM algorithm is used at all. The results of this comparison are reported in Fig. 12 and Fig. 13 for the two CMP architectures. These figures report percentages in energy reduction, in total performance (i.e., benchmark execution time) degradation, and in energy-delay-product improvement. In the second set of simulations, we compare the proposed DNN model based approach to the reinforcement learning approach described in [14] and the Kalman filtering approach from [17]. As in [17], we selected to work with a moving window of control periods of length $m = 5$ for the DNN model based approach. The results of this comparison are reported in Fig. 14 and Fig. 15 for the two CMP architectures.

6.5 Discussion

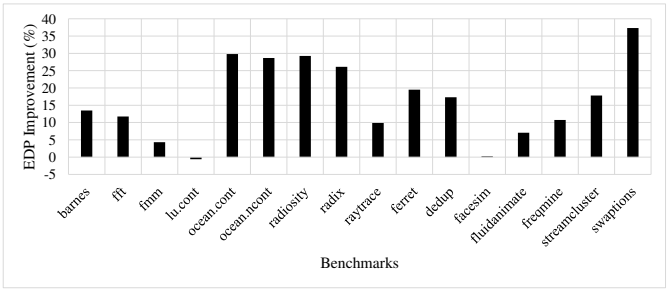
Looking at Fig. 12 and Fig. 13, we note that in the majority of cases, the proposed DNN model based approach achieves significant energy reduction while keeping the total performance loss under the user specified performance constraint fairly well. Nevertheless, the performance degradation is slightly larger than expected in some benchmarks. We attribute this to the fact that the DNN model is not a perfect oracle. Most importantly, we see that the EDP is improved in most of the cases. However, in some instances that is not the case. This is possible for difficult benchmarks that have their ROI fully packed with workload at all times. In such cases, there is practically very little room that could be exploited effectively towards energy reduction with minimal performance degradation via frequency throttling. Looking at Fig. 14 and Fig. 15, we note that the proposed DNN model based approach provides consistently better energy-delay-product



(a)



(b)



(c)

Fig. 13. Comparison of the proposed DNN model based energy optimization algorithm vs. no optimization at all for 64 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.

(EDP) values than both RL and Kalman filtering approaches. On average, the EDP improvement is 6.3% and 6% for the 16 core CMP architecture and 7.4% and 5.5% for the 64 core CMP architecture, respectively. This is summarized in Table 2.

TABLE 2
Average improvement in terms of EDP values.

Comparison	16 core CMP architecture	64 core CMP architecture
DNN vs. RL	6.3%	7.4%
DNN vs. Kalman	6%	5.5%

While the improvement is within the range of 5.5%-7.4% on average, we consider that this is valuable. Aside from the fact that this study does improve results over the existing approaches, and despite that DNN models require training data collection and training, the work in this paper sheds light on what a relatively straightforward DNN based approach for energy optimization would be able to achieve. This can be useful information for other researchers who may be interested in employing DNN models at the processor level - our results would provide an informed starting or reference point. We consider our work as a step towards what other researchers see as a necessity to address the complexity in designing CMP systems and machine learning techniques [32].

7 Conclusion

We proposed for the first time the use of DNN models for energy optimization in CMP systems. We introduced a novel algorithm for dynamic energy management under performance constraints. It uses a DNN model to directly specify optimal voltage-frequency pairs for each core in the CMP architecture. The proposed method is implemented in three phases including training data collection, model training, and model use in the dynamic energy management algorithm. Simulation results using a variety of benchmarks executed on 16 core and 64 core network-on-chip based CMP architectures demonstrated that the DNN model based energy optimization can achieve up to 55% energy reduction for 10% performance degradation constraints, compared to the case when no optimization is done. The proposed DNN approach was also compared against existing approaches based on reinforcement learning (RL) and Kalman filtering. We found that it provides average improvements in energy-delay-product of 6.3% and 6% for the 16 core CMP architecture and of 7.4% and 5.5% for the 64 core CMP architecture, respectively.

In future work, it would be interesting to extend the DNN model to situations when both DVFS and task mapping are used for energy optimization. Currently, it is unclear how the accuracy of the DNN model would be affected if it were used in a system that combines DVFS and task migration.

