Big Data for LITTLE Cores

Combining Control and Learning for Energy Efficiency

Anonymous Submission

Abstract

Streaming sensor processing forms a foundational workload for embedded and mobile devices. Control theoretic resource managers have demonstrated ability to meet streaming applications' performance requirements with minimal energy. Their designs, are brittle, however; they require a priori *models mapping applications' performance and energy to system resource allocation. Thus, a controller designed for one application and system will have to be redesigned and reimplemented to work in another deployment. We would like the benefits of control theoretic design* – i.e.,*formal, bounded convergence to required performance in dynamic environments – without* a priori *models. We therefore propose CALOREE, a generalized control theoretic resource manager that requires no* a priori *model of controlled applications. CALOREE combines a generalized control system (GCS) that runs on a local device with a hierarchical Bayesian model (HBM) that runs on a remote server. The HBM aggregates data from multiple applications and devices to produce highly accurate models that are sent to the generalized control system, customizing control to the particular application and device. We extend standard control theoretic formal analysis to show that CALOREE provides probabilistic convergence guarantees despite having no prior model of the controlled application.*

We implement CALOREE using ARM big.LITTLE systems to run streaming applications and an x86 server to implement the HBM. We test in both single-app and multi-app environments (where new applications enter the system and compete for resources). We evaluate CALOREE's ability to both provide reliable performance and low energy and compare to state-of-the-art learning and control techniques. We find that CALOREE's unique combination of control and learning consistently provides the most reliable performance, with a worst case error (in the multi-app case) between desired and delivered performance of only 12% compared to 70-80% for prior approaches. Additionally, CALOREE consistently provides the lowest energy with average savings anywhere from 8-47%.

1. Introduction

Streaming sensor processing forms an important workload for embedded and mobile devices. Supporting these applications on these platforms requires delivering reliable performance – to keep up with the sensor data – and minimizing energy.

Control theoretic resource managers have proven effective at delivering streaming applications reliable performance with minimal energy; *e.g.*, in mobile multimedia [**grace**, 25, 44] and webservers []. Control systems are not only effective in practice, they provide formal guarantees that the desired performance will be achieved [10, 14]. These guarantees hold true even in the face of unpredictable system dynamics, like changing application workload or changing resource availability.

While control theory is a general technique, implementations are problem-specific because traditional control design directly incorporates application- and system-specifc models. For example, a controller that manages video encoding on a phone using processor speed will be of no use to a wireless receiver on an embedded system. This brittleness of control design is further exacerbated by growing application diversity and hardware complexity. For example, heterogeneous processors expose a huge range of performance and energy tradeoffs, which vary from application to application. Several research projects create control theoretic libraries that users customize for specific applications and sysetms by providing models [18, 49]. While these approaches are a step in the direction of generality, they still require significant user input and prior knowledge of the application to be controlled. *This paper's goal is to provide the benefits of control theoretic techniques even for applications we have not encountered before and with minimal input from users.*

There are several challenges that must be addressed to meet this goal:

- *System complexity:* Heterogeneous processor designs expose multiple resources that interact in complex ways, often leading to non-convex optimization problems.
- *Overhead:* We have limited resources to devote to exploring these tradeoff spaces – certainly, we cannot spend more energy learning the tradeoff spaces than we would save by knowing them.
- *Guarantees:* We would like to maintain the ability to formally reason about the controlled system's dynamic behavior despite working without *a priori* knowledge.

To meet these challenges, we propose CALOREE[1], a unique combination of control theory and machine learning. CALOREE dynamically manages resources

[1] **C**ontrol **A**nd **L**earning for **O**ptimal **R**esource **E**nergy **E**fficiency

for streaming applications to ensure their performance needs are met while minimizing their energy consumption. CALOREE has two components. The first is a generalized control system (GCS) that runs on the embedded or mobile device and manages resoureces. The second component is a hierarchical Bayesian model (HBM) that runs on a remote server. When the GCS encounters a new application, it takes a small sample of the performance and energy in different resource configurations and sends those samples to the HBM. The HBM aggregates samples across devices and applications to produce an estimate of the performance and energy tradeoffs of each application and device. This estimate is stored in a performance hash table (PHT) that is sent to the GCS, which uses it to control its local application. Additionally, the HBM sends the GCS each application's estimated variance so that the controller tune its behavior not just to the model, but also to the potential range in the model's output.

CALOREE addresses the three challenges listed above. The HBM is well-suited to learning non-convex optimization spaces that arise in systems with many configurable resources [30], addressing the complexity challenge. The HBM is, however, computationally expenisve and requires samples of several different applications to produce accurate models. Therefore, we offload it to a remote server to address the overhead challenge. Finally, because the HBM communicates the model's variance, we are able to derive probabilistic control theoretical guarantees. These are not quite as strong as traditional control theoretic guarantees that assume the model is fixed and known, but they still provide some mechanisms for formal reasoning. While a traditional control system guarantees convergence, CALOREE allows the user to specify a lower bound on the probability that the system will converge to the desired performance. For example, a user can specify that the system will converge with at least 95% probability.

We implement CALOREE and test its ability to control streaming applications on heterogeneous ARM big.LITTLE devices, with the HBM implemented on an x86 server. We compare to state-of-the-art learning and control systems. We also test CALOREE in both *single-app* environments, where the streaming application runs by itself, and *multi-app* environments where other applications unpredictably enter the system and compete for resources. Overall we find that CALOREE achieves the:

- *Most reliable performance:*
  - In the *single-app* case, all methods achieve average performance close to the requirement. CALOREE's worst case performance is still within 4% of the target, while prior techinqiues achieve worst case error that is off by anywhere from 28% to 75%.
  - In the *multi-app* case, prior learning approaches average at least 10% performance error, prior control approaches average 5% error, and CALOREE achieves just 2.7% average error, almost half the next best competitor. Looking at worst case numbers, all prior approaches have a worst case of at least 70% error, while CALOREE's worst case performance is within 13% of the target.
- *Best energy savings:*
  - In the *single-app* case, CALOREE reduces average energy by 23-47% compared to prior learning approaches and by 24% compared to prior control approaches. For the most complicated applications, CALOREE provides energy savings of more the $2\times$ compared to the best prior approaches.
  - In the *multi-app* case, CALOREE reduces average energy by 31% compared to prior control approaches and by 8% compared to prior learning approaches. While, the savings compared to learning approaches may look small, in this case the learning approaches are saving energy by missing their performance goal.
- *Competitiveness with custom control:* We compare CALOREE to a control system with rigorous *a priori* per application models – including models of application interference in the multi-app case – something that would be impractical to deploy in the real world. We find that CALOREE's average performance error across all cases is only 2% worse than this ideal contoller and its average energy is only 7.5% higher. This result shows that CALOREE's considerable increase in generality comes with only minor reductions in performance and energy.

In summary, control theoretic approaches are well suited to manage resources in dynamic environments and machine learning techniques can produce accurate models of complex processors. *To the best of our knowledge, this is the first work to propose combining the two at runtime to control resource usage for a streaming application with no prior knowledge.* Describing the interfaces and design choices that make this combination work is the primary contribution of the paper. Additional contributions include formal analysis showing how to incorporate learned variance into the control theoretic guarantees of existing systems and the empirical evaluation that shows the combined control and learning system outperforms individual, state-of-the-art control or learning solutions.

