Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps

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 K. 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

The ability to cap peak power consumption is a desirable feature in modern data centers for energy budgeting, cost management, and efficient power delivery. Dynamic voltage and frequency scaling (DVFS) is a traditional control knob in the tradeoff between server power and performance. Multi-core processors and the parallel applications that take advantage of them introduce new possibilities for control, wherein workload threads are packed onto a variable number of cores and idle cores enter low-power sleep states. This paper proposes **Pack & Cap**, a control technique designed to make optimal DVFS and thread packing control decisions in order to maximize performance within a power budget. In order to capture the workload dependence of the performance-power Pareto frontier, a multinomial logistic regression (MLR) classifier is built using a large volume of performance counter, temperature, and power characterization data. When queried during runtime, the classifier is capable of accurately selecting the optimal operating point. We implement and validate this method on a real quad-core system running the PARSEC parallel benchmark suite. When varying the power budget during runtime, **Pack & Cap** meets power constraints 82% of the time even in the absence of a power measuring device. The addition of thread packing to DVFS as a control knob increases the range of feasible power constraints by an average of 21% when compared to DVFS alone and reduces workload energy consumption by an average of 51.6% compared to existing control techniques that achieve the same power range.

Categories and Subject Descriptors

D.4.1 [**Operating Systems**]: Process Management—*Scheduling*

General Terms

Measurement, Experimentation, Design

1. INTRODUCTION

With modern data centers growing larger and denser in order to meet increasing computational demand, energy consumption is fast becoming the largest contributor to the total cost of ownership of data centers and high performance computing (HPC) clusters [10, 5]. Thus, it is critically important to plan and manage facility power and cooling resources. Traditionally, each data center is given a maximum power constraint and is populated with the maximum number of server units that will stay within this limit minus a safety margin. These margins are carefully designed to avoid power consumption peaks that would trigger circuit breakers and interrupt execution. Constraints on power consumption also implicitly limit energy consumption and the associated costs.

A number of techniques for *power capping* have emerged to assist data center administrators in maintaining average power budgets [10, 5, 11] and in guaranteeing peak power constraints [13]. Some industry solutions manage the power budget of a set of nodes within a rack by allocating more power capacity to busy units and reducing the power constraints on idle nodes [1]. State-of-the-art techniques with fast response capability to sudden power spikes utilize fine-grained dynamic power monitoring enabled by power meters [2]. Once a power budget is allotted by higher-level software to server nodes, each node manages power within the desired cap individually.

Modern chips come equipped with a set of power management features: dynamic voltage and frequency scaling (DVFS) (e.g., P-states in Intel chips), clock gating, and automatic sleep states for unused processor units (entire cores as well as individual functional units, such as the L3-cache). Prior art has shown that it is possible to significantly improve energy efficiency by augmenting hardware energy-management features with OS-level control knobs, such as thread allocation and scheduling (e.g., [22]). These techniques are further enhanced when understood and performed at fine granularities (e.g., [14]). *Power capping* allows for energy management across multiple server nodes using these chip-level control mechanisms [10].

This paper proposes a novel technique for maximizing performance within variable power caps for multithreaded workloads. Recent trends show that common workloads on data center and HPC clusters are increasingly employing thread-level parallelism in order to capitalize on the increased hardware parallelism in multi-core systems. These workloads span a wide range of application domains, from modeling and scientific computing to financial applications and media processing. Parallel workloads bring additional challenges as well as opportunities for control. Parallel workloads have complex thread synchronization methods, making runtime performance and power analysis significantly more complicated. On the other hand, parallel workloads offer additional control knobs, such as selecting the number of active threads, for managing power-performance tradeoffs.

The proposed technique, **Pack & Cap**, is designed to optimally manage two control knobs within a single server node during runtime: (1) the voltage-frequency (V-F) setting, and (2) thread pack-

*This research is in part 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.

MICRO'11, December 3-7, 2011, Porto Alegre, Brazil

ing. Thread packing specifies how many threads should run on how many cores, and is used for packing multithreaded workloads onto a variable number of active cores. **Pack & Cap** employs thread packing and DVFS during runtime in order to optimize workload performance within a power cap, thus increasing energy efficiency. Our specific contributions are as follows:

- We elucidate the impact of DVFS and thread packing on the performance and energy consumption of a server operating within a power cap. We quantify the difference between thread packing and thread reduction (i.e., decreasing the number of threads), and demonstrate that our technique is capable of meeting a larger range of possible power caps in comparison to DVFS alone.
- We devise an offline *multinomial logistic regression (MLR)* classifier that characterizes the optimal thread packings and V-F settings as a function of user-defined peak power constraints. A large volume of performance counter data, power measurements, and temperature sensor data are collected on a real quad-core (Intel's Core i7) based system and are used to train the MLR classifier. We propose L1-regularization techniques to find the best set of inputs for the classifier.
- We query our MLR classifier during *online* operation to estimate the probability of each operating point (V-F setting and thread packing combination) yielding the optimal workload performance within a power constraint, and to select the likeliest optimal operating point. Our online classifier only makes use of performance counter and thermal sensor data. It does not require power measurements during online operation, which makes it an attractive low-cost technique.
- We implement **Pack & Cap** on a real quad-core (Intel's Core i7) based server and evaluate the performance and energy dynamically for the PARSEC parallel benchmark suite [6]. In our experiments, we adjust the desired power cap during runtime and conclusively demonstrate that our method maximizes workload throughput while adhering to the power budget 82% of the time. We also demonstrate that a DVFS-focused version of our classifier (i.e., without thread packing) outperforms existing DVFS techniques in the literature.

The remainder of the paper begins with a detailed explanation of the motivation for integrating thread packing and DVFS in Section 2. Section 3 explains the MLR classifier and the runtime control. In Section 4, we provide the experimental setup and evaluate **Pack & Cap** in comparison to various DVFS and thread packing settings. Section 5 discusses prior work and Section 6 concludes the paper.

2. MOTIVATION

Parallel workloads executing on multi-core processors introduce new challenges in optimization as performance and power consumption depend on the number of threads, thread interactions, and the number of available cores. In this paper, we investigate new techniques for maximizing the performance of parallel workloads running on a multi-core based system within a power cap. We consider two *control knobs*: DVFS and thread packing, where we define *thread packing* as the process of confining workload threads onto a variable number of cores. Thread packing differs conceptually from thread allocation in that we make no distinction among identical cores; i.e., if n of the threads are packed onto $x < n$ cores with identical resources, it does not matter which cores are chosen. Thread packing is plausible due to core symmetry in existing homogenous multi-core processors. In this work we consider a homogenous system without hyperthreading, which we disable for our test system in the BIOS settings. However, our approach is sufficiently flexible that it could easily be modified to distinguish between sets of logical cores on a hyperthreaded or heterogenous system if necessary.

