Identifying the Optimal Energy-Efficient Operating Points of Parallel Workloads

Ryan Cochran
School of Engineering
Brown University
Providence, RI 02912
ryan_cochran@brown.edu

Can Hankendi
ECE Department
Boston University
Boston, MA 02215
hankendi@bu.edu

Ayse Coskun
ECE Department
Boston University
Boston, MA 02215
acoskun@bu.edu

Sherief Reda
School of Engineering
Brown University
Providence, RI 02912
sherief_reda@brown.edu

ABSTRACT

As the number of cores per processor grows, there is a strong incentive to develop parallel workloads to take advantage of the hardware parallelism. In comparison to single-threaded applications, parallel workloads are more complex to characterize due to thread interactions and resource stalls. This paper presents an accurate and scalable method for determining the optimal system operating points (i.e., number of threads and DVFS settings) at runtime for parallel workloads under a set of objective functions and constraints that optimize for energy efficiency in multi-core processors. Using an extensive training data set gathered for a wide range of parallel workloads on a commercial multi-core system, we construct multinomial logistic regression (MLR) models that estimate the optimal system settings as a function of workload characteristics. We use L_1-regularization to automatically determine the relevant workload metrics for energy optimization. At runtime, our technique determines the optimal number of threads and the DVFS setting with negligible overhead. Our experiments demonstrate that our method outperforms prior techniques with up to 51% improved decision accuracy. This translates to up to 10.6% average improvement in energy-performance operation, with a maximum improvement of 30.9%. Our technique also demonstrates superior scalability as the number of potential system operating points increases.

1. INTRODUCTION

With the energy consumption of large-scale computing systems increasing by over 15% per year [11], power and energy management has assumed central importance in nearly every current and future computing domain. Most modern systems offer a slew of power management features, with dynamic voltage frequency scaling (DVFS) being one of the most well known. DVFS is an extremely powerful technique, and is used for modulating the operating frequency in order to take advantage of performance slack and reduce power consumption.

At the same time, with the increased hardware parallelism in multi-core systems, workloads are becoming increasingly parallel in high performance computing (HPC) clusters and datacenters. The typical applications in these domains span a diverse set of parallel workloads (e.g., modeling, scientific computing, financial applications, and media processing), all of which require complex thread synchronization models. This complexity makes the task of workload characterization for the purposes of energy, power, and performance optimization for parallel workloads extremely challenging. Energy, power, and performance characteristics also change dramatically as a function of the workload. Thus, modern HPC and datacenter systems require sophisticated management strategies that have extensive knowledge of the workload behavior. An effective runtime management technique must be capable of selecting the optimal system setting as a function of the workload.

This paper proposes a scalable approach for calculating the optimal DVFS setting and number of parallel threads for a parallel workload at runtime as a function of measurable workload characteristics. Our approach can be adapted to work with any conceivable power, energy, and performance objective formulation. For example, in situations that require maximum performance, the system operating point is selected such that execution delay is minimized. When the energy consumption is of primary concern, such as for battery operated devices, our proposed models can be trained to select an operating point that minimizes energy. In all cases, special care must be taken to avoid violating the system's peak power budget or degrading the system performance beyond desirable limits.

The objective of this paper is to provide a systematic approach for mapping complex parallel workload behavior to an optimal operating point decision. Our contributions are as follows:

- We investigate power, energy, and performance tradeoffs among the operating points available on a multicore system under various power/performance objectives and constraints. These tradeoffs are then used for guiding the power management strategies for multicore systems running parallel workloads.
- We present an online energy management technique that utilizes a *multinomial logistic regression* (MLR) classifier to find the optimal number of parallel threads for a multithreaded workload in conjunction with the optimal DVFS setting. Performance, power, and temperature data are collected on an Intel i7 based system and are utilized to train the L_1-regularized MLR model. In addition to the data collected on the target system, the model also takes an objective-constraint formulation as input. Once trained, the model can estimate the likelihood of optimality for each operating point and select the likeliest one.
- We provide a scalable power management methodology that is highly accurate under a set of objective functions relevant to HPC clusters and datacenters. We demonstrate the accuracy and scalability of our technique using the parallel workloads in the PARSEC benchmark suite [2]. Our technique outperforms previous DVFS techniques which are mainly guided by the frequency of the memory operations (e.g., [9, 5]) with 51% higher decision accuracy, improved scalability with respect to the number of potential operating, and a reduction of 10.6% average EDP across all workloads.

The rest of the paper is organized as follows. Section 2 provides an overview of the related work. Section 3 discusses the objective functions we use in optimizing energy consumption. Section 4 explains our methodology in detail. Section 5 demonstrates experimental results and Section 6 concludes the paper.

2. BACKGROUND

Dynamic power management on multi-core systems is a well established research area, with lower-power idle modes (e.g., [14]) and DVFS being two of the most commonly employed control knobs. In this work, we contribute a novel control knob in active thread-count modulation for parallel workloads in addition to conventional DVFS. Many software and hardware-based DVFS control approaches have been proposed in recent years. Multiple-clock domain design and voltage frequency island partitioning are examples of hardware-based DVFS methods [13, 7]. Kim *et al.* explore designing on-chip regulators and perform per-core DVFS in nanosecond granularity [10]. Teodorescu *et al.* propose process variation-aware algorithms for power management through scheduling and DVFS [19]. Software-based approaches require application level or compiler level support for power control [17, 1]. There are also a number of methods for optimizing energy consumption for systems where task graphs and deadlines are known *a priori* (e.g., [20, 21]).

