CALOREE: Combining Control and Learning for Predictable Performance and Low Energy

Anonymous Author(s)

Abstract

Many modern computing systems must provide reliable performance with minimal energy. Two central challenges arise when allocating system resources to meet these conflicting goals: (1) *complexity*—modern hardware exposes diverse resources with complicated interactions—and (2) *dynamics*—performance must be maintained despite unpredictable changes in operating environment or input. Machine learning accurately predicts the performance of complex, interacting resources, but does not address system dynamics; control theory adjusts resource usage dynamically, but struggles with complex resource interaction. We therefore propose CALOREE, a combination of learning and control that automatically adjusts resource usage to meet performance requirements with minimal energy in complex, dynamic environments. CALOREE breaks resource allocation into two sub-tasks: learning speedup as a function of resource usage, and controlling speedup to meet performance requirements. CALOREE also defines a general interface allowing different learners to be combined with a controller while maintaining control's formal guarantees that performance will converge to the goal. We implement CALOREE and test its ability to deliver reliable performance on heterogeneous ARM big.LITTLE architectures in both single and multi-application scenarios. Compared to state-of-the-art learning and control solutions, CALOREE reduces deadline misses by 2–6× while reducing energy consumption by 7–10%.

1 Introduction

Large classes of computing systems—from embedded to cloud—must deliver reliable performance to users while minimizing energy to prolong battery life or lower operating costs. To address these conflicting requirements, hardware architects expose diverse, heterogeneous resources with a wide array of performance and energy tradeoffs. Software must allocate these resources to guarantee performance requirements are met with minimal energy.

There are two primary difficulties in determining how to allocate heterogeneous resources. The first is *complexity*: resources interact in intricate ways, leading to non-convex optimization spaces. The second is *dynamics*: performance requirements must be met despite unpredictable disturbances; *e.g.*, phases in input or changes in operating environment. Prior work addresses each of these difficulties individually.

Machine learning handles complex modern processors, predicting an application's performance and power as a function of resource configurations [5, 10, 12, 23, 37, 42, 43, 49, 63]. These predictions, however, are not useful if the environment changes dynamically; *e.g.*, a second application enters the system. Control theoretic approaches dynamically adjust resource usage based on the difference between measured and expected performance [7, 18, 19, 22, 31, 54, 61]. Control provides formal guarantees of convergence to the desired performance despite unpredictable interference, but these guarantees are based on known relationships between resources and performance. If the relationships are not known or there is significant error between the expected behavior and the actual behavior, the controller will not converge.

Intuitively, combining learning and control should produce predictable behavior in complex, dynamic systems. Such a combination, however, must address two major challenges:

- Dividing the resource allocation problem into sub-tasks that suit learning and control's different strengths.
- Defining abstractions allowing sub-task solutions to coordinate well with each other.

We address the first challenge by splitting resource allocation into two sub-tasks. The first is learning speedup—instead of absolute performance—so that all unpredictable external interference is viewed as a change to a *baseline* performance and the relative speedup is independent of these changes. Learning is well-suited to predicting speedups as a function of resource usage and finding Pareto-optimal tradeoffs in speedup and energy. The second sub-task is controlling speedup dynamically based on the difference between measured and desired performance. Once the learner has found Pareto-optimal tradeoffs the problem is convex and well-suited to adaptive control solutions which deliver the required speedup even in dynamic environments.

We address the second challenge by defining an interface between learning and control that maintains control's formal guarantees. This interface consists of two parts. The first is a *performance hash table* (PHT) that stores the learned relationship between configurations and speedup. The PHT allows the controller to find the resource allocation that meets a desired speedup with minimal energy and requires only constant time—$O(1)$—to access. The second part of the interface is the learned variance. Knowing this value, the controller can adjust itself to maintain formal convergence guarantees even though the speedup is predicted by a noisy

Conference'17, Washington, DC, USA
2016. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

learning mechanism, rather than directly measured—as it would be in traditional control design.