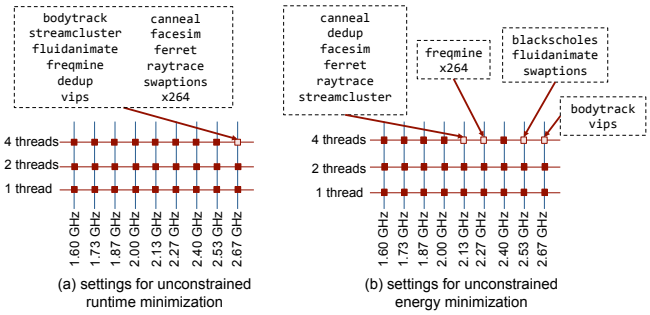

Figure 1: Optimal settings in the absence of power constraints.

Operation under Unconstrained Power. To motivate our approach we first explore the optimal V-F settings and thread configurations in the absence of power consumption constraints. Our quad-core processor test platform offers 9 V-F settings and we execute each PARSEC benchmark with 1, 2 or 4 threads using the native input set, where each thread is assigned to a different core. This yields a total of 27 possible settings corresponding to each V-F and thread-count combination. We identify the optimal settings for the first 100 billion retired μops in the parallel phase of each PARSEC benchmark. In Figure 1 we demonstrate (1) the settings that give the minimum runtime and (2) the settings that give the minimum energy. In both cases the results unanimously show that in the absence of any constraints on power consumption, the 4-thread setting is the best regardless of the objective. While it is obvious that the highest V-F setting will always give the minimum workload runtime, this is not necessarily the case for energy minimization. To provide more insight into the behavior of the workloads at various V-F settings, we plot in Figure 2 the workload energy (in KJ) versus runtime (in s) for four representative PARSEC workloads. The plots show convex-like energy-runtime curves, where the optimal point of each workload is determined by the tradeoff between power consumption and runtime savings as a function of frequency. In many cases the increased power consumption associated with higher V-F settings is not offset by sufficient reductions in runtime, leading to higher energy consumption.

Thread Packing versus Thread Reduction. We define *thread reduction* as the traditional process of launching an application with fewer number of threads than there are available cores. We seek to understand the difference in power and runtime between executing 4 threads of an application packed on i cores, versus executing i threads of the same application on i cores, where each thread is assigned to a different core. If $i = 4$ there is no difference between thread packing and thread reduction in terms of runtime and power consumption. If $i < 4$, there are *two* relevant comparison *cases* to investigate. In the *first case*, 4 threads packed onto a 1 core is compared to 1 thread executing on 1 core. In the *second case*, 4 threads packed on 2 cores is compared to 2 threads executing on 2 cores. While it is possible to pack 4 threads on 3 cores, some PARSEC workloads do not support execution with 3 threads.

To conduct a systematic comparison across all benchmarks, we execute a static comparison, where the number of threads does not change once a workload is launched. There are a total of 117 comparison points, where each point corresponds to one V-F setting (from a total 9 V-F settings) and one PARSEC benchmark (from a

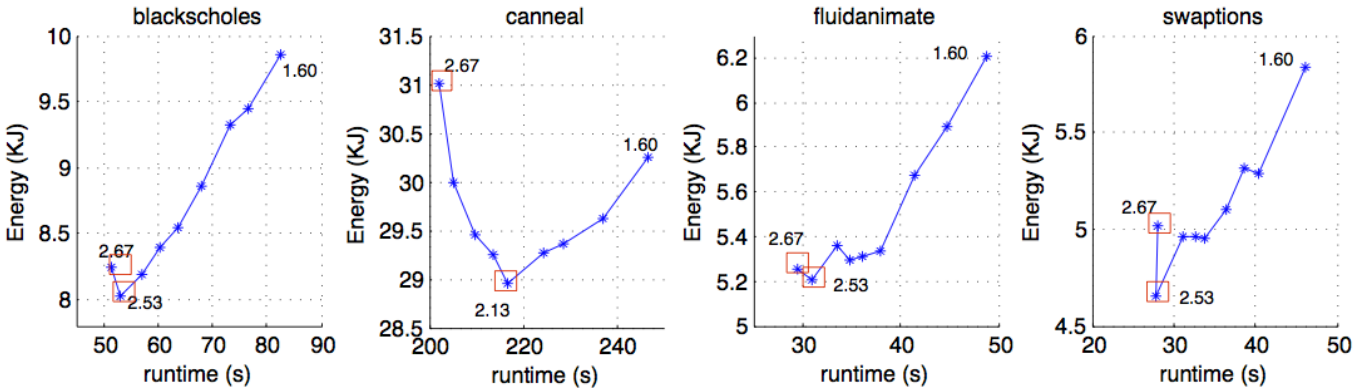

Figure 2: Impact of V-F settings on energy and runtime for the first 100 billion μops retired within the parallel phase of four representative workloads running with 4 threads. Optimal runtime and energy settings are marked with red squares.

	First case: 1 active core		Second case: 2 active cores	
	packing	reduction	packing	reduction
	4 threads on 1 core	1 thread on 1 core	4 threads on 2 cores	2 threads on 2 cores
runtime	1.000×	1.021×	1.000×	1.010×
power	1.000×	0.990×	1.000×	0.993×

Table 1: Comparison between thread packing and thread reduction. Results are an average of 117 comparison points (9 V-F settings × 13 PARSEC workloads) and normalized with respect to thread packing.

total of 13 benchmarks). Table 1 gives a summary of the comparison between thread packing and thread reduction.

- **Runtime.** For the first case of 1 active core, the runtime of thread reduction is larger by 2.1% in comparison to thread packing. For the second case of 2 active cores, thread reduction is larger by 1.5%.
- **Power.** For the first case of 1 active core, the power consumption of thread reduction is smaller by 1.01% than thread packing. For the second case of 2 active cores, it is smaller by 0.70%.

Ideally, we would also like to contrast thread reduction and thread packing in a dynamic setting. However, we are not aware of graceful mechanisms for changing the number of active threads of a parallel workload at an arbitrary execution point without considerable modification to the application code.[1] The results from thread reduction and thread packing show that the power and performance values are comparable to each other, with packing having a slight advantage in performance. In contrast to thread reduction, thread packing has the advantage of being easily modifiable. In a Linux environment, thread packing is implemented by setting the core affinities of the threads using the associated thread IDs. Thread packing has negligible implementation overhead as it does not require modifications to applications. Thus, thread packing is a more feasible and efficient solution for adaptive computing.

[1]Depending on the parallelization model and the software structure, it is possible to change the number of active threads during workload execution for each parallel region of the application. `pthreads` and `OpenMP` are the two commonly used parallelization models, and are represented in the PARSEC suite. Our experiments with the `bodytrack` benchmark show that changing the number of active threads from 2 to 4 through `OpenMP` scheduling clauses has similar performance in comparison to maintaining 4 threads and packing them on 2 or 4 cores.

Impact of Packing on Power Range. The combination of DVFS and thread packing increases the system's *power range* for a given application. We define the power range as the difference between the maximum peak power attained at the highest setting (4 active cores at 2.67 GHz) and the minimum peak power attained at the lowest setting (1 active core at 1.60 GHz). Figure 3 gives the power range of the PARSEC workloads on our system. The results show that the DVFS and thread packing combination consistently delivers a larger range than with DVFS alone, with an average increase in the power range by 21%.

