Big Data for LITTLE Cores

Combining Learning and Control for Mobile Energy Efficiency

Anonymous Submission

Abstract

Mobile systems must deliver performance to interactive applications and conserve resources to extend battery life. There are two central challenges to meeting these conflicting goals: (1) the complicated optimization spaces arising from hardware heterogeneity and (2) dynamic changes in resource availability and application behavior. Machine learning techniques handle complicated optimization spaces, but do not incorporate models of system dynamics; control theory provides formal guarantees of dynamic behavior, but cannot handle non-linear system models. In this paper, we propose a combination of learning and control techniques to meet mobile system's performance requirements on heterogeneous devices in unpredictable environments. Specifically, we propose CALOREE, which combines a hierarchical Bayesian model (HBM) with a lightweight control system (LCS). The HBM runs on a remote server, learning customized performance/power models by aggregating data across devices. The LCS runs on the mobile system and tunes resource usage to meet performance requirements efficiently. We propose a unique interface, the Performance Hash Table (PHT) which allows the LCS to apply the learned models in constant time. We implement CALOREE and test its ability to manage ARM big.LITTLE systems. Compared to existing learning and control methods, CALOREE delivers more reliable performance – only 2.1% error compared to 3.4-5.4% for learning and 4.6% for control – and lower energy – achieving 7% of optimal as compared to by 25-52% over learning and 26% over control. In a scenario where multiple applications compete for resources, these numbers improve: 7% error compared to 11-15% for learning and 9% for control and improvements of 2-20% and 3%, respectively, in energy efficiency.

1. Introduction

Mobile systems have clear requirements for user satisfaction: they must meet performance goals necessary for interacting with sensors and human users and must conserve energy to maximize battery life. To address these conflicting requirements, hardware platforms have become increasingly diverse. Many mobile processors support, for example, different core types with different performance/power tradeoffs, which can be operated at many different speeds. Meeting performance requirements is further complicated by the dynamic nature of computing systems: application resource demands can vary widely as a function of input or application phase and multiple applications may compete for resources.

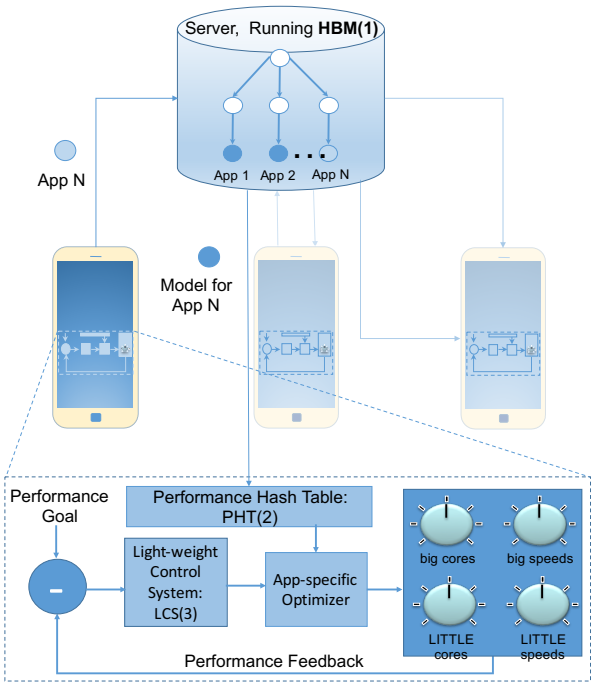

Figure 1: CALOREE overview.

Thus, two central challenges arise to meeting performance requirements with minimal energy on mobile systems: (1) complexity and (2) dynamics. Each challenge has been addressed individually. First, machine learning approaches can model the complicated, power/performance tradeoff spaces that arise on configurable, heterogeneous mobile systems [9, 4, 18, 38, 30, 32, 33]. Such approaches can learn complex models, identifying and avoiding local extrema that lead to inefficient resource usage. Second, control theoretic techniques ensure performance is met with minimal energy by tuning resource allocation using application feedback [44, 6, 15, 17, 47, 24, 42]. Control techniques provide a formal basis for reasoning about dynamics and can ensure performance requirements are met despite application, input, or workload fluctuations.

Learning and control techniques have complementary strengths and weaknesses. Learning approaches handle complexity – including non-linearities and non-convexity – but have no established mechanism for managing system dynamics. Control approaches handle dynamics, but rely on linear models that are increasingly insufficient to

capture the diversity of modern hardware.

We therefore propose combining learning and control techniques to manage both complexity and dynamics. We call our approach CALOREE[1] and it has three components as illustrated in Figure 1: (1) a hierarchical Bayesian model (HBM) that learns application-specific relationships between performance/power and resource usage, (2) a lightweight control system (LCS) that dynamically tunes resource usage to meet performance requirements with minimal energy, and (3) a performance hash table (PHT) that serves as the interface between learning and control.

The main challenge in combining a learning system with a control system is that the learner produces non-linear models of resources (*e.g.*,cores and clockspeeds) which are discrete, while the control system is based on continuous linear models. CALOREE bridges this gap by scheduling resources in time to meet a performance requirement with minimal energy.

The second challenge is that our learner (HBM) produces accurate models of power and performance, but like many sophisticated machine learning based models it is computationally expensive, so in CALOREE the HBM runs on a remote server. Moreover, as it is remote, it is also capable of aggregating data from multiple devices, which makes learner more powerful. The controller (LCS) runs on individual mobile devices and receives models from the HBM.