In this paper we propose CALOREE[1] a general methodology allowing different learning techniques to be paired with a controller. CALOREE tunes its internal parameters automatically; *i.e.*, *it requires no user-level inputs other than performance requirements*. We evaluate CALOREE by implementing the learners on an x86 server and the controller on heterogeneous ARM big.LITTLE devices. We compare to state-of-the-art learning (including polynomial regression [12, 49], the Netflix algorithm [3, 10], and a hierarchical Bayesian model [37]) and control (including proportional-integral-derivative [18] and adaptive, or self-tuning [30]) techniques. We set performance goals for a set of benchmark applications and then measure both the percentage of time the requirements are violated and the energy. We test both *single-app*—where an application runs alone—and *multi-app* environments—where background applications unpredictably enter the system and compete for resources. CALOREE achieves the:

- *Most reliable performance:*
  - In the *single-app* case, the best prior technique misses 12% of deadlines on average, while CALOREE misses only 5% on average—reducing deadline misses by more than 58% compared to prior approaches.
  - In the *multi-app* case, the best prior approach averages 30% deadline misses, but CALOREE misses just 5.6% of deadlines—a huge improvement over prior work.
- *Best energy savings:* We compare to an *oracle* with a perfect model of the application, system, and future events.
  - In the *single-app* case, the best prior approach averages 12% more energy consumption than the oracle, but CALOREE consumes only 5% more.
  - In the *multi-app* case, the best prior approach averages 18% more energy than the oracle, while CALOREE consumes just 8% more.

In summary, control handles dynamic environments while learning predicts performance for complex, heterogeneous processors. *CALOREE is the first work to combine learning and control to ensure application performance—both formally and empirically—without prior knowledge of the controlled application.* Formal analysis of CALOREE demonstrates convergence despite noisy inputs and shows how to integrate learned variance into control theoretic guarantees. We demonstrate the practical benefits of these contributions for mobile/embedded processors, finding CALOREE provides much more reliable performance and lower energy than learning or control solutions alone.

2 Background and Motivation

Many learning approaches predict an application's most energy efficient resource allocation. Such learning methods include *offline* techniques that build predictors with training data and then predict the behavior of new applications [11, 27, 29, 59, 60, 63]. *Online* techniques construct predictors while an application runs [28, 32, 42, 43, 51]. *Hybrid* techniques combine offline training with online predictor updates [9, 12, 37, 47, 49, 56, 58].

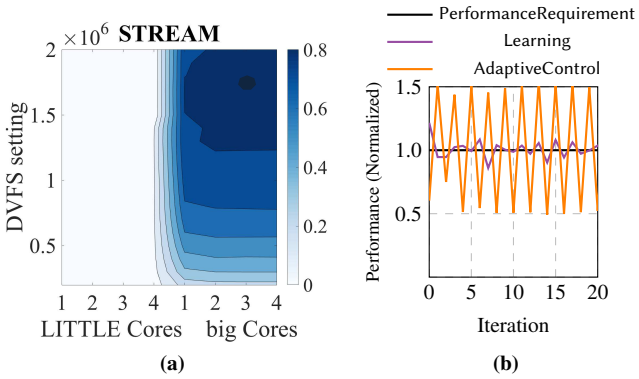


Figure 1. (a) `STREAM` performance as a function of configuration. (b) Managing `STREAM`'s performance: *Learning* handles the complex configuration space, but *control* oscillates.

Control theory provides techniques for maintaining desired behavior in dynamic systems [18]. *Adaptive controllers* or *self-tuning regulators* adjust their internal parameters in response to dynamic changes [30]. They have proven especially useful in webservers with fluctuating request rates [21, 34, 52] and multimedia applications with dynamically varying inputs [31, 35, 54]. Prior work has generalized adaptive control design by exposing key parameters to users who customize control to their needs [22, 61]. User customization provides greater flexibility, but the controller will not converge to the desired performance if the custom design does not accurately capture the relationship between resources and performance. This practice means users must not only be experts in their application domain, but must also have sufficient control knowledge to specify the parameters correctly.

This section illustrates how learning handles complexity, how control handles dynamics, and then describes a key challenge that must be overcome to combine learning and control.

2.1 *Learning* Complexity

We demonstrate how well learning handles complex resource interaction for `STREAM` on an ARM big.LITTLE processor with four big, high-performance cores and four LITTLE, energy efficient cores. The big cores support 19 clock speeds, while the LITTLE cores support 14.