Optimal DVFS and Thread Packing Under Power Caps. We previously examined the optimal operating points in the case of unconstrained power operation. We next compute the optimal operating points under a range of possible server power caps. We first consider runtime minimization under power constraints. Our system has 4 cores and 9 V-F settings for a total of 36 V-F and thread packing combinations. Figure 4 gives the peak power and runtime of the first 100 billion retired μops at all possible settings in the parallel phase of 4 representative PARSEC workloads. We also mark *power-runtime Pareto frontier* with a red line. For each point along the Pareto frontier, there is no alternative point that achieves lower peak power *and* shorter runtime. Thus, these points *dominate* the remaining points, and the Pareto frontier gives the optimal settings for runtime minimization as a function of the power cap. Note that each workload segment has its own characteristic Pareto frontier. Identifying the most suitable setting is dependent on workload characteristics, which may vary during execution.

Second, we consider energy minimization within power constraints. Figure 5 gives the peak power and energy of the first 100 billion retired μops at every settings in the parallel phase of four representative PARSEC workloads. We also mark *power-energy Pareto frontier* with a red line, where the points on the frontier dominate the other settings with reduced peak power *and* energy.

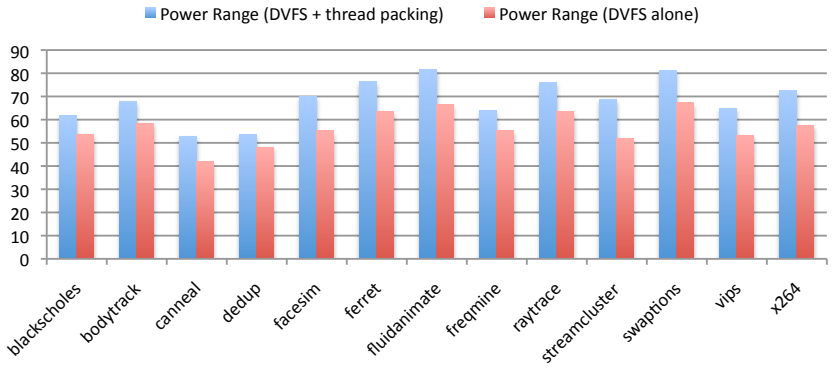

Figure 3: Dynamic power range of DVFS and thread packing versus DVFS alone.

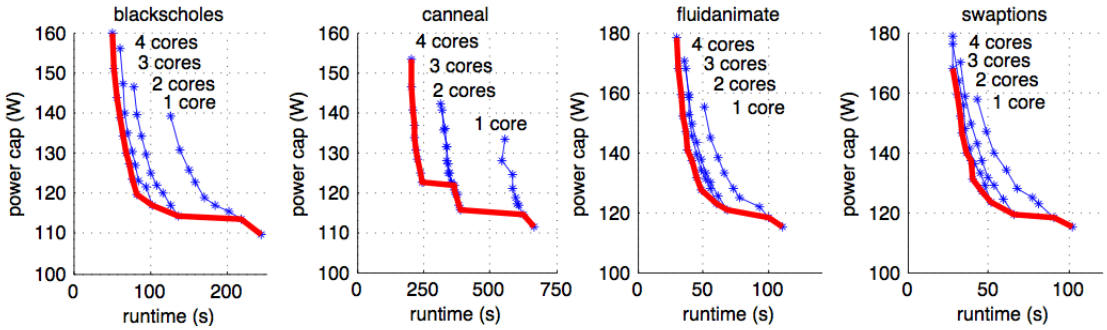

Figure 4: Impact of V-F and #thread settings for runtime minimization under power caps. Red line gives Pareto frontier of optimal settings at various power caps.

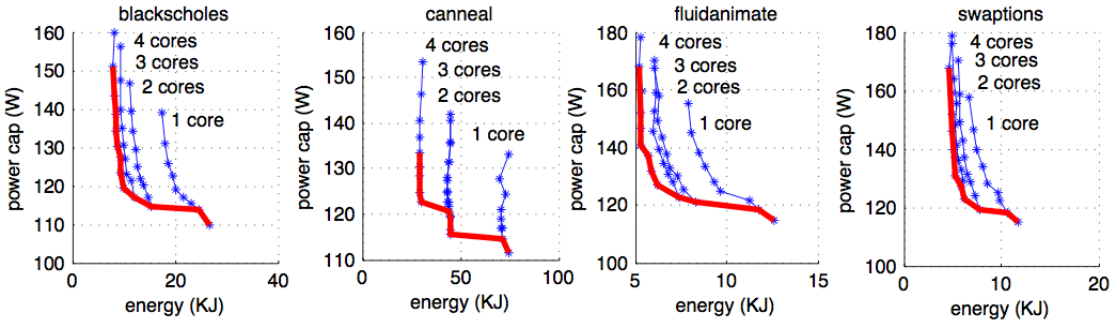

Figure 5: Impact of V-F and #thread settings for energy minimization under power caps. Red line gives Pareto frontier of optimal settings at various power caps.

The Pareto frontier gives the optimal settings for energy minimization as a function of the power cap. Note that the first point of the Pareto frontier naturally starts at the energy optimal setting given earlier in Figure 1. We denote the power consumption of this starting point for unconstrained minimum-energy operation with p_e and the power cap with p_c. Figure 4 and Figure 5 reveal the two following observations regarding runtime and energy minimization.

1. If $p_c > p_e$ then energy and runtime minimization lead to different settings. If the objective is energy minimization, the system should not scale its settings to utilize the available power budget because it will negatively impact energy consumption as given in Figure 5.
2. If $p_c \leq p_e$ then the system should scale its settings to give the closest power consumption to the given cap in order to minimize both runtime and energy. Thus, energy and runtime minimization lead to identical control decisions.

Because the second condition is often true for low to mid-range power constraints, in the process of optimizing for runtime we are simultaneously inducing significant energy savings in many cases. The important question of how to identify the optimal settings given workload dependencies is explored in depth in the following section.