The majority of recent work on power/energy-aware DVFS use information gathered from Performance Monitoring Units (PMUs), which expose hardware performance counters capable of counting various architectural events (μ-ops retired, l3-cache misses, etc.) [12, 3, 18]. Some methods attempt to optimize performance within power constraints. Etinski *et al.* propose a job scheduling policy that optimizes performance for a given power budget [6]. There are also global power management policies to maximize performance for a given power budget on multicore systems [8].

Many optimize for *Energy Delay Product* (EDP) or *Energy* $Delay^2$ *Product* (ED^2P), although there are also techniques that use *Energy per Instruction* (epi or epi^2) [15]. Isci *et al.* uses phase categories calculated using a metric for *Memory Operations per μ-op* in [9]. Each phase category is linked to a DVFS policy that attempts to minimize EDP. However, this approach requires reconfiguration of the phase boundaries to handle alternative objective formulations. Dhiman *et al.* propose an online learning model that follows a similar metric classification method for single-core processors [5]. In order to characterize workloads, this work breaks down the *Cycles Per Instruction* (CPI) into various components, such as baseline CPI, miss events CPI, and stall CPI, in order to construct the CPI stack of the executing task. This approach guarantees convergence to the optimal DVFS setting via online learning.

Our work makes several key contributions relative to the state-of-the-art: **(1)** Because our approach accesses built-in hardware performance counters, it does not require modifications to the OS scheduler or compiler required by many software-based techniques (e.g., [6]). **(2)** Many prior methods (e.g., [8]) rely on static trace analysis, while our approach is dynamic and based on training data. **(3)** Our technique shows superior accuracy and scalability relative to the most similar prior works [5, 9] (see Section 5), and shows that previous metrics based on memory and stall events are insufficient to characterize emerging parallel workloads. Our approach utilizes a large set of performance counter metrics and per-core thermal sensor measurements to characterize a diverse set of parallel workloads. **(4)** Our technique is general enough to accommodate a wide range of optimization formulations that are relevant to HPC clusters and datacenters. It provides a means to change both the optimization objective, as well as the constraints types.

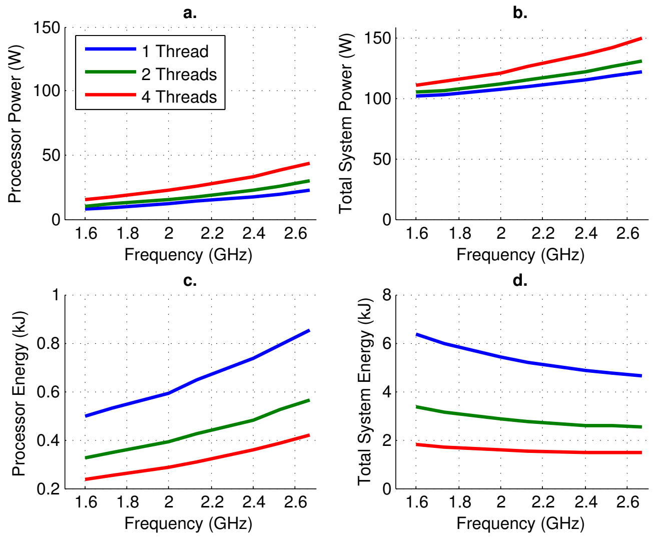

Figure 1: Power and energy characteristics for segment of `blackscholes` benchmark.

3. FORMULATIONS FOR ENERGY EFFICIENT COMPUTING

The fundamental objective of any energy-aware control scheme is to intelligently manage the trade-off between the energy consumption and execution delay within some system constraints. At first glance, it might seem that the tradeoff is straightforward: faster execution reduces the delay which may lead to lower energy. Faster execution, however, may cause higher power consumption which maybe offset the reduction in delay. For example, running a parallel workload with a higher number of threads or increasing the processor's frequency reduces the execution delay, but at the cost of higher power consumption because of the associated increase in operating voltage and utilization of processor units.

When minimizing energy or any objective derived from energy (e.g., EDP and ED^2P), the relationship between the control settings and the objective is not straightforward. The effect of thread-count and DVFS setting on execution delay is highly workload dependent: some workloads benefit from higher frequencies and higher parallelism, while others exhibit performance saturation. Furthermore, the optimal energy settings are highly dependent on what components of the power consumption are considered. When optimizing energy, one should only consider the components of the total power that vary as a function of the control settings, or can be eliminated during idle periods. For instance, if the components on a motherboard can be turned off or put in a low-power deep sleep state once a workload has completed, then they should be included in the energy optimization because the system can benefit from finishing faster. However, those components of the total power that cannot be switched off and invariant with respect to the control settings should be excluded as their contribution to the energy dissipation cannot be eliminated.

Inclusion/exclusion of various power components has a dramatic effect on optimization results. Figures 1a. and 1c. show the average processor power and processor energy dissipation for the first 100 billion microoperations (μ-ops) of the `blackscholes` workload, which is part of the PARSEC suite, as a function of the DVFS setting, while Figures 1b. and 1d. show the average total system power and energy. Including the power consumption of the motherboard components (memory, bridge components, fan, etc.) dramatically increase the fixed component of the power, and therefore the workload benefits most from faster execution, and energy decreases with higher frequency. If these power components are excluded, the energy decreases with thread-count but increases with frequency. The optimal energy point is determined by the balance between the fixed and variable power components, as well as the workload-dependent effects on the execution delay.

In this work, we focus on optimizing the processor's energy, excluding the contributions of the motherboard components on the assumption that the whole server system cannot switch to a sleep or nap mode once execution is finished [14]. In our experiments, we observe that the system power is higher than the processor power by an offset that is linearly correlated with the operating frequency. The range of this offset is changing between 108W and 115W depending on the workload characteristics. We expect future systems to include energy-proportional power management features (such as low-power states for memory units or hard disk unit) to help minimize the overall system energy cost.