Thirdly, for great user satisfaction, we would like this system to be fast. Hence, we propose that these models are stored in the PHT, a data structure that allows the LCS to determine energy minimal resource schedules in constant time. The PHT is CALOREE's key enabler. While learning and control systems exist, their combination requires an appropriate interface. While only tested with CALOREE we believe the PHT is general enough to be used by different learning and control systems, or even to solve different optimization problems.

We test CALOREE on ARM big.LITTLE platforms with 20 different benchmarks to evaluate the HBM's ability to learn application specific models and the LCS's ability to deliver performance efficiently. We compare to published learning and control methods in a variety of settings. While many applications have inherent dynamics (*i.e.*, different processing phases), we explicitly test the ability to adapt to the unknown by running each application with other, random applications. In summary, this paper makes the following contributions:

- Proposing the combination of a hierarchical Bayesian learning with a lightweight control system to meet the twin challenges of addressing complexity and dynamics to deliver performance with minimal energy on mobile systems.
- An interface for combining discrete learned models of resource usage with continuous control models of resource dynamics.
- Evaluating the implementation and comparison to existing, independent learning and control techniques.
- We evaluate both the ability to meet performance requests and minimize energy consumption. We find that CALOREE: **[TODO: Double check all the numbers once we have the final values from the evaluation section.]**
  - **Delivers Reliable Performance:** We calculate the error between the desired and delivered performance. When running one application at a time, CALOREE achieves an average error of 2.1%, compared to 3.4-5.4% for existing learning methods and 4.6% for existing control approaches.
  - **Uses Lower Average Energy:** We compare energy consumption to the optimal found through exhaustive search. When running one application at a time, CALOREE achieves an average energy consumption of 7%, greater than optimal compared to 25-52% for existing learning methods and 26% for existing control approaches.
  - **Provides Better Worst Case Behavior:** In the single application case, CALOREE worst observed error across all applications and targets is 9% compared to 19-73% for prior learning methods and 25% for existing control methods. The worst observed energy for CALOREE is 1.82× greater than optimal, while it is 4-12× greater for learning and 2.9 × greater for control.
  - **Adapts to Dynamics:** We test CALOREE's ability to deal with changing environments by running each benchmark with a performance target and then randomly starting another application on one core. Even though the available resources fluctuate dynamically, CALOREE's worst performance error is 30%, while it is 71-84% for prior learning approaches and 35% for prior control approaches. Additional studies show CALOREE reacting to input changes and application behavioral changes.

2. Motivational Example

In this section we present two simple examples that illustrate the complementary strengths and weaknesses of learning and control. We run our examples on mobile development boards featuring Samsung's Exynos 5 Octa with an ARM big.LITTLE architecture that has four energy-efficient LITTLE cores and four high-performance

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

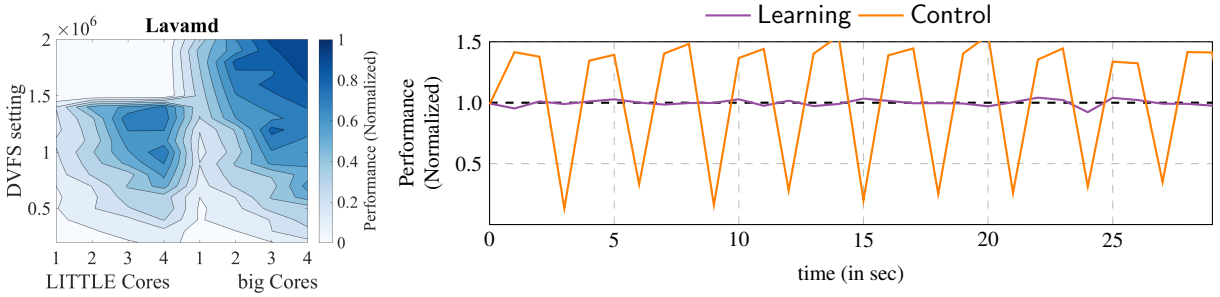

Figure 2: (a) Contour plot for normalized performance for `Lavamd` algorithm for different configurations(b) Time-line for running `Lavamd`. *Control* is running in isolation without an HBM based *learning* mechanism which leads to oscillations in the performance.

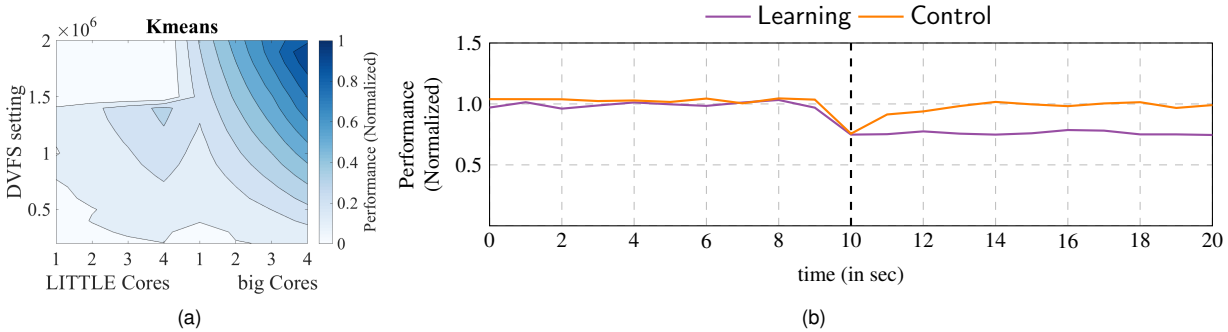

Figure 3: (a) Contour plot for normalized performance for `Kmeans` algorithm for different configurations(b) Time-line for running `Kmeans` alone until 10 seconds, when another application. *Learning* is running in isolation without an LCS based *control* mechanism which leads to a performance drop which does not recover by itself.

big cores. Each core cluster can be set to different clock speed, leading to a large configuration space for assigning resources to multi-threaded applications.