References

[1] J. Whitney and P. Delforge, "Data center efficiency assessment - scaling up energy efficiency across the data center industry: evaluating key drivers and barriers," *Natural Resources Defense Council (NRDC) Report*, 2014. [Online]. Available: https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf

[2] Annual Energy Outlook, U.S. Energy Information Administration (EIA), 2016. [Online]. Available: http://www.eia.gov/forecasts/aeo/data.cfm#enconsec

[3] United States Environmental Protection Agency, "Report to Congress on server and data center energy efficiency," *Report*, 2007. [Online]. Available: https://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf

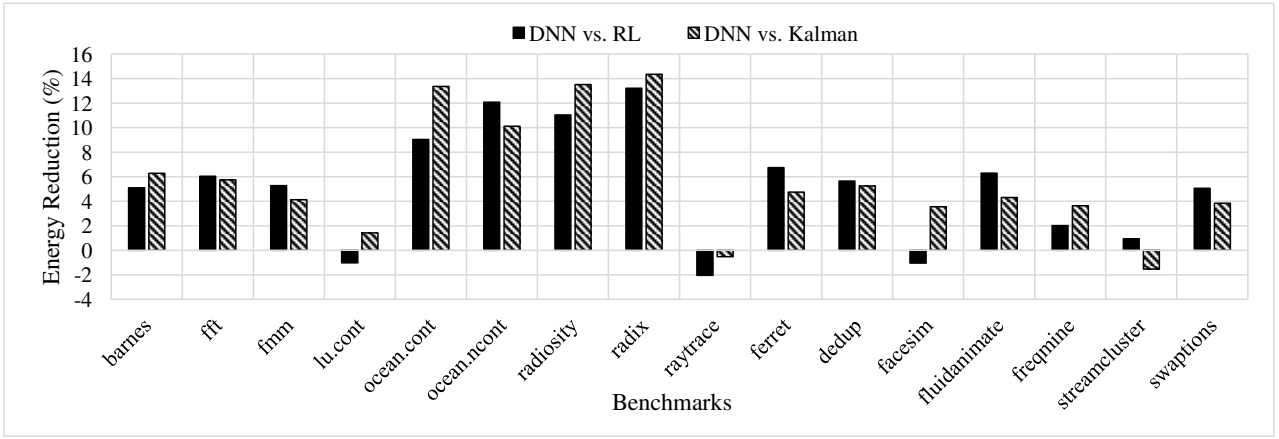
[4] A. Das, A. Kumar, B. Veeravalli, R.A. Shafik, G.V. Merrett, and B.M. Al-Hashimi, "Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems," *ACM/IEEE Design, Automation & Test in Europe Conference (DATE)*, 2015.

[5] R. Cochran, C. Hankendi, A.K. Coskun, and S. Reda, "Pack & cap: adaptive DVFS and thread packing under power caps," *ACM/IEEE Int. Symposium on Microarchitecture (MICRO)*, 2011.

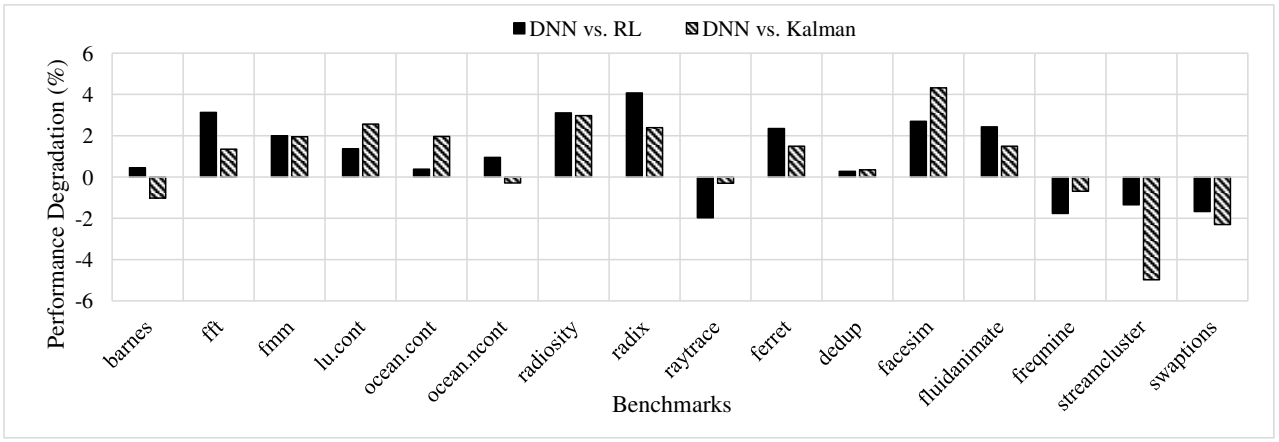
[6] G. Dhiman and T.S. Rosing, "Dynamic voltage frequency scaling for multitasking systems using online learning," *ACM/IEEE Int. Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

[7] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," *ACM/IEEE Int. Symposium on Quality Electronic Design (ISQED)*, 2012.