There are a variety of energy-aware optimization formulations that can be used to manage the tradeoff between performance and energy. Possible formulations include (a) minimize delay under peak power constraints, (b) minimize energy under peak power and delay constraints, (c) minimize EDP, (d) minimize ED^{2}P. Because no single formation dominates the others, we do not attempt to argue the validity of any single objective. We instead provide a means of translating knowledge about the workload into an accurate *a priori* prediction of the optimal system operating point for any formulation. Table 1 provides a summary of the formulations explored in this work. We next discuss the details of each formulation.

A. minDPC: The minDPC(P) objective formulation minimizes the execution delay within a power constraint P. For example, the formulation minDPC(50W) selects the system operating point that minimizes the delay of an execution interval such that the maximum power within that interval never exceeds 50W. Minimizing delay is important from a user perspective and constraining peak power is important for data center operators as it is a primary factor in provisioning costs as well as the utility costs [14]. Even though it reduces delay and restricts power, it does not necessarily minimize energy consumption because achieving the minimum energy may incur higher peak power at the benefit of reduced delay.

B. minEPDC: The minEPDC(P, D) objective minimizes energy within a power constraint P and a delay degradation lower-bound D, where D is the maximum allowable percentage increase in runtime delay relative to the minimum execution delay $d_{\min}$. For example minEPDC(50W, 5%) seeks to minimize energy while operating the system within a 50W power budget and without degrading the runtime execution delay by more than 5%. This formulation has an upper bound on the maximum power for the same reason as for minDPC(P). However, by minimizing energy directly instead of delay, this formulation does not incur the high power-cost of pushing the delay to its minimum level $d_{\min}$. Instead, it forces the delay to be below some tolerance with respect to $d_{\min}$, and the energy is minimized within this constraint. There is often little value to pushing the delay to its minimum feasible level. Web servers, for example, must achieve an expected latency below some tolerance, beyond which there is little visible improvement to a user. With this formulation, it is important to note the possibility of selecting power and delay constraints that cannot be achieved simultaneously.

C and D. minEDP and minED^{2}P: The minEDP and minED^{2}P objectives minimize the energy-delay-product and energy-delay-squared-product respectively. By combining the energy and the delay into a single value, these metrics may balance the need to minimize energy dissipation with the need to minimize the execution delay, with the latter skewed more in favor of reducing delay. While useful in design comparisons, we think that the EDP and ED^{2}P formulations are much less practical in optimization settings than the other objectives because by themselves, they fail to guarantee the performance and maximum power consumption are within an acceptable limits. Depending on the workload, they may also fail to be truly energy-aware because they do not explicitly minimize the energy dissipation. We nevertheless treat these as viable formulations in all of our work.

The aforementioned four formulations provide different power, energy, and performance trade-offs. These trade-offs need to be analyzed carefully to choose the most appropriate formulation that meets the specific needs of HPC clusters and datacenters. To illustrate the outcomes of the four formulations, we find the optimal settings for each of the four formulations for the `bodytrack` and `facesim` workloads from the PARSEC suite on a real quad-core based system. We report in Figure 2 the energy savings and performance degradation percentages with respect to energy and delay results of the highest frequency-voltage setting (2.67 GHz in our setup) with maximum number of threads (4). The results lead to the following insights:

1. *minEDP* does not provide guarantees on execution delay.

Abbreviation	Objective	Power-Delay Form	Power Constraint	Delay Constraint
minDPC(P)	Minimize *Delay*	Minimize $(p_{avg})^0(d)^1$	$p_{max} < P$	--
minEPDC(P, D)	Minimize *Energy*	Minimize $(p_{avg})^1(d)^1$	$p_{max} < P$	$d/d_{min} - 1 > D$
minEDP	Minimize *EDP*	Minimize $(p_{avg})^1(d)^2$	--	--
minED^{2}P	Minimize *ED^{2}P*	Minimize $(p_{avg})^1(d)^3$	--	--

Table 1: Energy-aware objective formulations that manage the tradeoffs between power, energy, and execution delay.

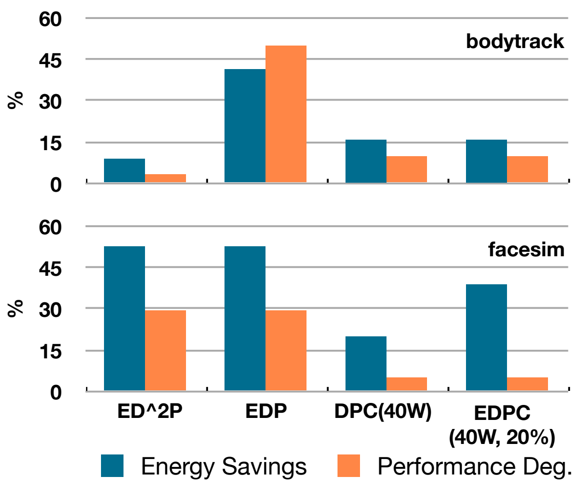

Figure 2: Energy savings and performance degradation for `bodytrack` and `facesim` for each objective function.

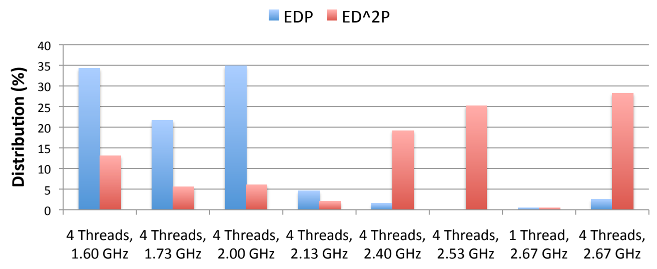

Figure 3: Optimal EDP and ED^2P operating point distribution for data data collected from entire PARSEC benchmark suite.