Each configuration (assignment of cores and clock-speeds) has different performance, and this performance will be application dependent. Figure 2a and Figure 3a show how performance varies as a function of both resource usage and application. The figures show cores on the x-axis and clockspeed on the y-axis, with performance shown as intensity – darker colors representing higher performance. The presence of local minima and maxima mean that simple gradient ascent/descent methods are not suitable to navigating these tradeoff spaces. Figure 2a and Figure 3a also show that the performance plot for the applications `kmeans` and `lavamd` against the clockspeed and cores is highly non-convex and contains at least 2 local minima corresponding to big and LITTLE cores. In addition, `lavamd` has other local minima and has a significantly more complicated tradeoff space than `kmeans`.

In this example we consider, a hierarchical Bayesian model based learner (LEO)that estimates application performance as a function of its resource usage [30]. Secondly, we consider POET, a control system designed to adjust resource usage to meet application performance requirements with minimal energy [17]. In this example we would refer to LEO as the *learning* algorithm and POET as the *control* mechanism and see develop an intuition on under what circumstances they perform better than each other which would lead into the key insight on why we their combination is right solution.

2.1. Complexity in *learning*

A number of machine learning approaches have been proposed to estimate application performance in a variety of scenarios [49, 23, 21, 40, 32, 22, 38]. Machine learning is well suited to building models of complicated systems like those shown in Figure 2a and Figure 3a .

To demonstrate how well suited learning is to managing complexity, we consider meeting a performance requirement for `lavamd`, the application with a complicated configuration space. We launch the application with a soft performance constraint and both *learning algorithm* (LEO) and the *control algorithm* (POET) adjust resource usage to meet that requirement with minimal resource usage. The *learning* algorithm works by estimating the

performance and power of all configurations and then using the lowest power configuration that meets the goal. On the other hand, The *control* algorithm does not have the knowledge of `lavamd`'s complicated profile but only has a knowledge of generic power/performance frontiers (similar to `kmeans` in this example) and it works by constantly measuring performance and adjusting resource usage to see that goals are met. While many controllers use linear models, POET uses a convex model and handles some non-linearities; however, it is sensitive to local maxima. Figure 2b 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 performance normalized to the goal. In this case, the learning approach achieves the performance goal and the controller oscillates wildly around it, sometimes not achieving the goal and sometimes delivering performance that is too high (and wastes energy). The problem stems from the fact that when the performance goal is not met the learner would adjust the resources to improve the performance based on an incorrect knowledge of the performance trade-off space. Hence, the knowledge as provide by the *learner* is extremely crucial for the controller to make the right decision.

2.2. *Control* in Dynamics

We now consider controlling performance in a dynamic environment using the `kmeans` application. In this scenario we start the application as the only application running on the system. Halfway through its execution we launch a second application on a single big core. This second application consumes about a quarter of the total resources. We again compare *learning* and *control* for this situation.

Figure 3b shows the results of this experiment, with time on the x-axis and performance on the y-axis, normalized to the target. The vertical dashed line shows when the second application begins. The figure clearly shows the benefits of a control system in this scenario. After a small dip in performance, the controller adjusts to return 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 tradeoff space every time the environment changes, this is obviously impractical.

Control systems are a great light-weight mechanisms for managing such dynamics. The control systems provide the necessary robustness to a system prone to such dynamics and fluctuations. These systems are resilient to scale change in the system performance or power and most of the dynamic changes (phase change or additional application running) reduces the performance at all configurations almost uniformly thus only affecting the scale of the performance but keeping the shape intact.

For this reason, control systems have been widely used to manage computing systems that have to deal with dynamic fluctuations. These approaches have been especially successful in webservers with fluctuating request rates [16, 26, 41] and multimedia applications which have to deal with multiple phases with different resource demands [27, 24, 42].

The primary contribution of this paper is to combine a learning technique that can handle complexity with a control system that can handle dynamic environments.

3. Combining Learning and Control

CALOREE combines learning with control to tackle the complexity and dynamics of modern mobile systems. Figure 1 shows a detailed overview of CALOREE. A mobile system runs an application and some small number of performance measurements are taken on a device and sent over 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. CALOREE uses an HBM based on LEO, a learning system build to estimate application performance and power for server systems [30]. LEO originally ran on the system to be optimized, but it is far too computationally expensive to be run on a mobile phone – even on the server for which it was designed, LEO has high overhead. In CALOREE, we configure LEO to run as a remote server, to build models of other systems based on data it is sent. Using this large volume of data, the HBM produces a model of performance and power for all combinations of resources on the device. On the mobile device, this model is stored in a novel data structure, called a *performance hash table* (PHT), which is the key interface between the remote HBM and the lightweight control system (LCS) that manages the device. The LCS adjusts resource usage to meet performance requirements for interactive mobile applications with minimal energy. The PHT is the key data structure allowing the LCS to apply the learned models in constant ($O(1)$) time. In this manner, the HBM handles the complexity of the device and the LCS handles the dynamics, with the PHT being the key interface linking the two. 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. Hierarchical Bayesian Learning

Machine learning models are predictive in nature. They take observations of some phenomena and create a model to predict future outcomes. In this work, we use a hierarchical Bayesian model (HBM) to turn observations of applications' performance and power into predictions about how other, unobserved resource allocations will alter that performance and power. We use a HBM be-