3. PACK & CAP METHODOLOGY

Our approach for runtime thread packing and DVFS control is split between an offline step and an online step. In the offline step, we use an extensive set of data gathered for the parallel workloads

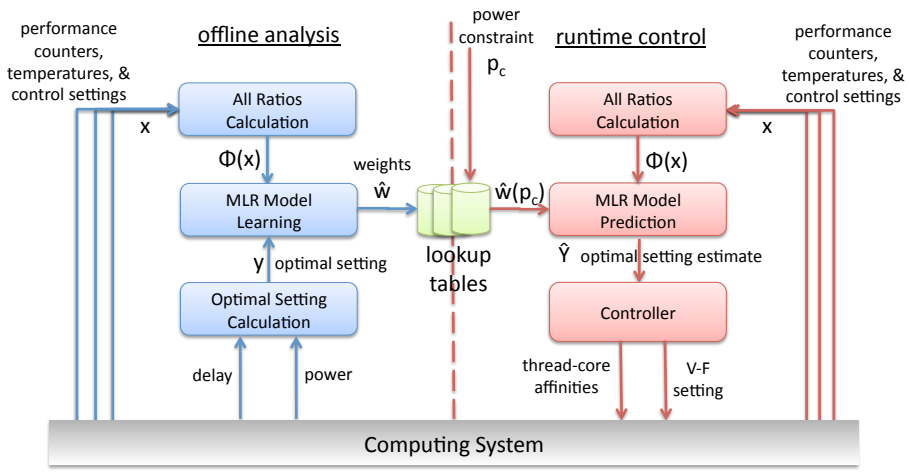

Figure 6: Overview of the Pack & Cap methodology.

in the PARSEC benchmark suite to train the workload-dependent classifiers. Each classifier takes performance counter and per-core temperature measurements as inputs, and outputs the system operating point with the highest probability of maximizing performance within a given power constraint. A classifier instance is trained for each desired power constraint. During runtime, we recall the model associated with the desired power constraint using a lookup table, and given the real-time measurements of various input metrics, the control unit sets the system operating point with the highest probability of being optimal. During runtime, our model curbs the system power within a power cap without using expensive server-level power measurement devices. Figure 6 gives a general outline of our approach.

3.1 Offline Characterization

The offline characterization step makes use of an L_1-regularized multinomial logistic regression (MLR) classifier [3]. One primary advantage of our offline characterization technique lies in the degree of automation. While previous techniques require manual input to select the performance metrics that are most relevant to energy, power and delay optimization, we use L_1-regularization to systematically find the relevant inputs and mask irrelevant ones. We can therefore start with a large set of input metrics, as our learning algorithm will only focus on the important ones. In determining the optimal inputs, we are able to develop general models that do not need to be tailored for specific workloads. Each objective formulation only requires a single model to encapsulate a wide range of workload behavior. In addition, by using a classifier that outputs a discrete value corresponding to the optimal setting, our approach does not require any intermediate estimates of the power or delay during runtime. Previous modeling techniques [17] rely on linear regression for estimating power and performance. Regression models predict continuous values representing power and performance as a function of performance counter inputs which are then used to inform the control decisions. By modeling the boundaries between the discrete decision outputs directly, we observe the MLR classifier to be a more stable predictor of optimal outcomes when compared to linear regression techniques, particularly when the test data differs significantly from the training data.

3.1.1 Multinomial Logistic Regression

The MLR classifier is a generalized linear model that estimates the probability of a discrete set of outcomes by fitting a multinomial logistic function to continuous input data [3]. In this work, the set of outputs corresponding to the set of feasible system operating points. Because the cores in our test setup are homogenous (identical resources) and hyperthreading is disabled, these operating points are only differentiated by the V-F setting and the number of active cores onto which the workload threads are packed. However, in systems that have heterogenous cores or hyperthreading, this approach can be easily adapted to differentiate various logical core subsets when training the MLR model. Whereas the models in our experiments treat 4 threads running on 2 logical cores at a particular V-F setting as a single operating point regardless of which 2 cores are active, in a heterogenous system the operating points would be differentiated based on core type.

The inputs to the MLR classifier include a set of workload metrics, which are functions of the system performance-counter values, per-core temperatures, and the current operating point. The logistic model classifier 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 under a particular objective formulation. The output with the highest probability is then chosen as the current operating point during runtime. In **Pack & Cap**, the optimal operating point corresponds to the setting that maximizes performance within a power constraint. However, the MLR classifier can be easily trained for other objective formulations such as energy, energy-delay product (EDP), or energy delay-squared product (ED^2P) minimization. The choice of objective formulation depends on the application.

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 nonlinear vector-valued 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 $\mathbf{w} \in \mathbb{R}^{Cm}$ 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 $\{1, ..., C\}$ 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. Measurements from the parallel phase of each benchmark are taken at each available V-F setting and thread packing, and the resulting data is divided into windows of fixed size in terms of the number of instructions retired (100 billion μops). By aligning these fixed instruction windows across all possible operating points, 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 function of the training data, expressed in Equation (2):

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} \left(-\sum_{i} \log Pr(y_i|\mathbf{x_i}, \mathbf{w}) \right), \tag{2}$$

where $\mathbf{x}_i$ 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.

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 evaluated on separate test data. We define accuracy as the percentage of execution intervals for which the true optimal system operating point is correctly predicted by the classifier.

3.1.2 L_1-Regularization and Input Selection

With a large number of inputs, it is possible to *over-fit* the classification model to the training data, producing excessively high weight estimates. While the model will fit the training data well, any minute disturbance in any input data not used to train the model will cause dramatic changes in the classifier output. One standard method of preventing over-fitting is L_1-regularization, in which an L_1-loss term is added to the loss function in Equation (2) as is shown in Equation (3).

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

The loss term induces a penalty for large weight values, and prevents the model from being over-fit to the training data. 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. The optimal degree of regularization α is experimentally determined by measuring the cross-validated test accuracy for α for a series of values spanning several orders of magnitude. The minimization in Equation 3 is performed using an iterative projected gradient descent method, which searches for the optimal weights within the boundaries induced by L_1-regularization.

L_1-regularization also has another extremely useful property in that it forces sparse solutions for $\mathbf{w}$ (i.e., solutions where many of the coefficients are zero or of negligible magnitude). Thus, L_1-regularization can identify the inputs that are most relevant to classification and eliminate irrelevant ones. Given the high complexity of parallel workload behavior, it is not immediately obvious what sort of inputs are useful in classification for various power, energy and delay objectives. Using L_1-regularization, however, the number of inputs can be almost arbitrarily high, and the most relevant inputs will be identified while 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 temperature* across all cores, *V-F setting*, *active-core count*, and a *constant term*. However, the vector-valued function of the inputs $\phi(\mathbf{x})$ has much higher dimensionality, taking every possible ratio of any two 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 ones.

Because the MLR model must perform thread packing within the power budget without significantly degrading performance, it must take into account the complex synchronization behaviors among the workload threads. We chose the PARSEC benchmarks for modeling and testing purposes because the workloads span a large range of thread interactions and parallelization models. By taking shared memory behavior (L3-cache misses) and synchronization behavior (load locks) into account, the MLR models encapsulate these dependencies. For instance, if the L3-cache were to decrease in size, the model would register this as a stronger negative correlation between performance and L2-cache misses, and the performance impact of thread packing would increase. In fact, the ratio of L3- to L2-cache misses is generally observed to be the metric with the highest logistic weight magnitudes across the different outputs.

3.2 Adaptive Runtime Control

The runtime overhead of the proposed technique is minimal, as the model weights are communicated from the offline method to the runtime method in the form of a lookup table as illustrated in Figure 6. The system simply logs performance counter and temperature data, and at regular intervals calculates the probability of each operating point being optimal according to Equation (1) using the set of weights corresponding to the current power constraint p_c. The system adapts to changes in the power constraint by retrieving the associated set of model weights that were learned offline. The overhead for each activation of our control algorithm is in the range 10 ms - 50 ms. With a control activation period on the order of seconds, which permits reasonable responsiveness, this overhead represents a very tiny fraction of the overall execution time.