For instance, minimizing EDP for `bodytrack` causes over 45% performance degradation which might not be acceptable for HPC clusters or data centers which provide services that must meet an expected latency. This example clearly demonstrates the downside of having no power or performance constraints on the objective function. In contrast, the $minEPDC(40W, 20\%)$ formulation achieves over 30% energy savings with the specified acceptable performance degradation for `facesim`.

2. Energy savings and performance degradation for $minED^2P$ objective function show significant variation across two benchmarks. Depending on the workload characteristics, energy savings differ by more than 40% and performance degradation differs over 25%. This high variation might cause unexpected outcomes in terms of operating costs and service reliability. Power management policies should be able to provide reasonable energy savings and performance for any type of workload, and must be robust to workload variations.

In addition to energy savings and performance degradation, peak power is an important constraint in many large-scale clusters. The power infrastructure of any HPC or data center may impose limitations on instantaneous power, which requires a formulation with a power bound, such as $minDPC$ or $minEDPC$. These two formulations have two distinct minimization targets, delay and energy, respectively. As minDPC minimizes delay without any energy limitations, this formulation targets HPC clusters or workloads where the primary concern is performance. For data centers and HPC clusters that can tolerate a limited level of performance degradation, we should optimize for $minEDPC$ by choosing the power budget and degree of performance degradation. Cluster operators generally might like to optimize either for delay ($minDPC$) or energy ($minEPDC$) given a set of constraints. Minimizing EDP or ED^{2}P might bring unpredictable energy-performance tradeoffs.

Within a single formulation, the power-performance trade-offs get more complicated as the optimal settings (number of threads and DVFS settings) change within and across workloads. For example, Figure 3 shows a histogram of the optimal DVFS and thread-count settings of the EDP and ED^{2}P metrics for all of the 100 billion UOPs execution segments taken across the entire PARSEC benchmark suite. With no consistent optimal setting, it is clear that it is not possible to derive a simple rule for optimizing a chosen formulation without considering the workload characteristics.

4. METHODOLOGY

Our proposed methodology for selecting the optimal thread-count and DVFS setting is split between an offline and online step. In the offline step, we use an extensive data set gathered for the parallel workloads in the PARSEC benchmark suite to train a *multinomial logistic regression* (MLR) classifier for a specific objective. During runtime, we recall the model associated with the desired objective from a lookup table, and given real-time measurements of various input metrics, predict the likeliest optimal operating point. These models can be trained to predict optimality for any of the formulations detailed in Section 3.

The primary advantage of our approach over previous approaches lies in the degree of automation in the offline step and the simplicity of our online step. While previous approaches require many assumptions on which workload metrics that are most relevant to desired objective, our approach performs this step automatically using an L_1-regularization technique. By accurately and automatically determining the optimal inputs, our approach avoids the need for multiple models associated with specific workloads to cover a wide range of workload behavior. We associate a single model with each objective formulation. In addition, our approach does not require any intermediate step in which the power, energy, and delay characteristics are estimated as a function of the system setting. This step requires extrapolation of the input metric behavior across the range of operating points. By performing these tasks in an offline fashion and exposing the estimated weights in a lookup table, the runtime overhead of our technique is minimized.

4.1 Multinomial Logistic Regression

The MLR classifier is a generalized linear model that estimates the probability of a discrete set of outputs by fitting a multinomial logistic function to continuous input data. In this work, the set of outputs corresponds to the set of feasible system operating points (i.e., combinations of DVFS settings and workload thread-counts). The inputs to this model are a set of workload metrics, which are themselves functions of the system performance-counter values, the per-core temperatures, as well as the current DVFS setting and thread-count. The MLR model is trained using a set of sample inputs and outputs gathered for a series of workloads at every possible control setting. Given the inputs during runtime, the logistic regression can then calculate a priori the probability of each candidate operating point being optimal for a particular objective formulation. The output with the highest probability is then selected.

Let y denote the output of the MLR classifier, $\mathbf{x}$ denote the vector of input values, and $\phi(\mathbf{x}) \in \mathbb{R}^m$ denote a fixed non-linear function of $\mathbf{x}$. The probability of a particular output c under the multinomial logistic model is expressed in Equation 1.

$$Pr(y = c|\mathbf{x}, \mathbf{w}) = \frac{exp(\mathbf{w_c}^T \phi(\mathbf{x}))}{\sum_{c'=1}^{C} exp(\mathbf{w_{c'}}^T \phi(\mathbf{x}))} \quad (1)$$

The variable $\mathbf{w_c} \in \mathbb{R}^m$ contains the weights associated with output c, and $\mathbf{w} \in \mathbb{R}^{C_m}$ is a vector concatenating the weights across all outputs. T denotes the transpose operator. Equation 1 maps a continuous real-valued argument $\mathbf{w_c^T}\phi(\mathbf{x})$ to a probability $y = c$ such that the probabilities across all of the possible outputs $y \in \{1, ..., C\}$ sum to 1. A positive weight $w_{ck} \in \mathbf{w_c}$ implies that a positive value on input $\phi_k(\mathbf{x}) \in \phi(\mathbf{x})$ increases the probability that $y = c$, and likewise a negative weight implies a decrease in probability.