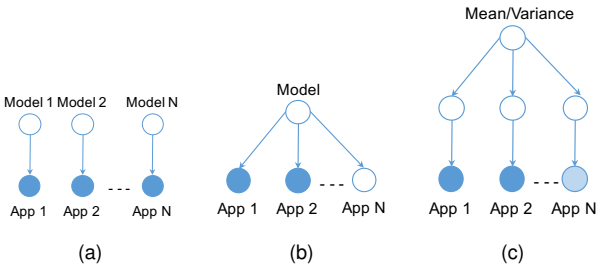

Figure 4: Comparison of online (a) offline (b) and hierarchical Bayesian models (c). The online model is concerned only with the current application (labeled N), the offline model combines all observations into one model, and the HBM builds per-application models, but makes them conditionally dependent on one another so that prior observations can be used to increase the accuracy of the model built for application N.

cause it provides a statistically sound framework for learning across applications and devices. The HBM is non-parametric in terms of resource configurations, hence well suited to learning the complicated trade-off spaces that arise from heterogeneity in modern mobile processors, like those illustrated in Figure 2a and Figure 3a. This non-smoothness is the reason why parametric models based on clockspeed/cores of these curves do not perform well and the hierarchical Bayesian model – which is non-parametric – is not affected by the shape of the curve. **[TODO: We still need one or two sentences to explain what this means and why it is good.]**

We compare CALOREE to an *online model*, which only uses only observations of the current application. For such model to be predictive, it must be parametric in terms of the configuration space (cores/clockspeed) since no other information is available. This online model requires a certain number of samples to estimate the model parameters with low error. Unfortunately, the model might still falter if the model parameterization is not correct; *e.g.*, if the model is misspecified and defined to be linear in cores, but the application actually hits an asymptote and no longer increases performance after some core count.

We also compare CALOREE to a purely *offline model* which only uses information from previously seen applications but lacks any knowledge of the current application to be optimized. This general model will capture trends; *e.g.*, when most applications should transition from LITTLE to big cores, but it might miss the key inflection point for some applications.

The HBM provides a good balance between the online and offline approaches. The key differences between these three approaches are illustrated in Figure 4. In this figure, circles represent random variables, directed edges show conditionally dependent relationships, and shading represents observability. A dark circle means we have seen all the data, a shaded circle means we have some partial observations, and the white circle means that the variable is unobservable. The models we are trying to learn are unobservable. A strictly online approach will handle each new application completely separately and ignore observations of previous applications. This approach never risks contaminating a model with unrelated observations, but it may take many observations of the current application to converge because it starts with no prior knowledge. The offline approach uses all observations from prior applications and will therefore converge very quickly; however, it is overly general and cannot learn features specific to a single application.

The HBM (illustrated in Figure 4c) is a superior combination of the above discussed approaches (*online* and *offline*). The model benefits from both (1) observations of the current application and (2) the previously observed applications. The model relies on accurately learning the correlations between different configurations based on the data from previously seen applications. Since, dimension of the model's parameter, the correlation matrix scales with the number of configurations the model is non-parametric. Unlike an *offline* scenario, here each application has its own model, allowing specificity, but these models are conditionally dependent on some underlying probability distribution with a hidden mean and co-variance matrix. In practice, the HBM will estimate a model for a new application using a small number of observations of that application (possibly from different devices) and combining those observations with many prior observations of other applications. This method generates a new model for the new application, which most agrees with the learned correlations and the new samples in hand. Rather than over-generalizing, the HBM implicitly uses similar applications to produce new models. In addition, the HBM's accuracy increases as more applications are observed because more different types of behavior are represented in the pool of prior knowledge. Of course, the computational complexity of learning – 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.

3.2. Controlling Computing Systems

Control theory provides a discipline for tuning parameters to ensure that a system meets some requirement in a dynamic environment. We use a controller to meet a performance goal (corresponding to a quality-of-service or real-time constraint) and adjust system resource usage to see that the goal is met. Our primary concern is making a control formulation that is general enough to be applied to a wide range of possible applications while receiving its model from the HBM.

We would like to use control and learning to solve

the problem of meeting an application's performance requirements while minimizing energy consumption. The difficulty is that classical control formulations integrate the models directly into the controller; *i.e.*, the application-dependent relationship between performance and the controlled resource is directly used in the control equations. For example, consider a control system designed to meet a performance goal by managing processor frequency. This controller's coefficients would directly encode the relationship between frequency and application performance. On mobile devices, we need to capture the relationship between all available resources and application performance, but this relationship is obviously application specific. 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.

We propose to address this problem using the classic computer science trick of adding a layer of indirection. This idea is illustrated in Figure 1. Instead of directly controlling resources using an application-dependent model, we will control *speedup* and pass that speedup value to a separate module that optimizes energy while respecting this speedup constraint using the learned models. Similar models have been used to build generalized controllers where users are responsible for supplying the models [17]. Our goal is to eliminate this user burden and have a remote HBM supply the model, which are not just more accurate but become more intelligent with time as it acquires more data.

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 [28, 10]. By measuring max speed online while the application runs, we can tune the control to a specific application. Using this definition of max speed, most speedups will be less than one. In addition to making max speed easier to measure, this has the nice property of bounding the learner's output, making for more robust learning [TODO: Nikita, what am I trying to say here? Or should we just put a forward reference because the next section benefits from a max speedup of 1] Of course, we still have the challenging problem of converting an abstract speedup into an actual resource allocation.

3.2.2. Optimizing Speedup We need to map the speedup produced by Eqn. 3 into a resource allocation. On our target system, an ARM big.LITTLE architecture, that specifically means mapping speedup into a number of big cores, a number of small cores, and a speed for both (on our system big and little cores can be clocked separately).