2. Motivational Example

We present two simple examples to illustrate the complementary strengths and weaknesses of learning and control. We use mobile development boards featuring Samsung's Exynos 5 Octa with an ARM big.LITTLE architecture

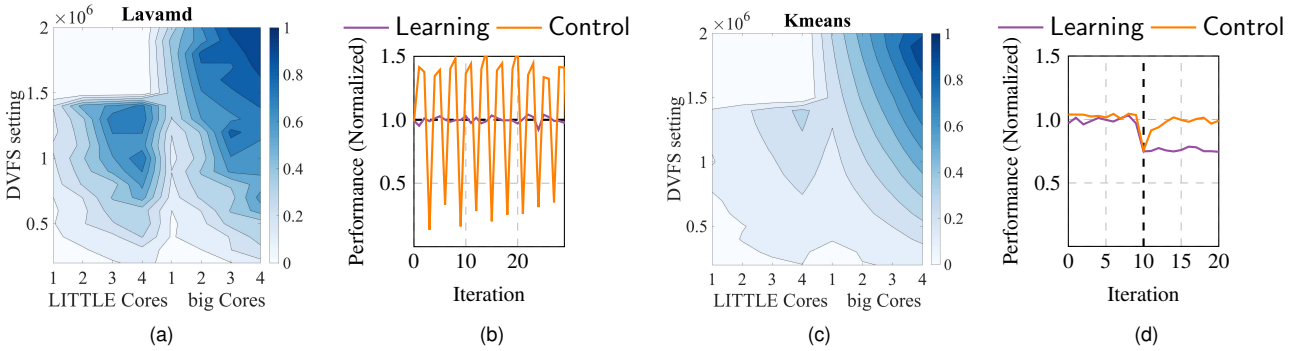

Figure 1: (a) Performance for `lavamd` as a function of configuration. (b) Managing `lavamd`'s performance: *Learning* navigates the complicated configuration space, but *control*'s simple model leads to oscillation. (c) Performance for `kmeans` as a function of configuration. (d) Managing `kmeans`' performance when another application starts: *Control* detects the change and adapts, but *learning* has no mechanism to handle these dynamics.

that has four energy-efficient LITTLE cores and four high-performance big cores. Each cluster can be set to different frequencies, leading to a large configuration space for assigning resources to multi-threaded applications.

Figures 1a and 1c show how performance varies as a function of both resource usage and application. The figures show cores on the x-axis and frequency on the y-axis, with darker colors representing higher performance. The presence of local minima and maxima mean that the function from resource usage to performance is non-convex. Therefore, simple gradient ascent/descent methods are not suitable to navigating these configuration spaces. Additionally, `lavamd` has a significantly more complicated configuration space than `kmeans`.

We consider prior *learning* and *control* approaches. LEO, a hierarchical Bayesian learner, estimates application performance as a function of its resource usage [30]. POET, a control system, adjusts resource usage to meet application performance requirements with minimal energy [18]. This section develops intuition about when one approach performs better than the other, motivating our proposal to combine the two.

2.1. *Learning* Complexity

Many machine learning approaches have been proposed to estimate application performance in a variety of scenarios [22–24, 34, 40, 42, 51]. Machine learning is well suited to building models of complicated systems like those shown in Figures 1a and 1c.

To demonstrate how well learning manages complexity, we consider meeting a performance requirement for `lavamd`, which has a complicated configuration space. We launch the application and use either *learning* or *control* to meet a performance requirement with minimal energy. The *learning* approach estimates all configurations' performance and power and then uses the lowest power configuration that delivers the required performance. The *control* approach has a generic model of performance/power frontiers (similar to `kmeans`) and it constantly measures performance and adjusts resource usage according to this generic model.

Figure 1b shows the results of controlling 30 iterations of `lavamd` to meet the performance requirement. The x-axis shows iteration number and the y-axis shows normalized performance. The learning approach achieves the goal, but the controller oscillates wildly around it, sometimes not achieving the goal and sometimes delivering performance that is too high (and wastes energy). The oscillations occur because the controller adjusts resources based on an incorrect (over-simplified) model of the configuration space. Hence, the *learner*'s ability to handle complex models is crucial for reliable performance in this example.

2.2. *Controlling* Dynamics

We now consider a dynamic environment. We begin with `kmeans` as the only application running on the system. Halfway through its execution, we launch a second app on a big core, dynamically altering resource availability.

Figure 1d shows the results of this experiment. The vertical dashed line represents when the second application begins. The figure clearly shows the benefits of a control system in this dynamic scenario. After a small dip in performance, the controller returns it back to the desired level. The learning system however, does not have any inherent mechanism to measure the change or adapt to the altered performance. While we could theoretically relearn the configuration space whenever the environment changes, doing so is impractical.

Control systems are a light-weight mechanism for managing such dynamics [14]. Control systems are resilient to scale change in the system performance or power. Many dynamic changes reduce all configurations' performance almost uniformly, changing the magnitude of performance

without altering the relative difference between configurations. For this reason, control systems have proven especially useful in webservers with fluctuating request rates [17, 27, 43] and multimedia applications with dynamically varying inputs [25, 28, 44].

3. Combining Learning and Control

CALOREE combines learning with control to tackle both complexity and dynamics. Figure **??** shows a detailed overview of CALOREE. A mobile system runs an application and some small number of measurements are taken and sent to a server. The server uses a hierarchical Bayesian model to combine these measurements with ones taken on other devices and with measurements of other applications. Using this large volume of data, the HBM produces an application-specific model of performance and power for all resource configurations. The model is sent to a lightweight control system (LCS) that manages the device, adjusting resource usage to meet application performance requirements with minimal energy. The learned models are stored a performance hash table (PHT), which is the interface between the remote HBM and the LCS that manages the device. The expensive process of constructing the PHT is done by the server. Once built, the PHT allows the LCS to apply the learned models in constant ($O(1)$) time. This section provides a brief overview of the relevant learning and control techniques used in CALOREE, and then describes the interface that combines them.

3.1. The Hierarchical Bayesian Model

Machine learning models are predictive in nature. Given some observations of a system, they create a model to predict future behavior in unobserved settings. CALOREE uses a hierarchical Bayesian model (HBM), based on LEO [30], to turn observations of applications' performance and power given some resource allocation into predictions of the performance and power of other, unobserved resource allocations. The HBM provides a statistically sound framework for learning across applications and devices.

The HBM is non-parametric in terms of resource configurations. Instead of modeling a configuration's performance/power as a function of the exact number of cores or clockspeed, it captures correlations between configurations. Hence, the HBM is well suited to learning complicated configuration spaces, like those in Figures 1a and 1c. The non-smoothness and application-specificity of these configuration spaces means that parametric models – based on clockspeed/cores – will not produce accurate predictions. In contrast, the HBM requires fewer assumptions about the relationship between configurations and performance/power and is thus more robust for this modeling problem [30].

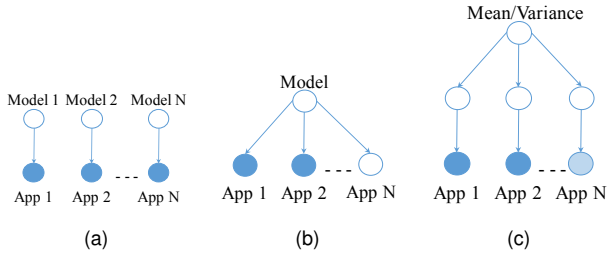

Figure 2: Comparison of online, offline, and hierarchical Bayesian models. Arrows represent dependences, circles are random variables, white circles are hidden and must be learned, solid circles are fully observed data, and shaded circles are partially observed.