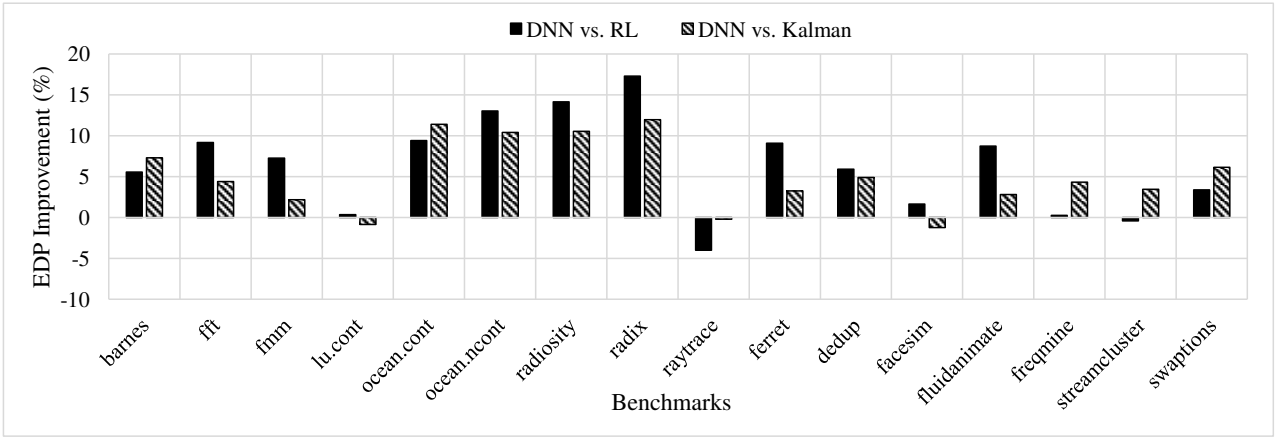
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TMSCS.2018.2870438, IEEE Transactions on Multi-Scale Computing Systems



(a)



(b)



(c)

Fig. 14. Comparison of the proposed DNN model based energy optimization algorithm against the RL and the Kalman filtering based approaches for 16 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.

[8] R. Ye and Q. Xu, "Learning-based power management for multicore processors via idle period manipulation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 7, pp. 1043-1056, July 2014.

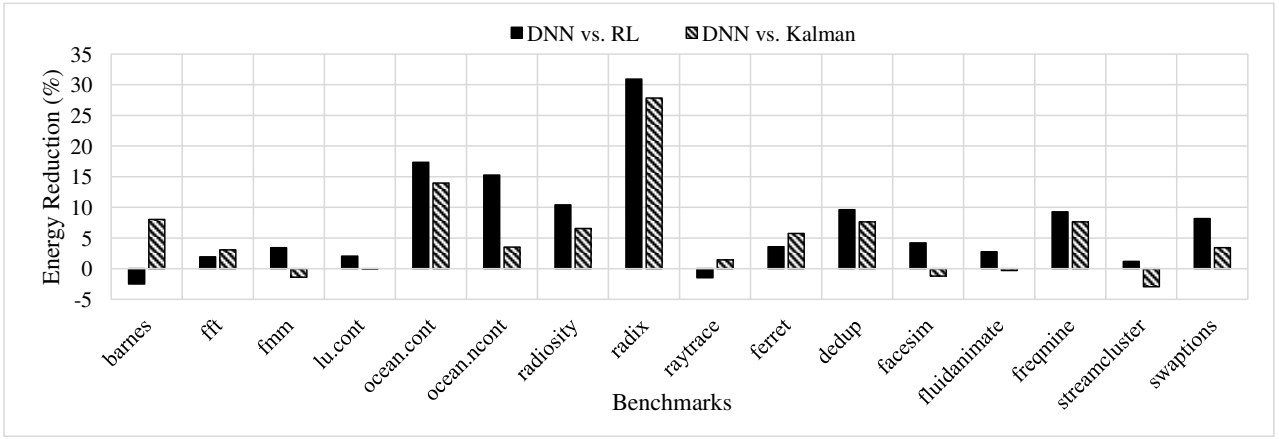
[9] B.K. Donohoo, C. Ohlsen, S. Pasricha, Y. Xiang, and C.W. Anderson, "Context-aware energy enhancements for smart mobile devices," *IEEE Trans. on Mobile Computing*, vol. 13, no. 8, pp. 1720-1732, July 2014.