The logistic weights are estimated using training data, which in this case is an extensive set of data gathered for the PARSEC benchmark suite. Each benchmark is executed to completion at each available DVFS setting and thread-count, and the resulting data is divided into windows of fixed size in terms of the number of instructions executed (100 billion μ-ops). By aligning these fixed instruction windows across all possible DVFS and thread-count settings, we are able to determine the optimal setting under a particular objective formulation at each stage of execution in the workload. These *true* values for y and the measured values for $\mathbf{x}$ for each fixed instruction window are then used to train the model.

The weights $\mathbf{w}$ in the logistic model are estimated by minimizing the conditional log-loss of the training data set, expressed in Equation 2, in which $\mathbf{x}_i$ and y_i represent the input values and output value respectively for instruction window i. The weight-estimate $\hat{\mathbf{w}}$ is found using a standard gradient-descent method, which uses the gradient of the objective function in Equation 2 to iteratively search for the optimal value.

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left(-\sum_i \log Pr(y_i|\mathbf{x_i}, \mathbf{w}) \right) \tag{2}$$

During runtime, the probability of each operating point being optimal under a particular objective formulation is calculated according to Equation 1. The classifier then selects the point that achieves the highest probability. In order to prevent over-fitting of the model to the training data, the accuracy of each classifier is calculated on separate test data. The accuracy is defined as the percentage of samples for which the true optimal system operating point is predicted by the classifier. A 5-fold cross validation scheme is used to measure the accuracy of each classifier on the test data. In this scheme, the entire PARSEC data set divided in five folds, where each fold is drawn at random from the data. Each fold then takes a turn as the test data while the remaining data is used to train the classifier. The reported classifier accuracies in Table 3 are the average accuracy across all folds. Cross-validation ensures that the accuracy results will generalize to an independent data set that is separate from the training data.

4.2 L1-Regularization and Input Selection

With a large number of inputs, it is possible to *over-fit* the classification model to the training data. While the accuracy of model in classifying the training data will appear to be very high, any deviation from the training data characteristics in the test data will significantly decrease the accuracy. One standard means of preventing over-fitting is L_1-regularization, in which an L_1 loss term is added to Equation 2 as in 3. The notation $||\cdot||_1$ indicates the L_1-norm. As the effect of the regularization is increased via the constant parameter α, the logistic weight values are forced to smaller magnitudes and the number of non-zero weights decreases, as illustrated in Figure 4. The optimal degree of regularization is experimentally determined by measuring the cross-validated test accuracy for α across several orders of magnitude.

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \left(\alpha \left\|\mathbf{w}\right\|_1 - \sum_i \log Pr(y_i|\mathbf{x_i}, \mathbf{w}) \right) \tag{3}$$

L_1-regularization also has another extremely useful property in that it forces sparse solutions for $\mathbf{w}$ (i.e., many of the coefficients are zero). Thus, L_1-regularization can be used to identify the inputs that are most relevant to classification and eliminate irrelevant inputs. Given the high complexity of workload behavior, it is not immediately obvious what sort of inputs are useful in classification for various power/energy/delay objectives. Using L_1-regularization, however, the number of inputs can be almost arbitrarily high, and the most relevant inputs will be identified through estimating $\mathbf{w}$.

In our implementation, the input vector $\mathbf{x}$ consists of the following values which can be measured in real-time: μ-ops retired, load locks, l3-cache misses, l2-cache misses, resource stalls, branch prediction misses, floating point operations, average per-core junction temperatures, processor frequency, and thread-count. However, the direct input to the MLR model, or $\phi(\mathbf{x})$, has much higher dimensionality, taking the raw inputs and their inverses, as well as every possible ratio of the raw inputs in $\mathbf{x}$. The L_1-regularized MLR classifier subsequently determines which ratios are important for a particular classifier and assigns zero weight to irrelevant input features. Table 2 shows magnitudes of the top 10 ratios for the $minEDP$ classifier in terms of the average absolute values of each weight variable across all outputs.

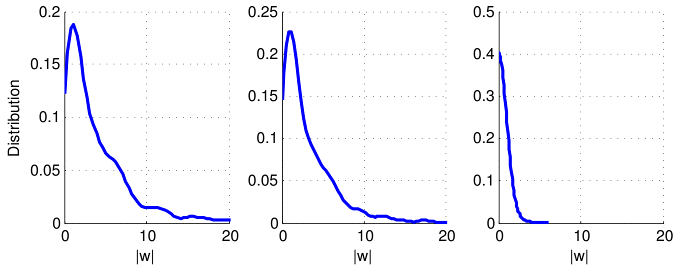

Figure 4: Distribution of L_1-regularized weight magnitudes for $\alpha = 1e-8$, $\alpha = 7.74e-4$, and $\alpha = 3.6$ respectively for minimum EDP classifier.

	Input
1.	(UOPs retired)/(l2-cache misses)
2.	(load locks)/(constant)
3.	(average core temperature)/(constant)
4.	(l2-cache misses)/(thread count)
5.	(constant)/(UOPs retired)
6.	(frequency)/(floating-point ops)
7.	(frequency)/(l2-cache misses)
8.	(branch misses)/(avgerage core temperature)
9.	(l2-cache misses)/(branch misses)
10.	(l2-cache misses)/(resource stalls)

Table 2: The top 10 most relevant workload metrics for the $minEDP$ classifier as determined by the sum of the absolute value of the weights across all output classes.

4.3 Runtime Behavior

The runtime overhead of the proposed technique is minimal. The system simply logs performance counter and temperature data and at regular intervals calculates the probability of each operating point being optimal using Equation 1. The process of training the model weights is performed in an offline fashion, and our results show that this single model is capable of predicting *a priori* the optimal operating point across the wide range of parallel workload behaviors in the PARSEC suite. Our own experiments verify that the application of Equation 1 can be performed with an overhead in the range of 10-50 ms. However, because a control activation period on the order of several seconds is observed to be sufficient to respond to changes in workload behavior, this overhead typically represents a very tiny fraction of the overall execution time.