Figure 1a shows `STREAM`'s performance for different resource configurations. This memory-bound application has complicated behavior: the LITTLE cores' memory hierarchy cannot deliver the required performance. The big cores' more powerful memory system delivers much greater performance, but the peak occurs with 3 big cores. Furthermore, at low

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

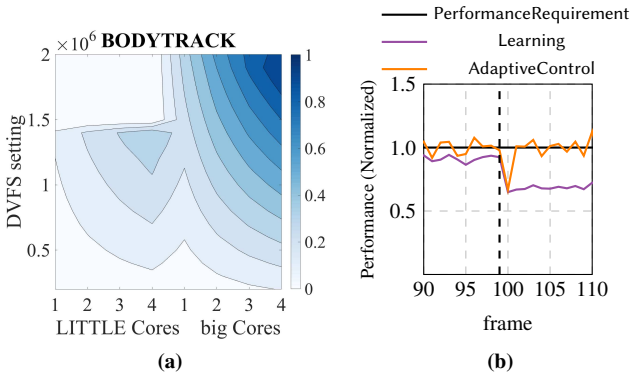


Figure 2. (a) bodytrack performance as a function of configuration. (b) Managing bodytrack's performance with another application: *control* detects the change (at the vertical dashed line) and adjusts, but *learning* cannot.

clockspeeds, these 3 big cores cannot saturate the memory bandwidth, while at high clockspeeds the performance drops as the processor overheats, triggering thermal management. For STREAM, the peak speed occurs with 3 big cores at 1.2 GHz, and it is not efficient to spend any time on the LITTLE cores. STREAM, however, does not have distinct phases, so once a resource allocator finds the most energy efficient configuration, it simply needs to maintain it.

Figure 1b shows 20 iterations of existing learning [37] and adaptive control [22] approaches allocating resources to STREAM. The x-axis shows iteration and the y-axis shows performance normalized to the requirement. The *learning* approach estimates STREAM's performance and power for all configurations and uses the lowest energy configuration that delivers the required performance. The *adaptive controller* begins with a generic notion of power/performance tradeoffs. As the controller runs, it measures performance and adjusts both the allocated resources and its own parameters. The adaptive controller dynamically adjusts to non-linearities with a series of linear approximations; however, inaccuracies in the relationship between resources and performance cause oscillations that lead to performance violations. This behavior occurs because the controller's adaptive mechanisms cannot handle STREAM's complexity, a known limitation of adaptive control systems [13, 22, 61]. Hence, the *learner*'s ability to predict complex behavior is crucial.

2.2 *Controlling* Dynamics

We now consider a dynamic environment. We begin with bodytrack running alone on the system. Figure 2a shows bodytrack's behavior. It achieves the best performance on 4 big cores at the highest clockspeed; the 4 LITTLE cores are more energy-efficient but slower. For bodytrack, the challenge is determining how to split time between the LITTLE and big cores to conserve energy while still meeting the performance requirements. Halfway through its execution, we launch a second application—STREAM—on a single big core, dynamically changing available resources.

Figure 2b shows the results of this experiment. The vertical dashed line—at frame 99—represents when the second application begins. The figure clearly shows adaptive control's benefits in this dynamic scenario. When the second application starts, the controller detects bodytrack's performance dip–rather than detecting the new application specifically—and it changes resource allocation (increasing clockspeed and moving bodytrack from 4 to 3 big cores). The learning system however, does not have any inherent mechanism to measure the change or adapt to the altered performance. While we could theoretically add feedback to the learner and re-estimate the configuration space whenever the environment changes, doing so is impractical due to high overhead for learners capable of handling this complexity [10, 11, 37].

2.3 Challenges Combining Learning and Control

The previous section motivates the first of CALOREE's contributions: splitting the resource allocation problem into prediction—handled by learning—and dynamic management—handled by control. CALOREE's second contribution is the interface that combines the two solutions. This subsection demonstrates the importance of this interface.

The controller's *pole* is a particularly important control parameter. Control engineers tune the pole to trade response time for noise sensitivity. Traditionally, the data used to set the pole comes from many observations of the controlled system and is considered *ground truth* [18]. CALOREE, however, must tune the pole based the learner's predictions, which may have noise and/or errors.

To demonstrate the pole's importance when using learned data, we again control bodytrack, using the adaptive controller from the previous subsection. Instead of measuring performance as a function of resource usage, we predict it using the learner from the first subsection. We compare the results with a carefully hand-tuned pole to those using the default pole provided by the controller developers [22].