The primary challenge here is that the HBM produces a non-linear function mapping discrete resource allocations into speedup and powerup, while Eqn. 3 is a continuous linear function. We bridge 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. Not surprisingly, there are typically many schedules that meet a particular performance requirement. To extend battery life, we would like to find a minimal energy schedule. Given a time interval τ, a workload W to complete in that interval, and a set of C configuration, we formalize this problem as:

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

$$s.t. \quad \sum_{c=0}^{C-1} \tau_c \cdot s_c \cdot b = W \tag{5}$$

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

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

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.

3.2.3. A Fast Algorithm For Resource Scheduling 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 heuristic solutions to the problem of minimizing energy while meeting a performance constraint [19]. They observed that there must be an optimal solution with the following properties:

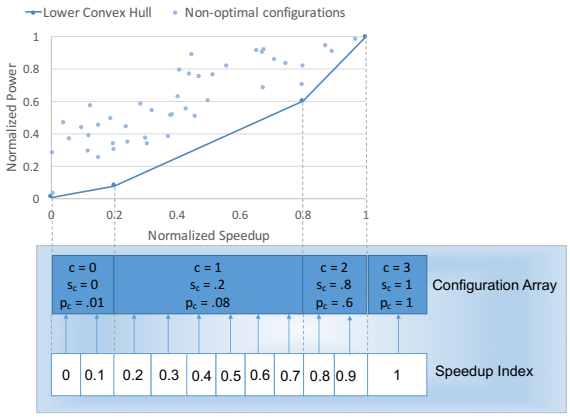

Figure 5: The Performance Hash Table and its relationship to the configuration space. The HBM encodes the configuration as the lower convex hull of points in the performance/power tradeoff space and stores those in a table indexed by speedup.

- 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 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 the power performance tradeoff 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 5. 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 [19]. Therefore, the HBM estimates the power and performance of all configurations, then finds the lower convex hull, and sends that to the LCS. The 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 5. 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 5 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 $s(t)$.

For example, consider the PHT in Figure 5 and an optimizer trying to meet a speedup $s(t) = .65$. To find *hi*, the optimizer indexes at .6 and walks up to find $c = 2$ with $s_c = .8$, setting $hi = 2$. To find *lo*, the optimizer indexes the table at .7 and walks backward to find $c = 1$ with $s_c = .2$, setting $lo = 1$.

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 schedule of resource allocations 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 always index the configuration array at most one entry 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 a small mobile devices with Ubuntu 14.04 running on them. The ODROID boards have Samsung Exynos 5 Octa processors based on the ARM big.LITTLE architecture. Each processor has 19 speed settings with all being available for its 4 big cores and 13 for 4 LITTLE cores. Each board has a power meter updated at 1 msintervals. Each resource configuration (combination of big/LITTLE cores and clockspeeds) has a different performance and power, which is, itself, application dependent.

We use 20 different benchmarks from five suites including PARSEC (`blackscholes`,`facesim`, `ferret`, `x264`) [3], Minebench (`Kmeans`, `non fuzzy kmeans (Kmeansnf)`), and Rodinia (`backprop`,`cfd`, `nn`, `lud`, `backprop`, `bfs`) [5]. We also use a partial differential equation solver (`jacobi`) and a memory intensive benchmark pair (`stream`) and (`stream_threads`) [29]. Our benchmarks vary from compute-bound applications to memory bound workloads and also workloads that are a combination of both as shown in Figure **??**. Eachapplication has been instrumented to report its performance as heartbeats/second an application-specific performance metric [17].

4.2. Evaluation metrics

CALOREE uses the model for power and performance and its controller actively uses this knowledge to meet the performance target. 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 evaluate energy savings by constructing an oracle. 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. **[TODO: Should we multiply this by 100, so that it is also a %? That seems like a good idea – might as well be consistent across the two metrics. We don't even have to change the data, just the axis labels.]**

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 [19]. Pace-to-idle is proven to be never 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 current application in a small number of configurations then performs polynomial multivariate regression and estimates the performance and power of the unobserved configurations. We use the *Online* learning model in combination with *Pace-to-idle(P2I)* heuristic.
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.We use the *Offline* learning model in combination with *Pace-to-idle(P2I)* heuristic.
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. 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 performance accuracy and energy in 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.

5.1. Performance and Energy for One App

To test the ability to deliver performance and minimize energy, we set a range of targets ranging 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: 4.4%, 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 30% more energy than optimal, offline: 16%, LEO: 20%, POET: 13%, and CALOREE: 5%. 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 Multile 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 clearly 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.

As in the prior study, CALOREE has lower MAPE than the baseline algorithms. Across all applications and targets, the online model produces an average error of 15%, the offline: 14%, LEO: 11%, POET: 9%, and CALOREE: 7%. In this case the average numbers skew CALOREE's benefit as no approach does very well with the 90% target and all but online do well at the 10% target. So, we also consider worst case behavior for all targets but the 90% one. The online approach has a highest MAPE of 80% for LUD at the 70% target. Offline's highest MAPE is 71% for STREAM at the 30% target. LEO's highest MAPE is 70%, also for STREAM at 30%. POET's highest MAPE of 35% also occurs for STREAM at 30%. CALOREE's highest MAPE of 24% occurs with LUD at the 70% target. 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 it starts with better models that are more robust to variation.

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 drag down energy efficiency for all approaches.