Figures 2a–2c compare CALOREE to *online* and *offline* learning models. The online model only uses observations of the current application and it must be parametric in the configuration space, since it has no other information. The online model is thus highly dependent on the model parameterization; *e.g.*, if the model is specified to be linear in frequency, but an application is memory-bound, the online model will over-allocate resources. The offline model only uses information from previously observed applications and lacks knowledge of the current application. This general model will capture trends – *e.g.*, when most applications should transition from LITTLE to big cores – but it will miss key inflection points for applications that deviate from the general trend.

The HBM is a combination of the *online* and *offline*) approaches, incorporating both (1) observations of the current application and (2) observations of other applications to learn correlations between different configurations. The HBM's correlation matrix captures these relations and scales with the number of configurations, making the model non-parametric. In practice, such a non-parametric model is much more flexible; *e.g.*, it can learn a linear relationship between frequency and performance for a compute-bound application and that there is no relationship for a memory-bound one. Unlike the pure online approach, in the HBM all applications' models are conditionally dependent on a hidden mean and covariance matrix. Rather than over-generalizing (like the offline model) or over-specifying (like the online model) the HBM implicitly uses a pool of similar applications to produce new models. Additionally, the HBM's accuracy increases as more applications are observed because more behaviors are represented in the pool of prior knowledge. Of course, the HBM's computational cost – which is linear in the number of applications – also increases with increasing applications, but this is why we offload the learning to a remote server.

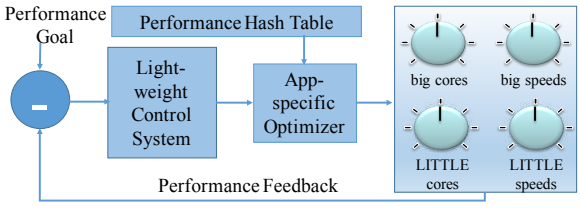

Figure 3: Light-weight control system (LCS)

3.2. The Lightweight Control System

Control theory provides a discipline for tuning system parameters to ensure operating goals are met in a dynamic environment. We use a controller to adjust system resource usage to see that the performance goals (corresponding to a quality-of-service or real-time constraint) are met over time. The difficulty is that classical control formulations integrate the application-dependent relationship between performance and the controlled resource directly in the control formulation. This makes the control models too specific to a particular class of applications. Thus, we face the problem of implementing a general control system that is applicable to a number of applications, and where the models relating resources to performance are not known ahead of time, but are provided by the HBM at runtime.

CALOREE addresses this problem using the classic computer science approach of adding a layer of indirection, as illustrated in Figure **??**. Instead of directly controlling resources using an application-dependent model, CALOREE controls *speedup* and a separate module uses the learned models to optimize energy while respecting this speedup constraint. Similar models have been used to build generalized controllers where users are responsible for supplying the models [18, 49]. Our goal is to eliminate this user burden and have a remote HBM supply the model.

3.2.1. Controlling Speedup We write a simple difference model relating speedup to performance:

$$perf(t) = m \cdot speedup(t-1) + \delta \tag{1}$$

where m is the *max speed* of the application, here defined as the speed when all resources are available. While m is application specific, it is easy to measure online, by simply allocating all resources. Such a configuration should not violate any performance constraints (although it is unlikely to be energy efficient) so it is safe to take this measurement without risk of violating performance constraints.

With this model, the control law is simply:

$$error(t) = goal - perf(t) \tag{2}$$

$$speedup(t) = speedup(t-1) - \frac{error(t)}{m} \tag{3}$$

which states that the speedup to apply at time t is a function of the previous speedup, the error at time t and the max speed m. This is a very simple *deadbeat* controller that provides all the standard control theoretic formal guarantees [14]. Using the above definition of max speed, most speedups will be less than one. In addition to making max speed easier to measure, this definition bounds the HBM's output, making for more robust learning.

3.2.2. Optimizing Speedup CALOREE must turn the speedup produced by Eqn. 3 into a resource allocation. The primary challenge here is that the HBM produces a non-linear function mapping the discrete system resource allocations into speedup and powerup, while Eqn. 3 is a continuous linear function. CALOREE bridges this divide by assigning time to resource allocations such that the average speedup over a control interval is that produced by Eqn. 3.

We call an assignment of time to resources a *schedule*. We call a combination of settings for each resource a *configuration*. For example, using two big cores at 1.5 GHzand putting all the little cores to sleep is one configuration. There are typically many schedules that meet a required speedup. To extend battery life, CALOREE finds a minimal energy schedule. Given a time interval T, a workload W to complete in that interval, and a set of C configurations, we formalize this problem as:

$$\arg\min_{\tau} \quad \sum_{c=0}^{C-1} \tau_c \cdot p_c \tag{4}$$

$$\text{such that} \quad \sum_{c=0}^{C-1} \tau_c \cdot s_c \cdot b = W \tag{5}$$

$$\sum_{c=0}^{C-1} \tau_c = T \tag{6}$$

$$0 \leq \tau_c \leq \tau, \qquad \forall c \in \{0,\ldots,C-1\} \tag{7}$$

where p_c and s_c are the estimated powerup and speedup of configuration c and τ_c is the amount of time to spend in configuration c. Eqn. 4 simply states that the objective is to minimize energy (power times time). Eqn. 5 states that the work must be done, while Eqn. 6 requires the work to be done on time. Eqn. 7 simply avoids negative time.

3.3. The Performance Hash Table

While most linear programming problems would be inefficient to solve repeatedly on a mobile device, the one in Eqns. 4–7 has a constant time, $O(1)$, solution. Kim et al. analyzed solutions to the problem of minimizing energy while meeting a performance constraint [20]. They observed that there must be an optimal solution with the following properties:

- At most two of τ_c are non-zero, meaning that at most two resource configurations will be used in any time interval. This property both drastically limits the search space and puts a limit on the overhead of switching configurations, since it is never profitable

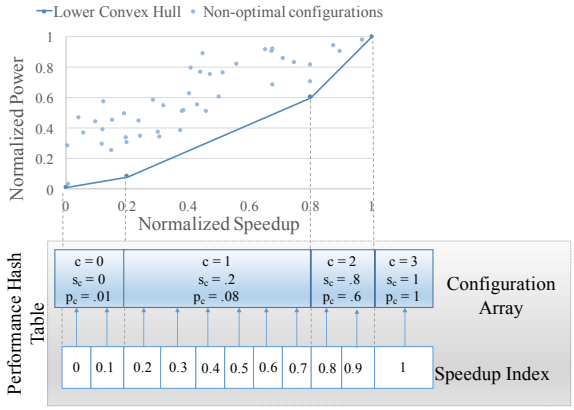

Figure 4: The Performance Hash Table and its relationship to the power/performance tradeoff space. The HBM encodes the configuration as the lower convex hull of points in the performance/power tradeoff space and stores those in a table indexed by speedup.

to switch more than twice in an interval.

- If one plots the configurations in the power and performance tradeoff space (so that performance is on the x-axis and power is on the y-axis) the two configurations with non-zero τ_c lie on the lower convex hull of that space.