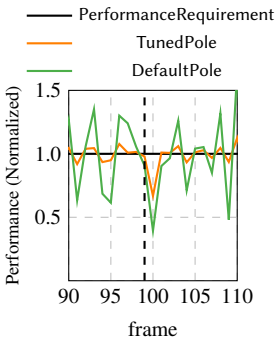


Figure 3. Comparison of carefully tuned and default poles.

As shown in Figure 3, the carefully tuned pole converges. The default pole, however, oscillates around the performance target, resulting in a number of missed deadlines. Additionally, the frames that exceed the desired performance waste energy because they spend more time on the big, inefficient cores. The pole captures the system's *inertia*—dictating how fast it should react to environmental changes. If the learner

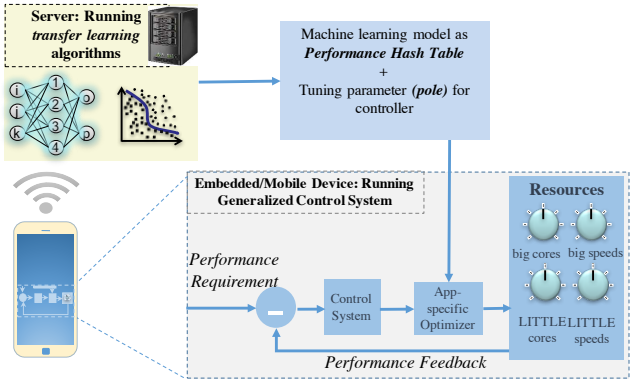


Figure 4. CALOREE overview.

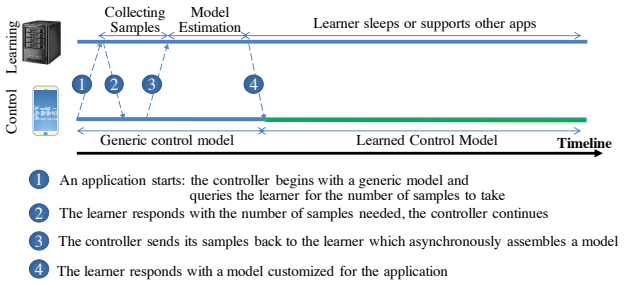


Figure 5. Temporal relationship of learning and control.

is noisy or inaccurate, the controller should trust it less and move slowly. Rather than require users with both computing and control knowledge to tune the pole, *CALOREE incorporates the learner's estimated variance to compute a pole that provides probabilistic convergence guarantees.*

3 CALOREE: Learning Control

Figure 4 shows CALOREE's approach of splitting resource management into separate learning and control tasks and defining an interface between them. When a new application enters the system with a performance requirement, an adaptive control system allocates resources using generic predictions and records performance and power. The recorded values are sent to a learner, which predicts the application's performance and power in all other resource configurations. The learner extracts those that are Pareto-optimal and packages them in a special data structure: the performance hash table (PHT). The PHT and the estimated variance are sent to the controller, which sets its pole and selects an energy minimal resource configuration with formal guarantees of convergence to the desired performance. CALOREE's only user-specified parameter is the performance requirement.

Figure 5 illustrates the asynchronous interaction between CALOREE's learner and controller. The controller starts–using a conservative, generic speedup prediction—when a new application launches. The controller sends the learner the application's name and device type (message 1, Figure 5). The learner determines how many samples are needed for an accurate prediction and sends this number to the controller (message 2). The controller takes these samples and sends the performance and power of each measured configuration to the learner (message 3). The learner may require time to make predictions; so, the controller does not wait, but continues with the conservative model. Once the learner predicts the optimal configurations for this application, it sends that data and the variance estimate to the controller (message 4), which uses the learner's application-specific predictions from then on.

Figure 5 shows several key points about the relationship between learning and control. First, the controller never waits for the learner: it uses a conservative, less-efficient control specification until the learner produces application-specific predictions. Second, the controller does not continuously communicate with the learner—this interaction happens once at application launch. Third, if the learner crashed, the controller defaults to the generic adaptive control system. If the learner crashed after sending its predictions, the controller does not need to know. Finally, the learner and controller have a clearly defined interface, so they can be run in separate processes or physically separate devices.

This section first describes traditional adaptive control. We then generalize this approach, separating out parameters to be learned. Next, we discuss the class of learning systems that work with CALOREE. Finally, we formally analyze CALOREE's guarantees.