[10] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 9, pp. 1395-1408, Sep. 2010.

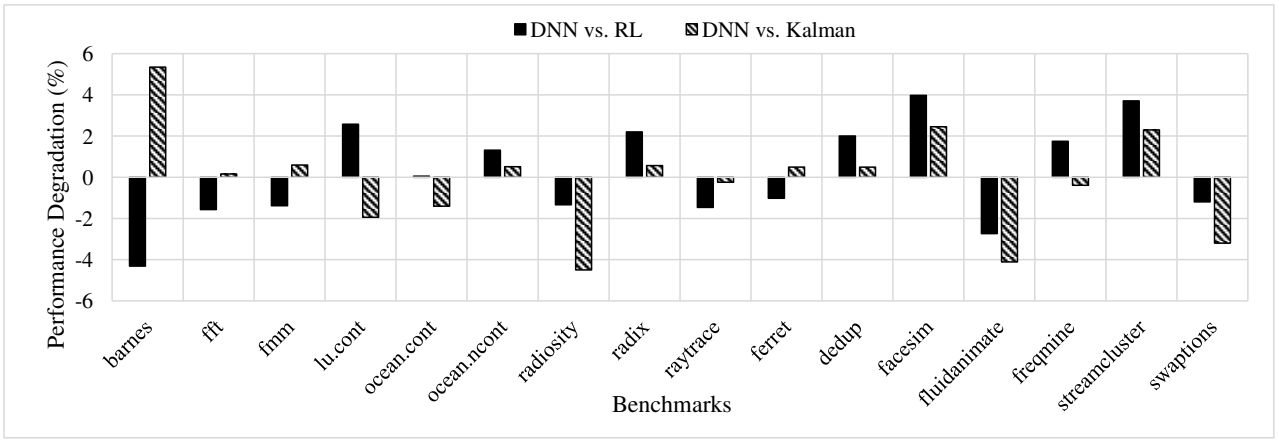
[11] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," *ACM/IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[12] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, article 24, March 2013.

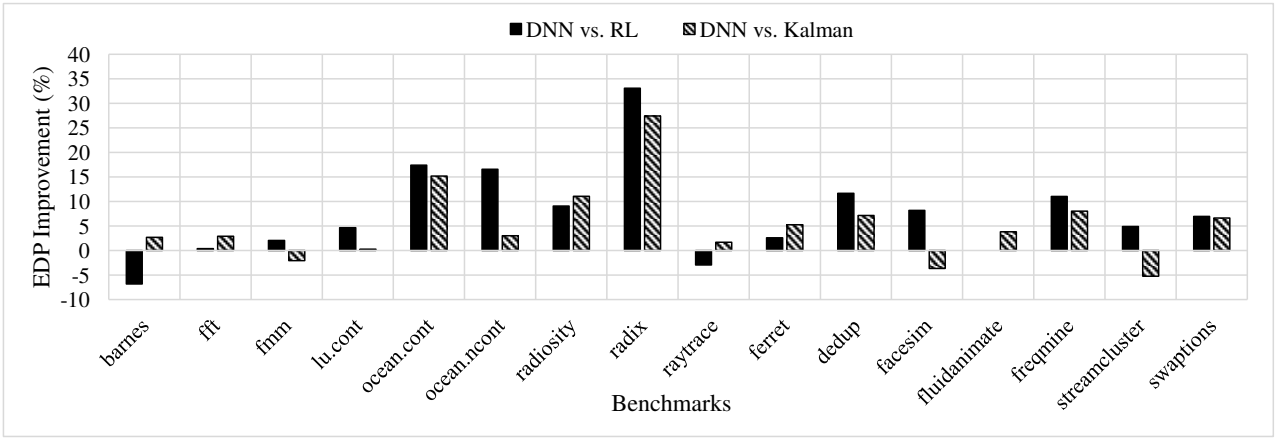
[13] A. Das, R. Shafik, G. Merrett, B. Al-Hashimi, A. Kumar and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," *ACM/IEEE Design Automation Conference*



(a)



(b)



(c)

Fig. 15. Comparison of the proposed DNN model based energy optimization algorithm against the RL and the Kalman filtering based approaches for 64 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.