To demonstrate the benefits of CALOREE in this dynamic environment we look at the specific example of `bodytrack` at the 70% performance target. The time series data for bodytrack is shown in Figure **??**. 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 performacne 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 appli-

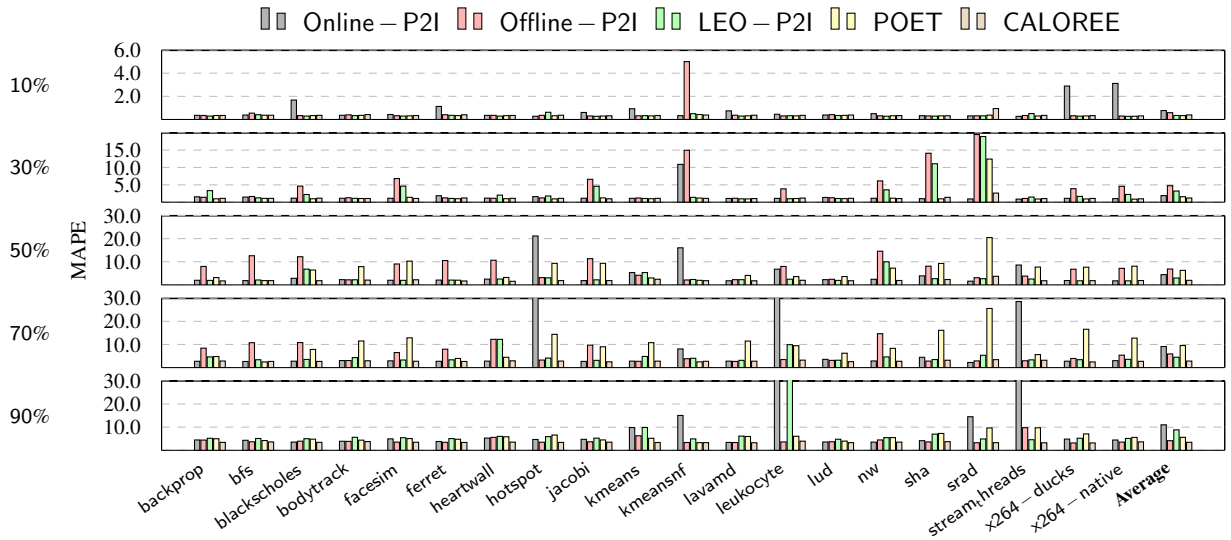

Figure 6: Single Applications MAPE

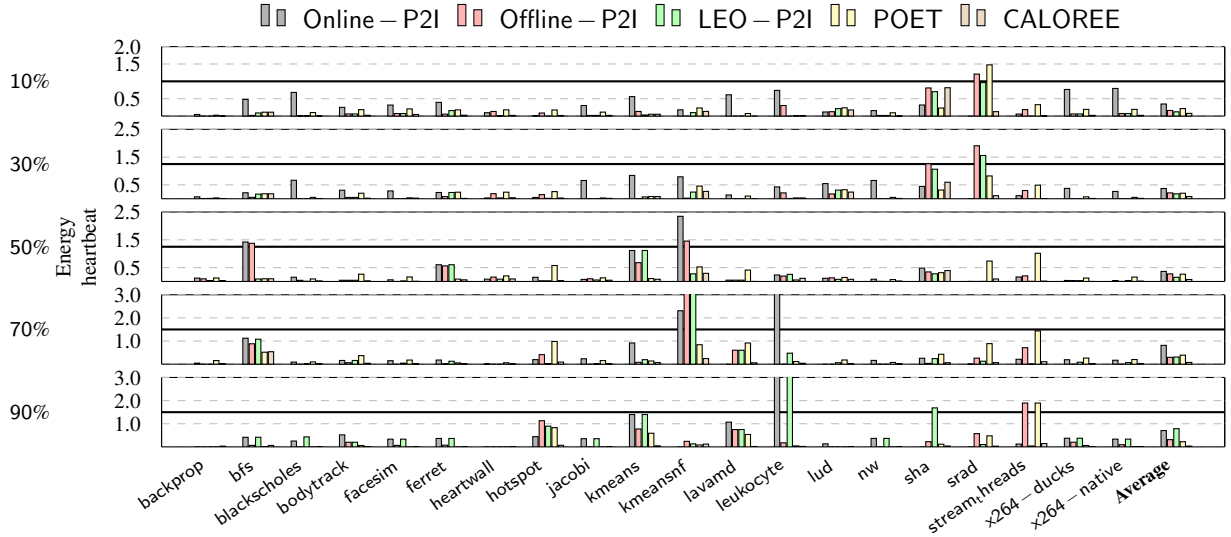

Figure 7: Single Applications energy

cation. 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 behavior specific to this application on this system.

5.3. Sensitivity to the measured samples

We examine CALOREE' sensitivity to sample size. We vary the number of samples taken online and show how that 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 **??** 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 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.

5.3.1. Phase change In this experiment we demonstrate that CALOREE allows applications to operate well by changing resource allotment even if the input to the application is changing with time. In Figure 12 we see `X264`, a video encoder application runs in 2 different phases with phase change at 180^{th} frame. The video encode is running