While our approach is designed to change workload thread-counts mid-execution, the limitations of our workload infrastructure do not permit us to change threads in this way without major code modifications. However, recent literature verifies that there are viable means of changing workload thread-counts after the workload has already been launched via the native functions provided in the openMP library. These functions allow thread-count changes for parallel regions of code with an overhead of 200 μs. [4]. Previous works also show that it is possible to implement same functionality on POSIX library with a similar overhead [16]. The task of DVFS throttling can be performed with around 100 μs overhead [10].

As the runtime overhead is low, the system operator has room to use additional techniques on top of the proposed method to maintain optimality if workloads with highly diverging behavior from the training data set arrive at the target system. In such a scenario, the MLR classifier can be used to predict an initial estimate of the optimal setting. While this initial estimate might not be correct all of the time, it is still highly likely to be *near* the optimal setting in terms of its DVFS value and the number of threads. The system can then periodically increment or decrement the DVFS setting and/or number of threads and observe the affect on the objective (energy, delay, EDP, or ED^2P). If the the objective value decreases, then the system can adopt this new setting. Such techniques can also be accommodated to adapt to changes in environment and cooling apparatus.

5. EXPERIMENTAL RESULTS

This section provides the details of the experimental setup and results. We compare our technique to prior DVFS techniques in terms of accuracy in selecting optimal points, scalability, and energy-performance benefits.

We use an extensive data set (4158 samples) of power, temperature, and performance counter data collected across all workloads in the PARSEC benchmark suite [2] to train MLR models to predict optimal operating points. Each PARSEC benchmark is executed at all available system operating points, which include various combinations of DVFS settings and thread-counts. The data for each workload's region-of-interest (ROI), i.e., the parallel phase, is divided into 100 billion UOP execution intervals, and these data points are then aligned for each workload across all available system settings to determine the optimal point for each interval. We then train the MLR classifier using the methodology described in Section 4, and assess the accuracy of our classifier in predicting the optimal operating point for each interval. Our experimental setup is as follows.

- All data collection is performed on an Intel Core-i7 940 45nm quad-core processor, running the 2.6.10.8 Linux kernel OS.
- Performance counter data are collected using the `pfmon` (version 3.9) utility. This utility is configured to poll performance counters for each core at 100 ms intervals. The performance counters collect architectural information used for differentiating workload behavior: μ-ops retired, load locks, l3-cache misses, l2-cache misses, resource stalls, branch misses, and floating point operations.
- Each core on the Core-i7 processor is equipped with with a digital thermal sensor, measuring the maximum junction temperature. The pfmon tool is interfaced with the Linux `lm-sensors` library to report these per-core temperatures at 100 ms intervals in addition to the performance counter data.
- The power consumption of the processor and the total power consumption of the motherboard are measured using two Agilent A34401 digital multimeters.
- Power and performance counter data are collected for each workload in the PARSEC 2.1 benchmark suite at every available frequency and number of threads. The Core-i7 processor frequencies are manipulated with the Linux `cpufreq` library functionality, and can be set using DVFS to any frequency in the following set: {1.60 GHz, 1.73 GHz, 2.00 GHz, 2.13 GHz, 2.40 GHz, 2.53 GHz, 2.67 GHz}. The number of threads employed by each benchmark is set using the command line options provided in the PARSEC suite. Within each benchmark execution, data are only recorded for predefined ROI points, which excludes the serial boot-up and output phases of the applications.
- The power, temperature, and performance counter data are synchronized and logged with in-house software on a separate machine via an ethernet connection.

We implement previous methods from Isci *et al.* [9] and Dhiman *et al.* [5] for comparison purposes. Dhiman et al. uses a CPI based metric, μ , and Isci et al. uses Mem/μ-op metric to choose the appropriate v-f setting for the current workload. We set the threshold values for μ and Mem/Uop metrics in a such way that the overall accuracy is maximized. For each objective function, we construct specific v-f tables that will produce the most accurate decision. In the case of three available v-f settings, if $\mu < 0.85$ or $Mem/Uop \geq 0.15$, we apply the lowest available frequency (1.60 GHz). If $\mu \geq 0.90$ or $Mem/Uop < 0.03$ we apply the highest frequency (2.67 GHz). For the values in between we use 2.00 GHz.

In our first experiment, we assess the accuracy of our classifier for a series of potential objective functions using the training methodology described in Section 4. The resulting accuracies are reported in Table 3. The table shows that the percentage of execution intervals for which the classifier accurately predicts the optimal DVFS setting and thread counts. The candidate operating points for this experiment include each of the nine DVFS/thread-count pairings from the sets displayed in the first line of Table 4. The overall reported accuracy is the number of execution intervals correctly predicted across all workloads (as opposed to the average accuracy value).

The proposed technique is able to achieve a minimum of 51% increase in accuracy for the $minEDP$ classifier in comparison to the prior approaches. Our MLR-based technique is able to discern subtle differences in workload characteristics and their effects on energy and delay. The previous techniques rely on static threshold values that are set during an offline analysis. Thus, they are not sufficiently flexible to adapt to variations during the execution. These disadvantages of the previous techniques result in significantly lower accuracy values. Our technique also has high accuracy in predicting operating points for $minDPC$ and $minEPDC$ constraints across a range of power-delay constraints as shown in Table 3.