Once an optimal point is predicted, this information is passed to a controller that enforces the setting on the system. The runtime overhead for performing DVFS control is very low, and is on the order of 100 μs [15]. The overhead for shifting threads among the cores for thread packing is potentially higher (reaching millisecond range in some cases). However, these effects are automatically factored into our delay estimation in the experiments, which nevertheless show significant improvement in performance, power, and energy.

4. EXPERIMENTAL RESULTS

This section provides the details of the experimental setup, as well as the results showing that our adaptive **Pack & Cap** approach is capable of leveraging DVFS and thread packing to maximize performance within an arbitrary power cap. We first quantify the ability of our runtime approach to adapt to changing power constraints and workloads. We then compare the performance of our adaptive approach in terms of the execution delay against the delay for a known optimal static setting for each benchmark. Third, we show that adding thread packing as a control knob leads to better tradeoff between power and performance, and superior adherence to system power cap when compared to pure-DVFS. Finally, we verify that our L_1-regularized MLR method for DVFS alone outperforms the

accuracy of existing energy-aware DVFS approaches.

In the offline characterization step, we use an extensive set of power, temperature and performance counter data collected for each PARSEC benchmark at all feasible system operating points (V-F and thread packing combinations). For each workload's parallel phase (region of interest, ROI), we divide the data into 100 billion μ-op execution intervals. By aligning the data points for each workload across all available system settings, we are able to measure the *true* optimal point for any objective function for each interval. We then train an MLR classifier for each desired power constraint using the methodology explained in Section 3. Our experimental setup for data collection and online control is as follows:

- All collection and control experiments are 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. We poll performance counters for each core at 100 ms intervals. The 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*. The PARSEC benchmarks do not make use of the network card or graphical processor; thus, we found no value for collecting usage statistics from these components.
- Each core on the Core-i7 processor is equipped 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.
- The server's total power consumption is measured using an Agilent A34401 digital multimeter.
- We control the system operating points (V-F settings and thread-packing combinations) using Linux C library interfaces. The Core-i7 processor frequencies are manipulated with the `cpufreq` utility, and the available frequency settings are: {1.60 GHz, 1.73 GHz, 1.87 GHz, 2.00 GHz, 2.13 GHz, 2.27 GHz, 2.40 GHz, 2.53 GHz, 2.67 GHz}. The voltages associated with each frequency are set automatically in hardware. The thread-packing assignments are controlled using using the `sched_setaffinity` interface in the Linux scheduler.
- We disable the hyperthreading feature in our processor using our system's BIOS settings.
- To implement data collection and runtime control, we interface our data measurement and control apparatus to a MATLAB module compiled as a C-shared library. This module is configured to read lookup tables generated offline, buffer incoming performance counter and temperature data, and periodically output control decisions to a control unit. The runtime overhead for each activation of the control algorithm during runtime is in the range of 10-50ms.
- The number of threads for each benchmark is set using the command line options provided in the PARSEC suite.

In our first experiment, we demonstrate that our adaptive runtime approach consistently obeys a wide range of power constraints, regardless of workload behavior or physical operating conditions. During the execution of each parallel workload, we periodically change the power constraint to a random value in the 110W - 180W range, and measure the percentage of the execution time for which the power is within a tolerance of the cap value. A cumulative distribution function (CDF) for each benchmark is given in Figure 7, where the CDF plot gives the percentage of time the power caps

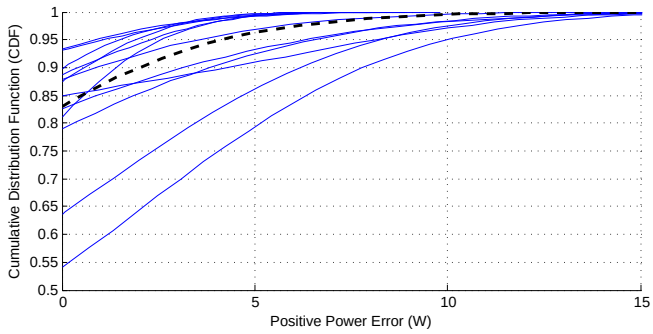

Figure 7: Cumulative distribution function (CDF) for the positive deviation of the observed power from the power cap for each PARSEC benchmark. The black dotted line represents the CDF for the entire data set on the average.

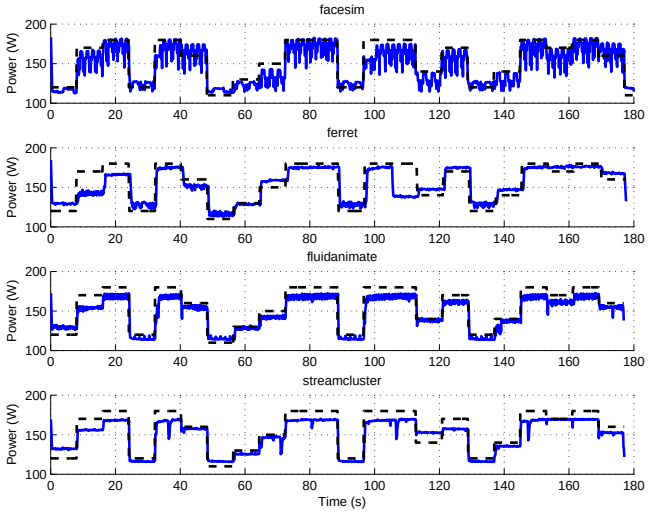

Figure 8: Measured power traces for facesim, ferret, fluidanimate, and streamcluster benchmarks as Pack & Cap adapts performance to constrain the power consumption with the power cap (black dotted line). The power cap is randomly modulated every 8 seconds.

are met within a certain tolerance. The average CDF is given by the dashed line. The results show that we are able to constrain the power consumption within the given cap (1) 82% of the time on average across all workloads, (2) 96% of the time within a 5W margin beyond the power cap, and (3) $>$ 99% of the time within a 10W margin.

The power traces in Figure 8 further confirm that **Pack & Cap** consistently adheres to the power constraint. Figure 8 also confirms that when operating along the Pareto frontier in power-delay space, minimizing delay under a power constraint is equivalent to maximizing the power within that constraint. Thus, the measured power for each benchmark follows closely below the power constraint. By encapsulating the workload dependence of the power-delay Pareto frontier in the MLR model, our runtime control scheme is able to maximize power budget usage in the face of dynamic workload behavior and without using server-level power meters. While we use power measurement data to verify this assertion and during offline model generation, we do not utilize any power measurements during runtime control. Figure 9 illustrates the changes in the DVFS

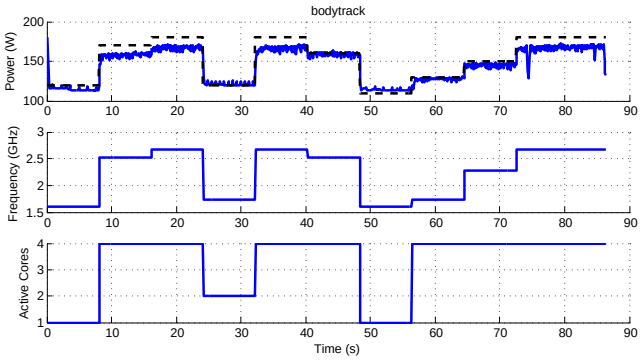

Figure 9: Demonstration of DVFS and thread-packing control for `bodytrack` under changing power caps.

and thread packing control knobs in response to the changing power constraints for the `bodytrack` benchmark.