We use these two facts to construct a constant time algorithm for finding the optimal solution to Eqns. 4–7 online. The intuition behind our solution is illustrated in the upper half of Figure 4. This figure shows a hypothetical example of the power and performance tradeoffs the HBM might learn for some application and device. Each point represents a configuration and each configuration is charted with normalized performance on the x-axis and normalized power on the y-axis. For any feasible performance requirement, there is a minimal energy schedule that uses no more than two of the configurations on the lower convex hull, as any points that lie above the lower convex hull require more power for equivalent performance [20]. Therefore, the HBM estimates the power and performance of all configurations, then finds the lower convex hull, and sends that to the LCS. This lower convex hull is the interface between the HBM and the LCS, and the key enabler of CALOREE's combination of learning and control.

The HBM stores the lower convex hull in a *performance hash table* (PHT). The PHT and its relationship to the lower convex hull is illustrated in Figure 4. It consists of two arrays, the first is an array of pointers into the second array, which stores the configurations on the lower convex hull sorted by speedup. Recall that speedups are computed relative to the maximum speed. We therefore know the largest speedup is 1, so we need only concern ourselves with speedups less than 1. The first table of pointers has a *resolution* indicating how many decimal points of precision it captures. The example in Figure 4 has a resolution of 0.1. Each pointer in the bottom table points to the configuration in the second array that has the largest speedup less than or equal to the index.

To use the table, the optimizer receives a speedup $s(t)$ from the controller. It needs to convert this into two configurations referred to as *hi* and *lo*. To find the *hi* configuration, the optimizer clamps the desired speedup to the largest index lower than $s(t)$ and then walks forward until it finds the first configuration with a speedup higher than $s(t)$. To find the *lo* configuration, the optimizer clamps the desired speedup to the smallest index higher than $s(t)$ and then walks backwards until it finds the configuration with the largest speedup less than $s(t)$.

Finally, the optimizer sets τ_{hi} and τ_{lo} by solving the following set of equations:

$$\tau = \tau_{hi} + \tau_{lo} \quad (8)$$

$$s(t) = s_{hi} \cdot \tau_{hi} + s_{lo} \cdot \tau_{lo} \quad (9)$$

In these equations, $s(t)$ is the speedup requested by the optimizer and s_c are speedups estimated by the learner.

By solving Eqns. 8 and 9, the optimizer has turned the controller's requested speedup into a resource allocation schedule using the models provided by the HBM. Provided that the resolution is large enough to get a good spread of configurations to indices, the optimizer will index the configuration one entry (on average) from where it needs to be. Thus, the entire optimization process runs in constant time – assuming that the learner is responsible for building the PHT once before passing it on to the optimizer. This efficiency comes at a cost of memory usage, as many of the entries in the speedup index table will point to redundant locations in the configuration array. This tradeoff is reasonable in practice as the code that runs on the mobile device must be fast or we risk wasting energy while trying to save energy. In practice, we recommend a table of size 100 which provides a sufficient resolution and is not too wasteful of space.

4. Experimental Setup

4.1. Platform and Benchmarks

We perform our experiments on four ODROID-XU3 devices which are small mobile devices running Ubuntu 14.04. The ODROID boards have Samsung Exynos 5 Octa processors based on the ARM big.LITTLE architecture. Each processor has 19 speed settings for its 4 big cores and 13 for 4 LITTLE cores. Each board has an on-board power meter updated at 1/4 msintervals. Each resource configuration (combination of big/LITTLE cores and clock-speeds) has a different performance and power, which is, itself, application-dependent.

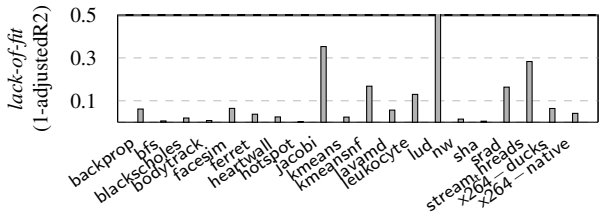

Figure 5: *Lack-of-fit* for performance vs clock-speed for a fixed number of cores. Lower value for lack-of-fit indicates a more compute-bound application whereas a higher value indicates a memory bound application.

We use 20 different benchmarks from different suites including PARSEC [3], Minebench [32], Rodinia [5], and STREAM [29]. Our benchmarks include compute-bound, memory-bound, and hybrid workloads as shown in Figure 5. That figure reports the *lack-of-fit* or the lack of correlation between frequency and performance. Applications with high lack-of-fit do not improve performance with increasing frequency, typical of memory bound applications, while applications with low lack-of-fit do see increasing performance with increasing clock-speed [31]. Applications with intermediate lack-of-fit tend to improve with increasing clock speed up to a point and then see no further improvement. Each application has been instrumented to report its performance as heartbeats/second an application-specific performance metric [18].

4.2. Evaluation Metrics

A performance target can be thought of as a quality of service expectation for the given application. We run our applications from 10% to 90% performance targets and observe how CALOREE performs under each constraint. We quantify the ability to meet performance targets using the standard metric *mean absolute percentage error* (MAPE). Suppose the application wants to emit n heartbeats while maintaining a speed of S_r, then MAPE is calculated as:

$$MAPE = 100\% \cdot \frac{1}{n} \sum_{i=1}^{n} max\left(\frac{S_r - s_m(i)}{S_r}, 0\right) \qquad (10)$$

where S_r is the performance target and $s_m(i)$ is the performance observed for ith heartbeat.

We run every application in every resource configuration and record performance and power for every heartbeat. By post-processing this data we can determine the optimal configuration for each heartbeat and each performance target. To produce an energy metric that we can compare across applications, we normalize energy as:

$$normalized\,energy = 100\%.(e_m/e_{optimal} - 1) \qquad (11)$$

where e_m is the measured energy and $e_{optimal}$ is the optimal energy produced by our oracle. We subtract 1, so that this metric shows the proportion of energy consumed over optimal.

4.3. Points of Comparison

We compare CALOREE's combination of HBM with control to baselines constructed with a combination of different learning models and a state-of-art controller based on the *Pace-to-idle (P2I)* heuristic [20]. Pace-to-idle is proven never to be worse than race-to-idle and often provides as much as 2× energy savings, but requires application-specific knowledge to implement. Pace-to-idle completes jobs in the most energy-efficient configuration and then transitions to a low-power sleep state until the next job is ready. As the energy-efficiency of a configuration is application-dependent, it is natural to combine pace-to-idle with machine learning approaches that can estimate the energy efficiency for different applications. In the next section, we will compare CALOREE's delivered performance and energy savings with:

1. *Online-P2I* – This strategy observes the application at small number of configurations then performs polynomial multivariate regression for the performance and power of unobserved configurations.
2. *Offline-P2I* – This is our zero-sample policy, used when we do not observe any values for the current application. This method takes the mean over all known applications and uses that to estimate power and performance for new applications. This strategy only uses prior information and does not update the model based on runtime observations.
3. *LEO-P2I* – We use the *LEO* learning model in combination with *Pace-to-idle (P2I)* heuristic.
4. *POET* – A state-of-the-art, open source control system designed to meet application performance with minimal energy [18]. POET requires users to specify a model of resource performance and power consumption. We use POET with the model produced by the *Offline* learner.

5. Experimental Evaluation

We evaluate CALOREE's ability to deliver requested performance with near-optimal energy. We first examine a single-application scenario, where each application is run by itself. We then consider a multi-application scenario where each application is run and then, halfway through, a second random application is launched, testing the ability to deliver performance in a changing environment. We then examine other dynamic scenarios where application inputs or behavior changes during application execution.