In our second experiment, we assess the scalability of the proposed technique when a higher number of operating points is available. In addition to the nine points considered previously, we calculate the overall accuracy of each classifier as the number of points increases to 12 and to 21 points as specified in Table 4. Increasing the number of possible outputs makes the classification more difficult. Not only are there fewer training examples for each output that can be used to estimate a classifier, but the differences in

Benchmark	Previous Approaches		Proposed Approach					
	$minEDP$ (Isci et al.)	$minEDP$ (Dhiman et al.)	$minEDP$	$minED^2P$	$minDPC$ (20W)	$minEPDC$ (20W, 10%)	$minEPDC$ (20W, 25%)	$minEPDC$ (50W, 25%)
blackscholes	5.9%	5.9%	98.7%	100%	100%	100%	98.8%	100%
bodytrack	63.6%	63.6%	72.9%	92.5%	97.7%	100%	100%	100%
canneal	100%	100%	100%	100%	100%	100%	100%	100%
dedup	50.0%	50.0%	88.3%	90.0%	100%	100%	100%	99.3%
facesim	0%	0%	99.7%	70.6%	100%	100%	97.2%	95.7%
ferret	57.9%	57.9%	62.5%	100%	100%	100%	100%	100%
fluidanimate	0%	8.7%	88.8%	95.0%	99.6%	100%	99.3%	98.6%
freqmine	3.0%	3.0%	77.4%	100%	100%	100%	99.7%	99.7%
raytrace	7.7%	92.3%	96.2%	100%	100%	100%	100%	100%
streamcluster	100%	100%	99.4%	100%	100%	100%	99.2%	100%
swaptions	5.0%	95%	100%	100%	58.6%	100%	97.8%	100%
vips	11.1%	11.1%	77.7%	56.9%	100%	100%	100%	60.7%
x264	16.7%	16.7%	73.2%	97.8%	97.8%	100%	88.1%	95.8%
Overall Accuracy	20.7%	36.4%	87.4%	92.5%	95.5%	100 %	99.3%	98.8%

Table 3: 5-fold cross-validated test accuracies for various objective formulations.

workload behavior among the outputs are much more subtle. Nevertheless, the results in Figure 5 illustrate that the accuracy of our technique for the $minEDP$ and $minED^2P$ is not only greater in value than for the previous techniques, but also scales better to a higher number of operating points. This experiment indicates that through higher sensitivity to fine-grained workload characteristics, our technique is more viable than previous techniques in an HPC cluster or datacenter scenario with potentially a very large number of operating points. Figure 5 shows successful scalability of our technique for the $minDPC$ and $minEPDC$ classifiers.

# Settings	Frequencies (GHz)	Thread Counts
9 settings	{1.60, 2.00, 2.67}	{1, 2, 4}
12 settings	{1.60, 2.00, 2.40, 2.67}	{1, 2, 4}
21 settings	{1.60, 1.73 , 2.00, 2.13, 2.40, 2.53, 2.67}	{1, 2, 4}

Table 4: Definition of the operating points used in each of the experiments.

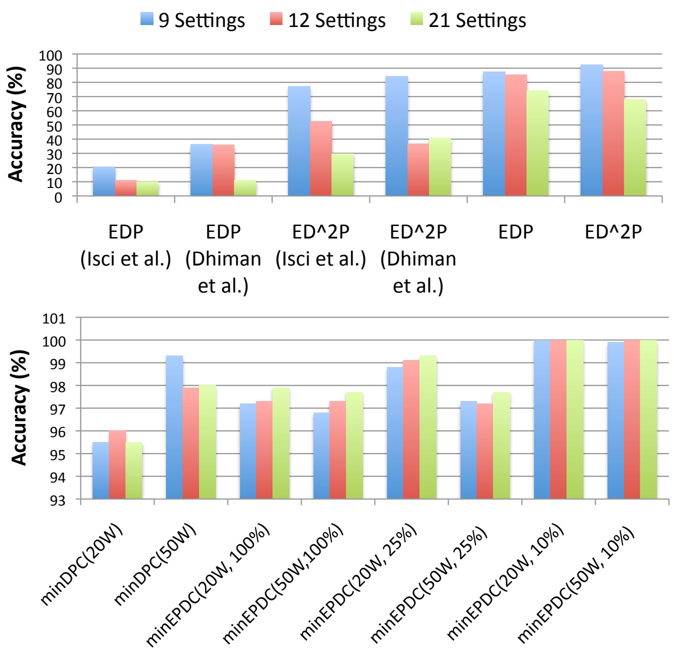

Figure 5: Optimal point accuracy for various methods as a function of the number of available settings.

Finally, we measure the impact of the classification accuracy on the EDP for each of the PARSEC workloads with 21 classifier outputs. The results show that overall, our MLR-based technique is capable of reducing the EDP by 16.9% relative to the approach in Isci *et al.*, and 10.9% relative to the approach by Dhiman *et al.* on average. It also achieves a maximum reduction in EDP of 33.2% and 30.9% relative to Isci *et al.* and Dhiman *et al.* respectively for the `vips` benchmark.