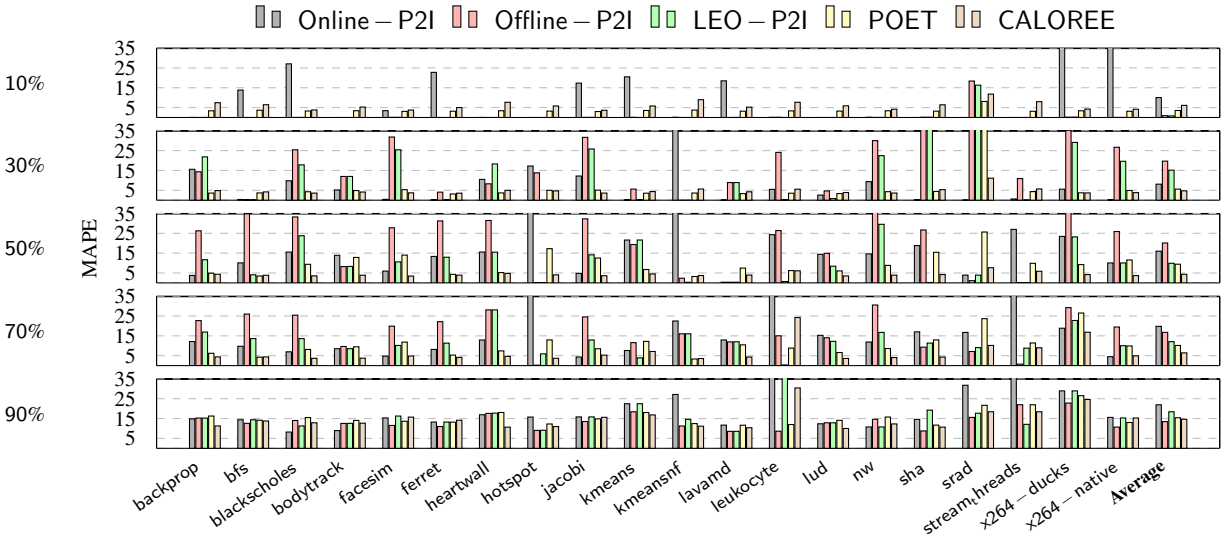

Figure 8: Multi Applications MAPE

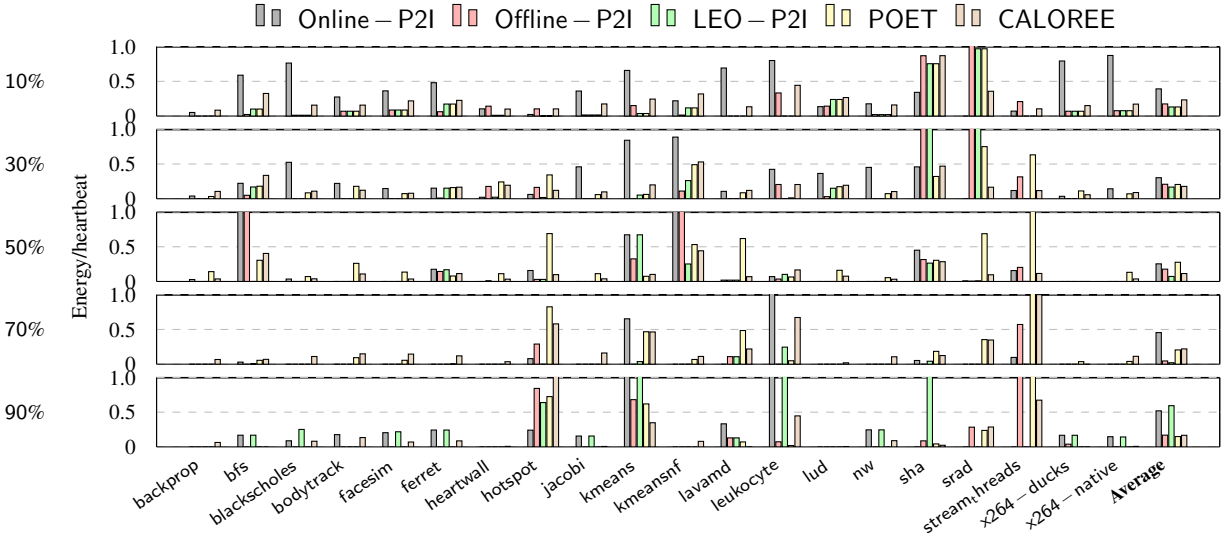

Figure 9: Multi Applications Energy

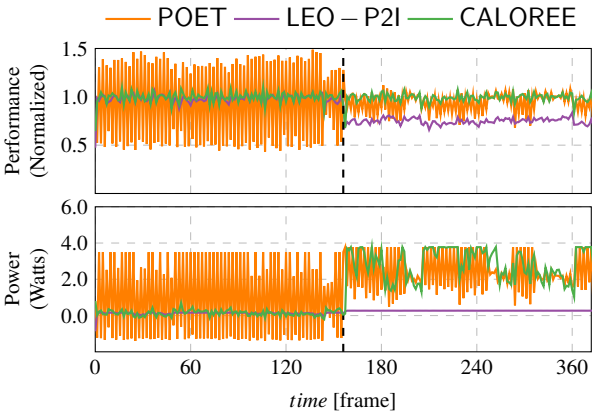

Figure 10: bodytrack multiapp

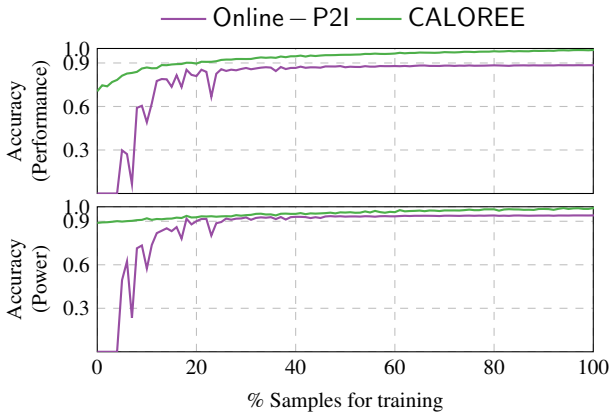

Figure 11: Estimation accuracy versus sample size.

difficult scenes in phase 1 and the scene become significantly less hard in phase-2. In phase-1, CALOREE and the baseline LEO-P2I seem to have lower MAPE as compared to POET. At the same time, these two algorithms

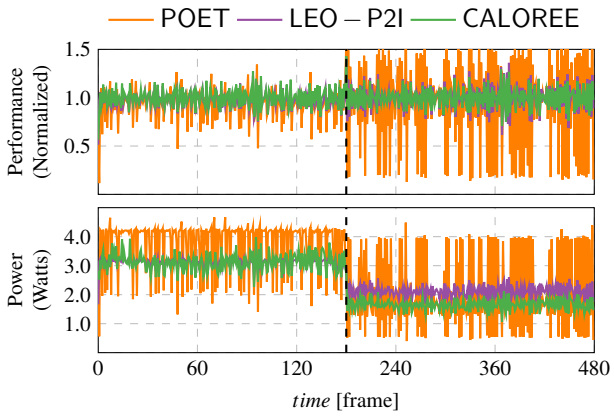

Figure 12: X264 phase change

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.