5.1. Performance and Energy for Single App

To test the ability to deliver performance and minimize energy, we set a range of targets from 10-90% of the maximum achievable performance on our system. The extreme targets are generally easy to hit. The most interesting performance targets are around 30%-70%, where there are

not obvious choices for configuration settings. These intermediate targets require a blend of big and LITTLE cores and getting that blend to both meet the performance and reduce energy is dependent on accurate models of performance and power.

The results for these single-application tests are shown in Figures 6 and 7. The benchmarks are shown on the x-axis; the y-axes show MAPE and the normalized energy, respectively. We find that CALOREE has lower MAPE and energy consumption than the baseline algorithms. Across all applications and targets, the online model produces an average error of 5.4%, the offline: 4.5%, LEO: 4.0%, POET: 4.7%, and CALOREE: 2.0%. These results show that CALOREE reduces error by a factor of two compared to prior approaches.

The energy consumption results are even more impressive. The online model requires an average of 52% more energy than optimal, offline: 25%, LEO: 31%, POET: 26%, and CALOREE: 7%. Thus the combination of learning and control provides a dramatic reduction in energy consumption compared to even state-of-the-art learning or control approaches. The reason is that the combination is complementary: LEO produces accurate models while control both can correct small errors in those models and adapt to dynamic changes in behavior.

5.2. Performance and Energy for Multiple Apps

In this experiment, we test the ability to deliver performance in an environment where applications compete for resources. We launch each of our benchmarks with a performance target (using the same targets as the prior study). Halfway through execution, we start another application randomly drawn from our benchmark set. We bind this application to one big core. Launching this second application clearly changes performance and power consumption. Delivering performance to the original application in this dynamic scenario tests the ability to react to environmental changes.

The results for the multi-application tests are shown in Figures 8 and 9. The benchmarks are shown on the x-axis; the y-axes show MAPE and the normalized energy, respectively. We note that the 90% target is generally not reachable in this scenario as it would require the controlled application to have exclusive use of all big cores.

CALOREE again has lower MAPE than the baseline algorithms. Across all applications and targets, the online model produces an average error of 38.5%, the offline: 15.5%, LEO: 19.7%, POET: 21%, and CALOREE: 18%. CALOREE is also more energy efficient than the baseline algorithms. Across all applications and targets, the average normalized energy for online model is 22%, the offline: 13.5%, LEO: 18%, POET: 15%, and CALOREE: 14%. But, the main benefits of CALOREE can be seen in terms of worst case scenarios where CALOREE is significantly better than the baselines. The online approach has a highest MAPE of 84%, for offline: 72%, for LEO: 84%, for POET: 37% and for CALOREE: 30%. CALOREE is also more energy efficient in the worst case scenario; the worst case energy consumption for online approach is 863%, for Offline: 307%, LEO: 747%, POET: 158.7% and CALOREE: 110%

POET and CALOREE both produce better outcomes in the dynamic scenario because they are specifically designed to handle system dynamics. CALOREE, however, does significantly better in the worst case than POET because the HBM produces more robust models. The energy consumption results are not as meaningful for this experiment, but they are included for completeness. Because there is another application running that is not under control, it consumes resources and energy that drags down energy efficiency for the control approaches. The learning approaches are not affected the same way. Because they do not re-allocate resources, the learning approaches actually save energy, but do so by missing the performance target.

To demonstrate CALOREE in this dynamic environment we look at the specific example of `bodytrack` with a 70% target. The time series data for bodytrack is shown in Figure 10. The figures show time (measured in frames) on the x-axis and performance/power on the y-axes. Performance is normalized to the target. There is a curve for each of LEO, POET, and CALOREE.

This small example clearly illustrates the benefits of CALOREE compared to prior approaches that use only control or learning. There are two key regions in the figures, the times before the second application starts (on the left of the vertical dashed line) and the times after (on the right). Before the second application starts (at the dashed line), both LEO and CALOREE do a good job of tracking the target and keeping energy low. In contrast, POET produces oscillating performance (and thus power) because it has a bad model of LITTLE core performance causing it to use the LITTLE cores too much and then too little. This oscillation also results in unnecessary power consumption. After the second application starts, POET and CALOREE recognize the performance has changed and adjust resource usage. CALOREE does a slightly better job of tracking the change, producing fewer oscillations. LEO cannot adjust to the change as it computes the optimal configuration once at the beginning of the application. Overall, LEO produces a MAPE of 13%, POET's is 9%, and CALOREE's is 4%. CALOREE produces better results than LEO because it reacts to the change; it produces better results than POET because it has captured the complex application-specific behavior on this system.

5.2.1. Phase Change In this experiment we demonstrate that CALOREE allows applications to operate well by

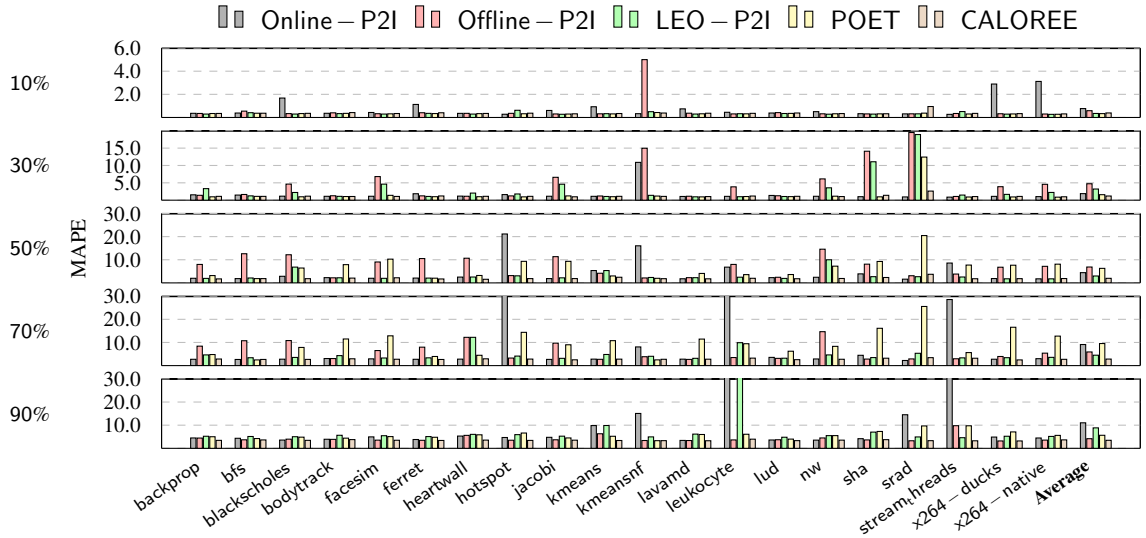

Figure 6: Comparison of application performance error for single application scenario.