(DAC), 2014.

[14] Z. Wang, Z. Tian, J. Xu, R. Maeda and H. Li, "Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system," *ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[15] D. Biswas, V. Balagopal, R. Shafik, B. Al-Hashimi and G. Merrett, "Machine learning for run-time energy optimisation in many-core systems," *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2017.

[16] R.G. Kim, W. Choi, Z. Chen, J.R. Doppa, P.P. Pande, D. Marculescu and R. Marculescu. "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Trans. on VLSI Systems*, vol. 24, no. 9, pp. 2488-2501, Sep. 2017.

[17] M.G. Moghaddam and C. Ababei, "Dynamic energy management for chip multiprocessors under performance constraints," *Microprocessors and Microsystems*, vol. 54, pp. 1-13, Oct. 2017.

[18] M.G. Moghaddam, W. Guan and C. Ababei, "Investigation of LSTM based prediction for dynamic energy management in chip multiprocessors," *IEEE Int. Green and Sustainable Computing Conference*, 2017.

[19] J.Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou, "Up by their bootstraps: online learning in artificial neural networks for CMP uncore power management," *HPCA*, 2014.

[20] J. Gao, "Machine learning applications for data center

optimization," *Google White Paper*, 2014. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42542.pdf.
[21] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, Sep. 2016.
[22] Kevin P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.
[23] Y. LeCun, "Learning invariant feature hierarchies," *ECCV*, 2012.
[24] G. Welch and G. Bishop, An Introduction to the Kalman Filter, Chapel Hill, NC: Univ. North Carolina, Chapel Hill, 1995.
[25] S. Bang, K. Bang, S. Yoon, and E. Chung, "Run-time adaptive workload estimation for dynamic voltage scaling," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, no. 9, pp. 1334-1347, Aug. 2009.
[26] S. Sarma and N. Dutt, "Minimal sparse observability of complex networks: application to MPSoC sensor placement and run-time thermal estimation & tracking," *(DATE)*, 2014.
[27] The vanishing gradient problem, 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap5.html
[28] T.E. Carlson, W.Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2011.
[29] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi, "McPAT: an integrated power, area, timing modeling framework for multicore and manycore architectures," *Int. Symposium on Microarchitecture (MICRO)*, 2009.
[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S., Corrado, A. Davis, J. Dean, M. Devin and S. Ghemawat, "Tensorflow: large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, March 2016.
[31] PARSEC and Splash2 benchmarks, 2017. [Online]. Available: http://parsec.cs.princeton.edu
[32] R.G. Kim, J.R. Doppa, P.P. Pande, D. Marculescu and R. Marculescu. "Machine learning and manycore systems design: a serendipitous symbiosis," *Submitted to Learning*, Dec. 2017. [Online]. Available: https://scirate.com/arxiv/1712.00076

Milad Ghorbani Moghaddam (S'16) received the B.S. degree from Ferdowsi University of Mashhad, Iran in 2008 and the M.Sc. degree from Isfahan University of Technology, Iran in 2011, both in computer engineering. Currently, he is a Ph.D. student in the Department of Electrical and Computer Engineering, Marquette University, Milwaukee, WI, USA. His main research interests include energy consumption, lifetime reliability of chip multiprocessors and full system simulators.

Wenkai Guan received the B.S. degree from Wuhan University of Technology with Excellent Undergraduate Student Honor in June 2015. He then spent half a year working as a research assistant at the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in Huazhong University of Science and Technology. Currently, Wenkai is pursuing the Ph.D. degree at Marquette University. His research interests are in multicore embedded systems.

Cristinel Ababei (SM'14) received the Ph.D. degree in electrical and computer engineering from the Univ. of Minnesota, Minneapolis, in 2004. He is an assistant professor in the Dept. of ECE, Marquette Univ. Prior to that, from 2012 to 2013, he was an assistant professor in the Dept. of EE, SUNY at Buffalo. Between 2008 to 2012, he was an assistant professor in the Dept. of ECE, North Dakota State University. From 2004 to 2008, he worked for Magma Design Automation, Silicon Valley. His current research interests include electronic design automation of systems-on-chip with emphasis on reliability and energy consumption, datacenters, parallel computing, and FPGAs.