5.3.2. 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. **[TODO: Expand on the controller overhead.]**

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 in Section **??** we break learning for resource management into three 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 [46, 23, 21, 7, 2]. 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 a model that accurately captures the relationship between the observed features and the behavior to be estimated. Applying the model online, however, tends to be low overhead. The main drawback is that the models are not updated as the system runs, so there is no chance to correct mistakes or adapt to specific workloads.

A good example of an offline approach applies learning to render web pages on mobile systems with low energy [49]. This system is similar in spirit to CALOREE. 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 [25, 32, 40, 33, 1, 22]. For example, Flicker is a configurable architecture and optimization framework that uses only online models to maximize performance under a power limitation [32]. Another example, ParallelismDial, uses online adaptation to tailor parallelism to application workload [40].

6.1.3. Hybrid Approaches Some approaches combine offline predictive models with online adaptation [48, 8, 43, 9, 38, 37, 45]. 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 OS level to manage system energy consumption [38, 37]. [45].

Still other approaches combine offline modeling with online model updates [14, 4, 18]. For example, 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. LEO, the system we extend in this paper, also uses a combination of offline and online approaches. LEO collects data about a number of applications offline and combines that with a small number of observations made online for the current application [30].

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 statistical regression to build a model that is then used to synthesize a control system [44, 27, 6, 15, 17, 47, 24, 36, 39, 35]. Over a narrow range of applications the combination of offline learning and control works well, 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 [42, 12, 13, 20, 27] and web-servers [16, 26, 41] 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. Several approaches provide control libraries that encapsulate control functionality and require users to input a model [47, 39, 36, 17]. Other approaches automatically synthesize both a model and a controller for either hardware [34] or software [10, 11]. JouleGuard combines learning for energy efficiency with control for managing application parameters [14]. 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.

Perhaps the most similar approach to CALOREE is Carat [31]. Carat aggregates data across many mobile devices and sends a report to 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. One of the key enablers for this approach is the interface that allows the learned models to be applied efficiently. We propose the performance hash table to serve as this interface. While our specific implementation is tailored for the problem of meeting a performance constraint with minimal energy, we believe this data structure is generally useful for problems where a computing system must meet a constraint in one dimension while optimizing some other dimension.

CALOREE is well-suited to managing the twin challenges of modeling system complexity and reacting to system dynamics. 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. *SIGMETRICS Perform. Eval. Rev.*, 39(1):1–12, June 2011.

[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. *ACM Trans. Comp. Syst.*, 22(2), May 2004.

[14] H. Hoffmann. Jouleguard: energy guarantees for approximate applications. In *SOSP*, 2015.

[15] 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.

[16] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *Computers, IEEE Transactions on*, 56(4), 2007.

[17] 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.

[18] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *ISCA*, 2008.

[19] 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.

[20] M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. xtune: A formal methodology for cross-layer tuning of mobile embedded systems. *ACM Trans. Embed. Comput. Syst.*, 11(4), Jan. 2013.

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

[22] B. C. Lee and D. Brooks. Efficiency trends and limits from comprehensive microarchitectural adaptivity. In *ASPLOS*, 2008.

[23] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS*, 2006.

[24] B. Li and K. Nahrstedt. A control-based middleware framework for quality-of-service adaptations. *IEEE Journal on Selected Areas in Communications*, 17(9), 1999.

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

[26] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE TPDS*, 17(9):1014–1027, September 2006.

[27] 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. *IEEE Transactions on Control Systems Technology (to appear)*.

[28] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Controlling software applications via resource allocation within the heartbeats framework. In *CDC*, 2010.

[29] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE TCCA Newsletter*, pages 19–25, Dec. 1995.

[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. 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, pages 10:1–10:14, New York, NY, USA, 2013. ACM.

[32] 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.

[33] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *MICRO*, 2001.

[34] 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.

[35] 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.

[36] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for qos management. In *RTSS*, 1997.

[37] 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.

[38] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: A platform for os-level power management. In *EuroSys*, 2009.

[39] 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. *Journal of Systems Architecture*, 57(4), 2011.

[40] S. Sridharan, G. Gupta, and G. S. Sohi. Holistic run-time parallelism management for time and energy efficiency. In *ICS*, 2013.

[41] Q. Sun, G. Dai, and W. Pan. LPV model and its application in web server performance control. In *ICCSSE*, 2008.

[42] 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. *IJES*, 4(2), 2009.

[43] 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.

[44] 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.

[45] 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*, pages 413–424. IEEE, 2012.

[46] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *HPCA*, 2003.

[47] R. Zhang, C. Lu, T. Abdelzaher, and J. Stankovic. Controlware: A middleware architecture for feedback control of software performance. In *ICDCS*, 2002.

[48] X. Zhang, R. Zhong, S. Dwarkadas, and K. Shen. A flexible framework for throttling-enabled multicore management (temm). In *ICPP*, 2012.

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