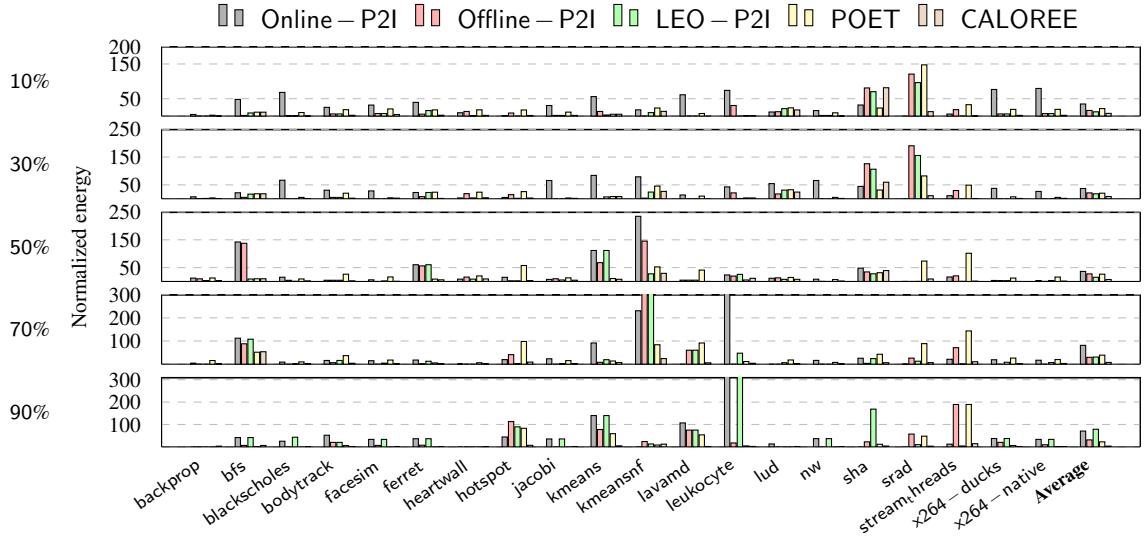

Figure 7: Comparison of application energy consumption for single application scenario.

changing resource allotment when the input varies with time. In Figure 11 we see `x264`, a video encoder application with 2 different phases, where the phase change occurs at the 180^{th} frame. The first scene is difficult and the second one becomes significantly easier. In the first phase, CALOREE and the baseline LEO-P2I seem to have lower MAPE as compared to POET. At the same time, these two algorithms seem to consume less energy than POET indicating that POET is operating at a configuration not on the Pareto frontier of power and performance. When we change phase, CALOREE and LEO-P2I both algorithms are still able to meet the performance target with far less fluctuations as compared to POET. But, CALOREE provides better energy savings as compared to LEO-P2I.

5.3. Sensitivity to the Measured Samples

We examine CALOREE's sensitivity to sample size. We vary the number of samples taken online and show how it affects the accuracy of the learned models. We note that this is simply the HBM's accuracy in producing the model used by the controller. This number is significant, because we do not want to take a large number of samples of new applications, if we can avoid it.

Figure 12 shows the results and compares CALOREE's accuracy to the online approach for learning both performance (top) and power (bottom). The figure shows sample size on the x-axis and accuracy on the y-axis. CALOREE's HBM initially performs as well as the Offline approach and as sample size increases, the accuracy uniformly improves and reaches greater than 90% with around 20 samples. On the other hand, the online approach needs at least 7 samples so that the design matrix

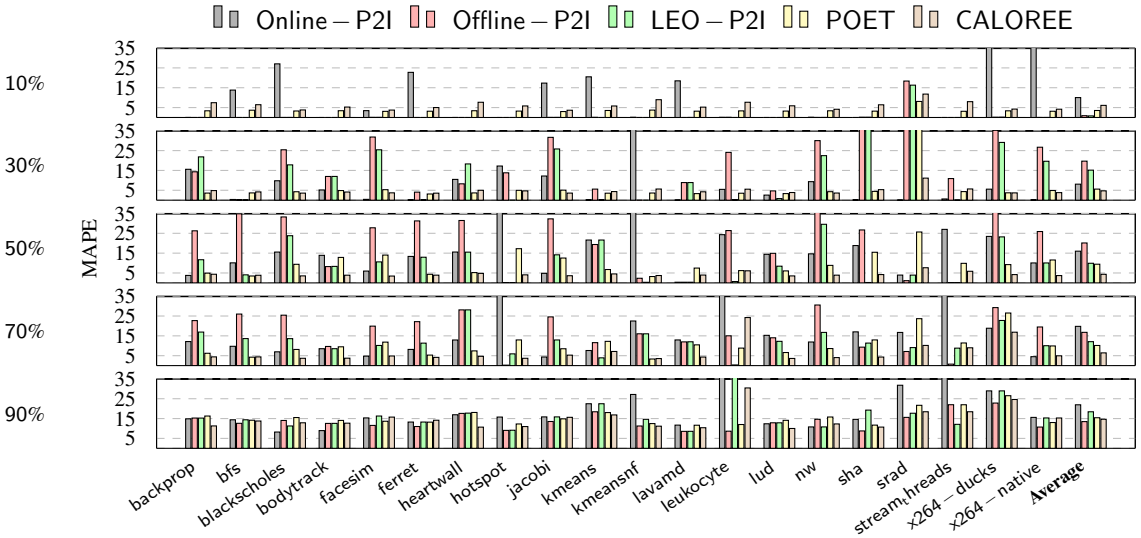

Figure 8: Comparison of application performance error for multiple application scenario.

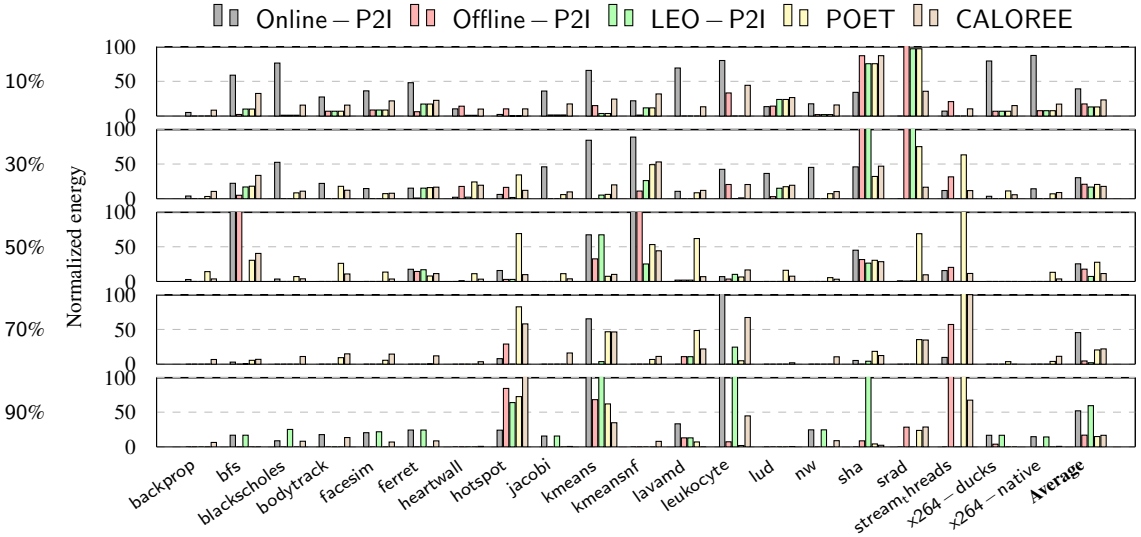

Figure 9: Comparison of application energy consumption for multiple application scenario.

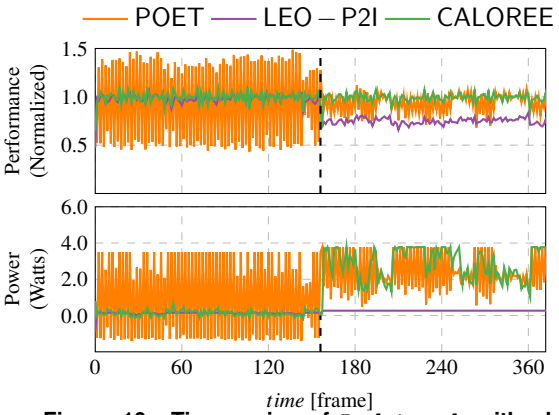

Figure 10: Time series of `Bodytrack` with phase change caused by multiple applications running.