Whereas in the first experiment we demonstrate that **Pack & Cap** adheres to the power cap even without power measurement devices, in our second experiment we demonstrate that the MLR classifier successfully predicts the operating points that optimize workload performance within the given power constraint. We execute each benchmark with a low-range and mid-range power constraint (120W and 150W respectively) and measure the execution delay for the parallel phase of each workload. For reference, we develop an *oracle* controller that knows *a priori* the global optimum setting for each benchmark. We accomplish this by executing every PARSEC workload at every available V-F setting and thread packing and selecting the setting with the lowest runtime that achieves the same average power as the proposed approach.

Figure 10 compares the runtimes for adaptive MLR to the static oracle for all workloads at power budgets of 120W and 150W. We expect that if the proposed approach is truly operating along the power-performance Pareto frontier, then the MLR delay should approximate the oracle delay. The results show that indeed the delays are very close in value, and in fact adaptive MLR actually achieves superior performance in many cases. These improvements are due to the fact that the adaptive approach allows the system setting to change in response to dynamic workload and operating conditions, whereas the oracle only optimizes according to the average behavior of the workload as a whole. The performance improvement achieved by our dynamic approach compared to the static oracle is more stark for the more constrictive power constraint. This is due to the fact that in the static approach, the operating point is set for the entire duration of the workload such that the most power-intensive execution phase is within the low constraint. By allowing the operating point to dynamically change depending on the workload characteristics, the dynamic approach introduces opportunities for the controller to boost performance as the workload enters less power-intensive phases.

In a third experiment, we quantify the benefits of thread packing as a control knob. In Figure 3, we have shown that thread packing increases the range of achievable power constraints over pure-DVFS. In the absence of thread packing, one can only resort to thread reduction in order to achieve the same range. This corresponds to running 1 or 2 threads on a quad-core machine (odd number of threads not supported by many PARSEC workloads). Barring special thread interfaces such as `OpenMP`, however, the number of threads cannot be dynamically adjusted during runtime for most workloads. Thus, if the power cap or workload behavior vary significantly during runtime, it is highly likely that the system operating point will not lie on the power-delay Pareto efficient curve (see Figure 4).

	DVFS + Thread Packing		DVFS 1-Thread Fixed		DVFS 2-Threads Fixed	
blackscholes	243 sec	125 W	NPE	NPE	NPE	NPE
bodytrack	99 sec	124 W	NPE	NPE	NPE	NPE
canneal	119 sec	116 W	NPE	NPE	97 sec	123 W
dedup	25 sec	118 W	NPE	NPE	22 sec	121 W
facesim	371 sec	123 W	NPE	NPE	421 sec	122 W
ferret	218 sec	126 W	510 sec	120 W	NPE	NPE
fluidanimate	275 sec	123 W	NPE	NPE	NPE	NPE
freqmine	301 sec	127 W	NPE	NPE	461 sec	123 W
raytrace	127 sec	121 W	404 sec	111 W	181 sec	120 W
streamcluster	312 sec	123 W	504 sec	121 W	282 sec	124 W
swaptions	171 sec	124 W	508 sec	117 W	NPE	NPE
vips	80 sec	123 W	NPE	NPE	104 sec	121 W
x264	79 sec	121W	NPE	NPE	NPE	NPE

Table 2: A comparison of the average power and execution delay of our MLR classifier with and without thread packing for a 130W power cap. NPE stands for Not Pareto Efficient, meaning that one of the other methods has lower delay and lower average power.

In contrast to pure-DVFS with thread reduction, thread packing assignments can always be smoothly adjusted during runtime to operate along the Pareto frontier without special interfaces or workload modifications. A comparison between thread packing and DVFS with fixed thread reduction is summarized in Table 2. The word *fixed* designates that the number of threads is held constant for the entire workload execution to reflect the lack of flexibility. For all methods, an MLR classifier is generated and each benchmark is executed to completion under a static 130W power budget. This low-range power budget is in the range of power values that is only attainable via thread packing or thread reduction. The resulting power and delay values show that only thread packing maintains Pareto efficiency for all workloads. When performing DVFS with fixed thread reduction, for most workloads thread packing achieves lower power and delay. The power and delay values for these workloads are marked NPE, or Not Pareto Efficient.

For workloads in which 1-thread fixed or 2-thread fixed is Pareto efficient, the advantage in power or delay is not proportional to the advantage offered by thread packing. Therefore, thread packing is able to achieve lower energy consumption for each workload, as illustrated in Figure 11. The improvement is especially stark in comparison to the 1-thread case. Thread packing is capable of matching the lower bound on the power cap associated with the 1-thread case, but achieves an average of 51.6% reduction in energy. When compared to the 2-thread case, thread packing is able to achieve a better power range, and an average of 15.6% reduction in energy.

It is interesting to look at the performance of our classifier if the number of threads is fixed to 1. In this case our classifier reduces to a pure DVFS classifier. In a fourth experiment, we compare the ability of our L_1-regularized MLR classification methodology in selecting optimal V-F settings to state-of-the-art energy-aware DVFS techniques proposed by Dhiman et al. [8] and Isci et al. [14]. Both of these techniques utilize performance counter metrics and look-up tables to dynamically select the V-F settings that optimize energy efficiency during runtime. The approach in [8] uses a CPI-based metric, and [14] computes the ratio of memory-bus transactions to μ-ops. While both metrics are selected based on their strong correlations to energy efficiency, they are a small subset of the exhaustive set of ratios employed by the MLR approach. Because the MLR approach uses L_1-regularization to automatically select the ratios that are most relevant to classification, it is significantly better at predicting the optimal DVFS settings across the