6. CONCLUSIONS

In this paper we have explored the optimal control settings (e.g., number of threads and DVFS settings) of multi-threaded workloads running on a multi-core processor. By optimally adjusting the values of these settings across and within workloads, it is possible to achieve better performance-energy operation. In contrast to previous approaches that focused on optimizing one formulation (e.g., minimizing ED^{2}P), we explored the impact on performance and energy as a function of a number of problem formulations. We investigate four problems formulations: (i) minimize EDP, (ii) minimize ED^{2}P, (iii) minimize delay under peak power constraints, and (iv) minimize energy under peak power and delay constraints. We have found that these formulations do not necessarily dominate each other, and that the best choice is driven by the context of the application and operator goals. To handle such diverse range of formulations, we have proposed a multinomial logistic regression classifier that is capable of finding the optimal operating points across and within workloads during execution. The classifier is trained using input data from performance counters, runtime delays, and energy consumption measurements. To minimize the runtime penalty of classification and to avoid over fitting, it is necessary that the classifier uses a minimal number of performance counters to arrive to its results. Thus, we have proposed L_1-regularization techniques to naturally discover the inputs that are most relevant towards determining the optimal settings. Using a solid experimental setup in which performance counter values and power measurements are collected on a real quad-core based system running the PARSEC parallel workloads, we have demonstrated that our method is capable of achieving far greater accuracy that previous methods. This accuracy in deciding the optimal operating points translates to a 10.9% improvement in EDP over the

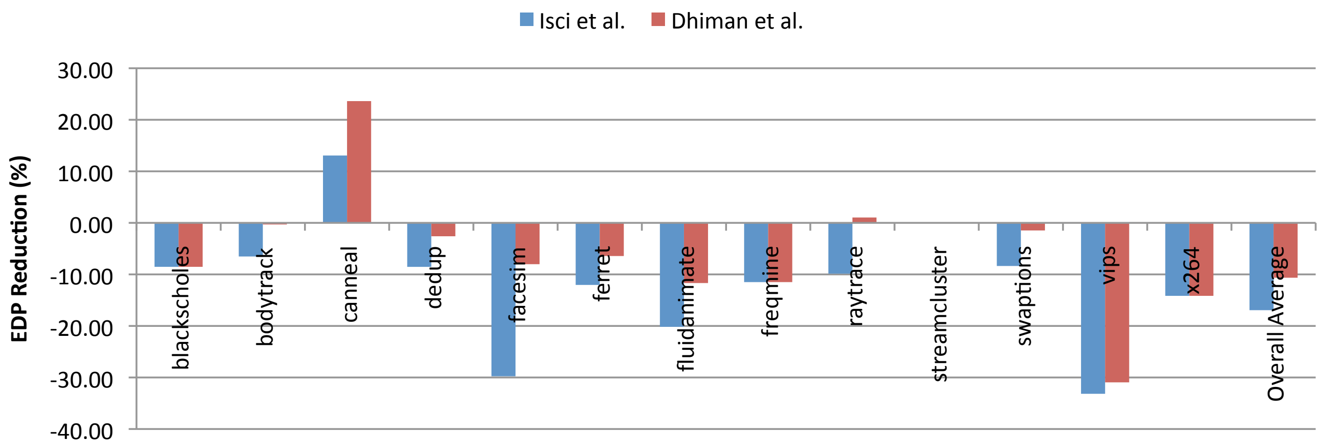

Figure 6: EDP reduction percentage of proposed technique relative to previous methods for 21 candidate operating points.

best performing previous method, with a maximum improvement of 30.9%.

7. ACKNOWLEDGEMENTS

This research has in part been funded by Dean's Catalyst Award at College of Engineering, Boston University. R. Cochran and S. Reda are partially supported by NSF grants number 0952866 and 1115424.

8. REFERENCES

[1] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints. In *Proceedings of Design, Automation and Test in Europe Conference*, 2002.

[2] C. Bienia and K. Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[3] K. Choi, R. Soma, and M. Pedram. Dynamic voltage and frequency scaling based on workload decomposition. In *Proceedings of international symposium on Low power electronics and design*, 2004.

[4] M. Curtis-Maury, F. Blagojevic, C. D. Antonopoulos, and D. S. Nikolopoulos. Prediction-based power-performance adaptation of multithreaded scientific codes. *IEEE Trans. Parallel Distrib. Syst.*, 19:1396–1410, October 2008.

[5] G. Dhiman and T. S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Proceedings of International Symposium on Low PowerElectronics and Design*, 2007.

[6] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Optimizing job performance under a given power constraint in hpc centers. In *Proceedings of the International Conference on Green Computing*, 2010.

[7] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of International Symposium on Low PowerElectronics and Design*, 2007.

[8] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the 39th International Symposium on Microarchitecture*, 2006.

[9] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of the 39th International Symposium on Microarchitecture*, 2006.

[10] W. Kim, M. S. Gupta, G. yeon Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *International Symposium on High-Performance Computer Architecture*, 2008.

[11] J. G. Koomey. Worldwide electricity used in data centers. *Environmental Research Letters*, 3(3):034008, 2008.

[12] Li and J. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *International Symposium on High-Performance Computer Architecture*, 2006.

[13] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho. Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. In *International Symposium on Computer Architecture*, 2003.

[14] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, ASPLOS '09, 2009.

[15] M. Moeng and R. G. Melhem. Applying statistical machine learning to multicore voltage & frequency scaling. In *Conf. Computing Frontiers*, 2010.

[16] G. J. Narlikar and G. E. Blelloch. Pthreads for dynamic and irregular parallelism. In *In Proc. of Supercomputing Õ98*, pages 4–1. IEEE, 1998.

[17] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *Proceedings of the 38th Conference on Design Automation*, 2001.

[18] K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Computer Architecture News*, 2009.

[19] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *International Symposium on High-Performance Computer Architecture*, 2008.

[20] H. Yu, B. Veeravalli, and Y. Ha. Dynamic scheduling of imprecise-computation tasks in maximizing qos under energy constraints for embedded systems. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, 2008.

[21] Y. Zhu and F. Mueller. Feedback edf scheduling of realtime tasks exploiting dynamic voltage scaling. *Real-Time Systems Journal*, 2005.