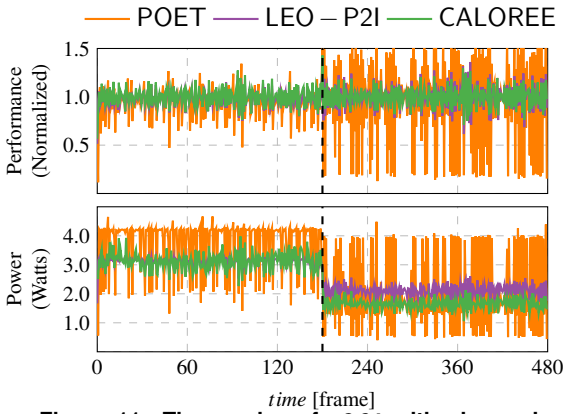

Figure 11: Time series of `X264` with phase change caused by different inputs.

for the polynomial regression has more samples than variables. As we get more samples the accuracy for the online model improves but still does not meet CALOREE's accuracy for the same number of samples.

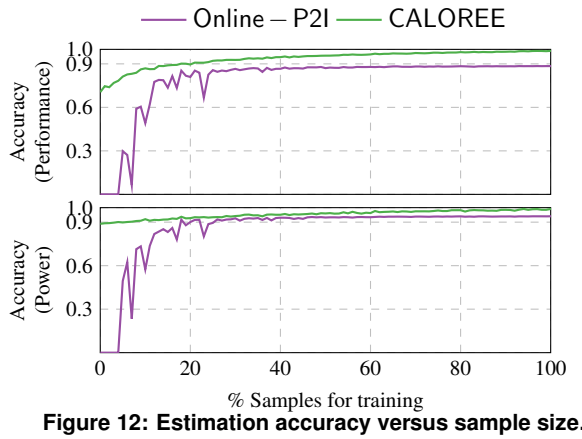

Figure 12: Estimation accuracy versus sample size.

5.4. Overhead

The main overhead of CALOREE is due to sampling where the applications need to run through a few configurations before CALOREE can reliably estimate the entire power and performance frontier. We argue that the sampling cost can be distributed across devices by asking each of them to contribute samples for estimation. Once the sampling phase is over, the HBM is quite fast and can generate an estimate as fast as 500 ms which is significantly smaller than the time required for sampling the applications.

The LCS requires only a few floating point operations to execute, plus the table lookups in the PHT. To evaluate the LCS' overhead, we time 1000 iteration of the controller. We find that it is under 2 microseconds, which is significantly faster than we can change any resource allocation on our system. We conclude that the LCS has negligible impact on performance and energy consumption of the controlled device.

6. Related Work

We discuss related work in managing resources to meet performance goals and reduce energy.

6.1. Machine Learning

There are a huge array of different learning techniques that are applicable to different problems. As discussed in Section 3.1 we break learning for resource management into 3 categories: offline, online, and hybrid approaches.

6.1.1. Offline Learning The offline approaches build models before deployment and then use those fixed models to allocate resources [2, 7, 22, 24, 48]. In these approaches, the model-building phase is generally very expensive, requiring both a large number of samples and substantial computation to turn those samples into accurate predictive models. Applying the model online, however, tends to be low overhead. The main drawback is that the models are not updated as the system runs which is a problem for adapting workloads. A good example of an offline approach applies learning to render web pages on mobile systems with low energy [51]. It builds an offline model mapping web page features into estimations of performance for different core types. When a new page is downloaded, the system quickly estimates the resource need to render the web page and uses the lowest energy resources that will still maintain user satisfaction. The mapping of web pages to resource use is very complicated and this approach deals with that complication. It does not, however, address system dynamics; *e.g.*, when other apps are running concurrently with the web browser.

6.1.2. Online Learning Online techniques use observations of the current application to tune system resource usage for that application [1, 23, 26, 34, 35, 42]. For example, Flicker is a configurable architecture and optimization framework that uses only online models to maximize performance under a power limitation [34]. Another example, ParallelismDial, uses online adaptation to tailor parallelism to application workload [42].

6.1.3. Hybrid Approaches Some approaches combine offline predictive models with online adaptation [8, 9, 39, 40, 45, 47, 50]. For example, Dubach et al. propose such a combo for optimizing the microarchitecture of a single core [9]. Such predictive models have also been employed at the operating systems level to manage system energy consumption [39, 40]. [47].

Many other approaches combine offline modeling with online updates [4, 15, 19]. Bitirgen et al use an artificial neural network to allocate resources to multiple applications in a multicore [4]. The neural network is trained offline and then adapted online using measured feedback. This approach optimizes performance but does not consider power or energy minimization.

6.2. Control

Almost all control solutions can be thought of as a combination of offline model building with online adaptation. Usually the model building involves substantial empirical measurement and modeling to build a model that is then used to synthesize a control system [6, 16, 18, 25, 28, 37, 38, 41, 46, 49]. The combination of offline learning and control works well over a narrow range of applications, as the offline models capture the general behavior of the entire class of application and require negligible online overhead. This focused approach is extremely effective for multimedia applications [12, 13, 21, 28, 44] and web-servers [17, 27, 43] because the workloads can be characterized ahead of time so that the models produce sound control.

Indeed, the need for good models is the central tension in developing control for computing systems. It is always possible to build a controller for a specific application and system by extensively modeling that pair. More general controllers which work with a range of applications

have addressed this issue with models in several ways. Some provide control libraries that encapsulate control functionality and require users to input a model [18, 38, 41, 49]. Others automatically synthesize both a model and a controller for either hardware [36] or software [10, 11]. JouleGuard combines learning for energy efficiency with control for managing application parameters [15]. In JouleGuard, a learner adapts the controller's coefficients to model certainty, but JouleGuard's learner does not produce a new model for the controller. Because JouleGuard's learner runs on the same device as the controlled application, it must be computationally efficient and thus it cannot identify correlations across applications or even different resource configurations. CALOREE is unique in that a remote server generates an application-specific model automatically. By offloading the learning task, we are able to (1) combine data from many applications and systems and (2) apply computationally expensive, but highly accurate learning techniques.

Carat also aggregates data across multiple devices [33]. Carat uses learning to generate a report for human users about how to configure their device to increase battery life. While both Carat and CALOREE learn across devices, they have very different goals. Carat's goal is to return very high-level information to human users; *e.g.*, you should update a driver to extend battery life. CALOREE returns lower-level models to another automated system that will apply those models to save energy.

7. Conclusion

This paper presents CALOREE, a combination of machine learning and control for managing resources in mobile systems. We have compared CALOREE to prior learning and control approaches and find that CALOREE delivers more reliable performance with lower energy both in a quiescent device, where the controlled application is the only one running, and in a more realistic scenario where other applications can interfere with the managed application. CALOREE's ability to manage both complexity and dynamics make it well-suited to managing modern mobile devices which must deliver reliable performance to users while dealing with energy limitations.

References

[1] J. Ansel, M. Pacula, Y. L. Wong, C. Chan, M. Olszewski, U.-M. O'Reilly, and S. Amarasinghe. "Siblingrivalry: online autotuning through local competitions". In: *CASES*. 2012.