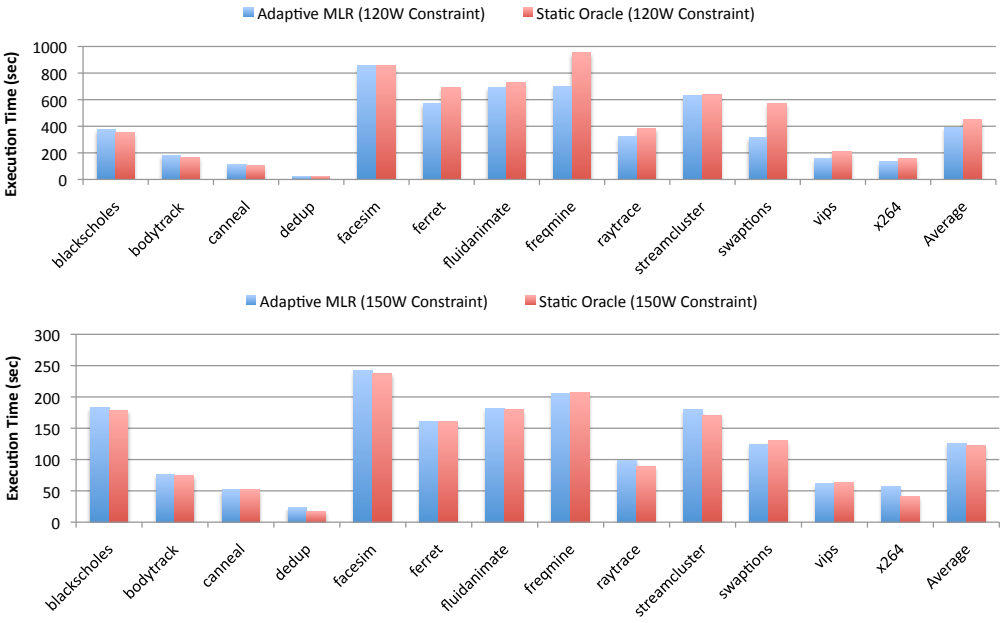

Figure 10: Execution delay comparison between Pack & Cap and a static oracle setting for each benchmark.

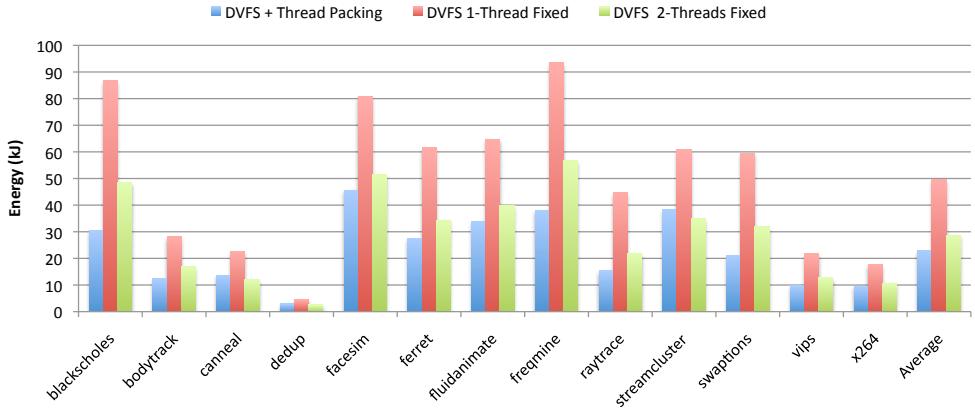

Figure 11: Comparison of energy consumption for each benchmark with and without thread-packing with 130W power budget.

entire range of workload behavior. Table 3 displays the accuracy of each technique in selecting the V-F setting that minimizes energy within a power constraint p_c for fixed size instruction intervals (100-billion μ-ops). As both of the previous approaches target single-threaded workloads, this comparison is made while executing single-thread configurations for each benchmark. The reported accuracies in Table 3 reflect the fraction of intervals in each benchmark for which the optimal setting matches the *true* optimal settings obtained from our offline data.

For the $p_c < 130W$ objective, there are more variations across benchmarks in terms of the optimal V-F settings in comparison to the $p_c < 140W$ objective. As a result, accuracy percentages for $p_c < 130W$ are significantly lower than $p_c < 140W$. As we see in Table 3, our technique (MLR, without thread packing) outperforms the other two techniques in selecting the optimal V-F settings, especially for lower power budgets.

5. RELATED WORK

DVFS is a popular method for power management due to the cubic dependency of dynamic power on frequency (frequency scales with voltage squared). To identify the best V-F setting, the majority of recent work on DVFS control rely on information gathered from performance counters during runtime without the need for special compiler support or modifications to the applications [18, 26]. Some software-based approaches require application-level or compiler-level support for power control [25, 4]. Most methods optimize for energy, energy-delay-product (EDP) or energy-delay-squared-product (ED^2P). Isci *et al.* in [14] derive phase categories based on a metric for the memory operation rate (mem ops/μ-op), and each category is a mapped to an optimal V-F setting. Similarly, Dhiman *et al.* propose an online learning model for single-core processors in [8]. In order to characterize workloads, they break down the cycles per instruction (CPI) metric into various components such as baseline CPI, miss events CPI, and stall CPI. This approach guarantees convergence to the optimum voltage-frequency (V-F) setting using online learning. These approaches focus on energy, EDP or ED^2P minimization without considering power caps.

Multi-core processors introduce new opportunities for power management as they enable larger degrees of freedom in job scheduling

	Isci et al. [14]		Dhiman et al. [8]		Proposed	
	$p_c < 130W$	$p_c < 140W$	$p_c < 130W$	$p_c < 140W$	$p_c < 130W$	$p_c < 140W$
blackscholes	97.4%	81.8%	100%	99.9%	100%	88.8%
bodytrack	85.0%	99.9%	6.8%	94.7%	96.0%	79.8%
canneal	99.9%	96.6%	96.2%	96.2%	100%	100%
dedup	23.7%	100%	2.5%	62.2%	100%	100%
facesim	96.9%	100%	99.9%	96.2%	96.7%	80.7%
ferret	51.0%	97.0%	36.8%	66.3%	97.7%	92.4%
fluidanimate	11.4%	100%	19.9%	100%	98.6%	79.2%
freqmine	5.9%	56.0%	1.0%	5.0%	100%	90.2%
raytrace	99.9%	88.3%	0%	97.0%	100%	92.3%
streamcluster	99.6%	99.6%	27.3%	27.3%	93.1%	92.4%
swaptions	0%	21.2%	100%	100%	87.2%	86.1%
vips	99.1%	32.3%	99.0%	99.4%	100%	100%
x264	0.5%	1.9%	1.5%	99.2%	100%	94.4%
Average	59.3%	75.0%	45.4%	79.6%	97.7%	90.4%

Table 3: Accuracy of DVFS techniques in selecting the optimal operating points for single-threaded execution.

and allocation. Rangan *et al.* propose a scalable DVFS scheme for multi-core systems that enables thread migration among homogeneous cores with heterogeneous power-performance capabilities [22]. Rather than changing the V-F settings on demand, they assign fixed V-F settings to different cores and migrate the applications to reach the desired level of performance within a given power budget. For applying DVFS under power constraints, Etinski *et al.* propose a job scheduling policy that optimizes performance for a given power budget [9]. Isci *et al.* evaluate global power management policies with objectives such as prioritization, power balancing and optimized throughput for various benchmark combinations and power budgets [13]. However, their approach does not provide dynamic adaptation to different workloads. Teodorescu *et al.* propose algorithms for power management through scheduling and DVFS under process variations [27].

Most multi-core processors support setting independent frequencies for the cores but a common voltage level is usually set to support the highest frequency. Independent voltages require extensive design investments in the power-delivery network and the off-chip power regulators. To increase the granularity of DVFS control, multiple clock domain design and voltage frequency island partitioning have been proposed [19, 12]. To reduce the overhead of runtime voltage conversion, Kim *et al.* explore designing on-chip regulators and perform core-level DVFS [15].