[2] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe. "Language and compiler support for auto-tuning variable-accuracy algorithms". In: *CGO*. 2011.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. "The PARSEC Benchmark Suite: Characterization and Architectural Implications". In: *PACT*. 2008.

[4] R. Bitirgen, E. Ipek, and J. F. Martinez. "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach". In: *MICRO*. 2008.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. "Rodinia: A Benchmark Suite for Heterogeneous Computing". In: *IISWC*. 2009.

[6] J. Chen and L. K. John. "Predictive coordination of multiple on-chip resources for chip multiprocessors". In: *ICS*. 2011.

[7] J. Chen, L. K. John, and D. Kaseridis. "Modeling Program Resource Demand Using Inherent Program Characteristics". In: *SIGMETRICS Perform. Eval. Rev.* 39.1 (June 2011), pp. 1–12. ISSN: 0163-5999.

[8] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. "Pack & Cap: adaptive DVFS and thread packing under power caps". In: *MICRO*. 2011.

[9] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle. "A Predictive Model for Dynamic Microarchitectural Adaptivity Control". In: *MICRO*. 2010.

[10] A. Filieri, H. Hoffmann, and M. Maggio. "Automated design of self-adaptive software with control-theoretical formal guarantees". In: *ICSE*. 2014.

[11] A. Filieri, H. Hoffmann, and M. Maggio. "Automated multi-objective control for self-adaptive software design". In: *FSE*. 2015.

[12] J. Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile applications". In: *SOSP*. 1999.

[13] J. Flinn and M. Satyanarayanan. "Managing battery lifetime with energy-aware adaptation". In: *ACM Trans. Comp. Syst.* 22.2 (May 2004).

[14] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004. ISBN: 047126637X.

[15] H. Hoffmann. "JouleGuard: energy guarantees for approximate applications". In: *SOSP*. 2015.

[16] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal. "A Generalized Software Framework for Accurate and Efficient Managment of Performance Goals". In: *EMSOFT*. 2013.

[17] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. "Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control". In: *Computers, IEEE Transactions on* 56.4 (2007).

[18] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. "POET: A Portable Approach to Minimizing Energy Under Soft Real-time Constraints". In: *RTAS*. 2015.

[19] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach". In: *ISCA*. 2008.

[20] D. H. K. Kim, C. Imes, and H. Hoffmann. "Racing and Pacing to Idle: Theoretical and Empirical Analysis of Energy Optimization Heuristics". In: *CPSNA*. 2015.

[21] M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. "xTune: A Formal Methodology for Cross-layer Tuning of Mobile Embedded Systems". In: *ACM Trans. Embed. Comput. Syst.* 11.4 (Jan. 2013).

[22] B. Lee, J. Collins, H. Wang, and D. Brooks. "CPR: Composable performance regression for scalable multiprocessor models". In: *MICRO*. 2008.

[23] B. C. Lee and D. Brooks. "Efficiency Trends and Limits from Comprehensive Microarchitectural Adaptivity". In: *ASPLOS*. 2008.

[24] B. C. Lee and D. M. Brooks. "Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction". In: *ASPLOS*. 2006.

[25] B. Li and K. Nahrstedt. "A control-based middleware framework for quality-of-service adaptations". In: *IEEE Journal on Selected Areas in Communications* 17.9 (1999).

[26] J. Li and J. Martinez. "Dynamic power-performance adaptation of parallel computation on chip multiprocessors". In: *HPCA*. 2006.

[27] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son. “Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers”. In: *IEEE TPDS* 17.9 (2006), pp. 1014–1027.

[28] M. Maggio, H. Hoffmann, M. D. S. an d Anant Agarwal, and A. Leva. “Power optimization in embedded systems via feedback control of resource allocation”. In: *IEEE Transactions on Control Systems Technology (to appear)* ().

[29] J. D. McCalpin. “Memory Bandwidth and Machine Balance in Current High Performance Computers”. In: *IEEE TCCA Newsletter* (Dec. 1995), pp. 19–25.

[30] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann. “A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints”. In: *ASPLOS*. 2015.

[31] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. “Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling”. In: *ICS*. 2002.

[32] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. “MineBench: A Benchmark Suite for Data Mining Workloads”. In: *IISWC*. 2006.

[33] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma. “Carat: Collaborative Energy Diagnosis for Mobile Devices”. In: *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’13. Roma, Italy: ACM, 2013, 10:1–10:14. ISBN: 978-1-4503-2027-6. DOI: 10.1145/2517351.2517354. URL: http://doi.acm.org/10.1145/2517351.2517354.

[34] P. Petrica, A. M. Izraelevitz, D. H. Albonesi, and C. A. Shoemaker. “Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems”. In: *ISCA*. 2013.

[35] D. Ponomarev, G. Kucuk, and K. Ghose. “Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources”. In: *MICRO*. 2001.

[36] R. Pothukuchi, A. Ansari, P. Voulgaris, and J. Torrellas. “Using Multiple Input, Multiple Output Formal Control to Maximize Resource Efficiency in Architectures”. In: *ISCA*. 2016.

[37] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. “No "power" struggles: coordinated multi-level power management for the data center”. In: *ASPLOS*. 2008.

[38] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. “A resource allocation model for QoS management”. In: *RTSS*. 1997.

[39] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. “Energy Management in Mobile Devices with the Cinder Operating System”. In: *EuroSys*. 2011.

[40] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. “Koala: A Platform for OS-level Power Management”. In: *EuroSys*. 2009.

[41] M. Sojka, P. Písa, D. Faggioli, T. Cucinotta, F. Checconi, Z. Hanzálek, and G. Lipari. “Modular software architecture for flexible reservation mechanisms on heterogeneous resources”. In: *Journal of Systems Architecture* 57.4 (2011).

[42] S. Sridharan, G. Gupta, and G. S. Sohi. “Holistic Run-time Parallelism Management for Time and Energy Efficiency”. In: *ICS*. 2013.

[43] Q. Sun, G. Dai, and W. Pan. “LPV Model and Its Application in Web Server Performance Control”. In: *ICCSSE*. 2008.

[44] V. Vardhan, W. Yuan, A. F. H. III, S. V. Adve, R. Kravets, K. Nahrstedt, D. G. Sachs, and D. L. Jones. “GRACE-2: integrating fine-grained application adaptation with global adaptation for saving energy”. In: *IJES* 4.2 (2009).

[45] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker. “Scalable thread scheduling and global power management for heterogeneous many-core architectures”. In: *PACT*. 2010.

[46] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. “Formal online methods for voltage/frequency control in multiple clock domain microprocessors”. In: *ASPLOS*. 2004.

[47] W. Wu and B. C. Lee. “Inferred models for dynamic and sparse hardware-software spaces”. In: *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*. IEEE. 2012, pp. 413–424.

[48] J. J. Yi, D. J. Lilja, and D. M. Hawkins. “A Statistically Rigorous Approach for Improving Simulation Methodology”. In: *HPCA*. 2003.

[49] R. Zhang, C. Lu, T. Abdelzaher, and J. Stankovic. “ControlWare: A middleware architecture for Feedback Control of Software Performance”. In: *ICDCS*. 2002.

[50] X. Zhang, R. Zhong, S. Dwarkadas, and K. Shen. “A Flexible Framework for Throttling-Enabled Multicore Management (TEMM)”. In: *ICPP*. 2012.

[51] Y. Zhu and V. J. Reddi. “High-performance and energy-efficient mobile web browsing on big/little systems”. In: *HPCA*. 2013.