As the need for power capping and peak power management grows, some of the recently proposed techniques have explicitly focused on meeting power budgets or peak power constraints at runtime. Cebrian *et al.* propose a power balancing strategy that dynamically adapts the per-core power budgets depending on the workload characteristics [7]. However, for balanced workloads which have even power consumption among cores (e.g., highly parallel workloads such as `blackscholes` in PARSEC), this strategy would not perform well as it relies on borrowing power budgets from cores that consume lower power than the others. Sartori *et al.* propose a peak power management technique for multi-core systems by choosing the power state for each core that meets the power constraints [24]. The power capping strategy proposed by Gandhi *et al.* meets the power budget by inserting idle cycles during execution [11]. This approach targets controlling the average power consumption, and does not provide peak power guarantees. A number of approaches meet the power budgets through hardware reconfiguration. Meng *et al.* propose a power management strategy through dynamic reconfiguration of cores by cache resizing [21]. Kontorinis *et al.* propose a table-driven adaptive core reconfiguration technique that configures core resources such as floating point units and load-store queues to meet peak power constraints [16].

Current processor and system vendors have begun to provide peak power management features in commercial products. AMD has introduced *PowerCap Manager* for 45 nm Opteron processors [23]. For data center power management, HP and Intel jointly offer a power capping technique which adjusts power caps according to busy/idle states of the nodes [1]. This technique utilizes the DVFS states and the throttling (idle cycle insertion) capabilities at the chip-level. Besides sleep modes, power nap modes, in which the system can enter and exit from low-power modes in milliseconds, have been also proposed to cope with the demand variation patterns in data centers [20].

This paper brings the following important innovations over the state-of-the-art: (1) Our technique targets caps on peak server power, while most prior techniques (such as throttling and DVFS) focus on maintaining an average power consumption value; (2) we do not require any modifications to the hardware beyond fundamental DVFS capabilities, which are already included in most processors; (3) we do not require power metering during runtime which saves large investment in power metering infrastructure; (4) we use thread packing in addition to DVFS, which increases the power range and improves performance compared to applying DVFS alone; (5) we split our technique into offline and runtime phases, where the offline phase uses machine learning techniques to build models from a large volume of characterization data, and the runtime phase uses table look-ups to efficiently select the best settings; and (6) we conclusively demonstrate that our method successfully optimizes performance and energy efficiency under power caps on a real quad-core based system using realistic multi-threaded applications. Our approach is also applicable to single-threaded applications running on multi-core based systems.

6. CONCLUSIONS

Power capping is an increasingly prevalent feature in modern servers as it enables administrators to control energy costs. In this work we propose **Pack & Cap**, a novel technique for maximizing the performance of multithreaded workloads on a multi-core processor within an arbitrary power cap. We introduce thread packing as a control knob that can be used in conjunction with DVFS to manage the power-performance tradeoff. We demonstrate that thread packing expands the range of feasible power caps, and it enables fine-grained dynamic control of power consumption. In devising a MLR classifier approach to identifying optimal operating points, we demonstrate that it is possible to automatically select Pareto-optimal DVFS and thread packing combinations during runtime. Using a large body of characterization data gathered from the PARSEC benchmark suite, we train sophisticated classifier mod-

els that encapsulate the workload dependence of the power-delay Pareto frontier. By performing model learning offline and exposing the models via lookup tables, we reduce the runtime overhead of our control scheme to a low-cost probability calculation. We implemented **Pack & Cap** on a real quad-core based system with a wide range multi-threaded workloads. Our experiments demonstrate that our method is capable of adhering to a power cap 82% of the time while maximizing performance, even in the absence of a power measuring device. Thread packing increases the range of feasible power constraints by an average of 21% when compared to DVFS alone and reduces workload energy consumption by an average of 51.6% compared to existing control techniques that achieve the same power range.

7. REFERENCES

[1] Hp-intel dynamic power capping. http://www.hpintelco.net/pdf/solutions/SB_HP_Intel_Dynamic_Power_Capping.pdf, 2009.

[2] HP Power Capping and HP Dynamic Power Capping for ProLiant servers, Technology Brief, 2nd Edition. http://h20000.www2.hp.com/bc/docs/support/SupportManual/c01549455/c01549455.pdf, 2011.

[3] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, first edition, 2004.

[4] 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*, pages 168–175, 2002.

[5] L. A. Barroso and U. Holzle. *The Datacenter as a Computer*. Morgan and Claypool Publishers, 2009.

[6] 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.

[7] J. M. Cebrian, J. L. Aragon, and S. Kaxiras. Power token balancing: Adapting cmps to power constraints for parallel multithreaded workloads. In *Proceedings of International Parallel and Distributed Processing Symposium*, pages 431–442, 2011.

[8] 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*, pages 207–212, 2007.

[9] 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*, pages 257–267, 2010.

[10] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the International Symposium on Computer Architecture*, pages 13–23, 2007.

[11] A. Gandhi, R. Das, J. O. Kephart, M. Harchol-balter, and C. Lefurgy. Power capping via forced idleness. In *Proceedings of Workshop on Energy-Efficient Design*, 2009.

[12] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of International Symposium on Low PowerElectronics and Design*, pages 38–43, 2007.

[13] 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 International Symposium on Microarchitecture*, pages 347–358, 2006.

[14] 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 International Symposium on Microarchitecture*, pages 359–370, 2006.

[15] W. Kim, M. Gupta, G.-Y. 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*, pages 123 – 134, 2008.

[16] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar. Reducing peak power with a table-driven adaptive processor core. In *In Proceedings of the International Symposium on Microarchitecture*, pages 189–200, 2009.

[17] B. Lee and D. Brooks. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 185–194, 2006.

[18] J. Li and J. Martinez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *International Symposium on High-Performance Computer Architecture*, pages 77 – 87, 2006.

[19] 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*, pages 14–27, 2003.

[20] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: eliminating server idle power. In *Proceeding of the International conference on Architectural support for programming languages and operating systems*, pages 205–216, 2009.

[21] K. Meng, R. Joseph, and R. P. Dick. Multi-optimization power management for chip multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 177–186, 2008.

[22] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: fine-grained power management for multi-core systems. In *International Symposium on Computer Architecture*, pages 302–313, 2009.

[23] T. Samson. AMD brings power capping to new 45nm Opteron line. http://www.infoworld.com/d/green-it/amd-brings-power-capping-new-45nm-opteron-line-906, 2009.

[24] J. Sartori and R. Kumar. Distributed peak power management for many-core architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 1556–1559, 2009.

[25] D. Shin, J. Kim, and S. Lee. Low-energy intra-task voltage scheduling using static timing analysis. In *Proceedings of the Design Automation Conference*, pages 438–443, 2001.

[26] K. Singh, M. Bhadauria, and S. A. McKee. Real time power estimation and thread scheduling via performance counters. *SIGARCH Computer Architecture News*, 37:46–55, July 2009.

[27] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *International Symposium on High-Performance Computer Architecture*, pages 363–374, 2008.