3.1 Traditional Control for Computing

A multiple-input, multiple-output (MIMO) controller manages multiple resources to meet multiple goals. The inputs are measurements, *e.g.*, performance. The outputs are the resource settings to be used at a particular time, *e.g.*, an allocation of big and LITTLE cores and a clockspeed for each.

These difference equations describe a generic MIMO controller for allocating n resources to meet m goals at time t:[2]

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}\cdot\mathbf{x}(t) &+\mathbf{B}\cdot\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\cdot\mathbf{x}(t) & \end{aligned}, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^q$ is the controller's *state*, an abstraction of the relationship between resources and goals; q is the controller's *degree*, or complexity of its internal state. $\mathbf{u}(t) \in \mathbb{R}^n$ represents the current resource *configuration*; *i.e.*, the ith vector element is the amount of resource i allocated at time t. $\mathbf{y}(t) \in \mathbb{R}^m$ represents the value of the goal dimensions at time t. The matrices $\mathbf{A} \in \mathbb{R}^{q\times q}$ and $\mathbf{B} \in \mathbb{R}^{q\times n}$ relate the resource configuration to the controller state. The matrix $\mathbf{C} \in \mathbb{R}^{m\times q}$ relates the controller state to the expected behavior. This control definition does not assume the states or the resources are independent, but it does assume a linear relationship.

For example, in our ARM big.LITTLE system there are four resources: the number of big cores, the number of LITTLE cores, and the speeds for each of the big and LITTLE cores. There is also a single goal: performance. Thus, in

[2]We assume discrete time, and thus, use difference equations rather than differential equations that would be used for continuous systems.

this example, $n = 4$ and $m = 1$. The vector $\mathbf{u}(t)$ has four elements representing the resource allocation at time t. q is the number of variables in the controller's state which can vary between, we can choose q to be between 1 to n. The matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ capture the linear relationship between the control state $\mathbf{x}$, the resource usage $\mathbf{u}$, and the measured behavior. In this example, we know there is a non-linear relationship between the resources. We overcome this difficulty by tuning the matrices at each time step—approximating the non-linear system through a series of changing linear formulations. This approximation is a form of *adaptive* or *self-tuning* control [30]. Such adaptive controllers provide formal guarantees that they will converge to the desired performance even in the face of non-linearities, but they still assume convexity.

This controller has two major drawbacks. First, it requires matrix computation, so its overhead scales poorly in the number of resources and in the number of goals [18, 48]. Second, the adaptive mechanisms require users to specify both (1) starting values of the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ and (2) the method for updating these matrices to account for any non-convexity in the relationship between resources and performance [22, 30, 48, 61]. Therefore, typically 100s to 1000s of samples are taken at design time to ensure that the starting matrices are sufficient to ensure convergence [14, 33, 44].

3.2 CALOREE Control System

To overcome the above issues, CALOREE abstracts the controller of Eqn. 1 and factors out those parameters to be learned. Specifically, CALOREE takes three steps to transform a standard control system into one that works without prior knowledge of the application to be controlled:

1. controlling *speedup* (which is an abstraction of performance) rather than resources;
2. turning speedup into a minimal energy *resource schedule*;
3. and exploiting the *problem structure* to solve this scheduling problem in constant time.

These steps assume a separate learner has produced predictions of how resource usage affects performance and power. The result is that CALOREE's controller runs in constant time without requiring any user-specified parameters.

3.2.1 Controlling Speedup

CALOREE converts Eqn. 1 into a single-input (performance), single-output (speedup) controlling using $\mathbf{A} = 0$, $\mathbf{B} = b(t)$, $\mathbf{C} = 1$,$\mathbf{u} = speedup$, and $y = perf$; where $b(t)$ is a time-varying parameter representing the application's *base speed*—the speed when all resources are available—and $perf$ is the measured performance. Using these substitutions, we eliminate $\mathbf{x}$ from Eqn. 1 to relate speedup to performance:

$$perf(t) = b(t) \cdot speedup(t-1) \tag{2}$$

While $b(t)$ is application-specific. CALOREE assumes base speed is time-variant as applications will transition through phases and it estimates this value online using the standard technique of Kalman filter estimation [55].

CALOREE must reduce the error between the target performance and the goal: $error(t) = goal - perf(t)$. Given Eqn. 2, CALOREE uses the integral control law [18]:

$$speedup(t) \quad = \quad speedup(t-1) - \frac{1-\rho(t)}{b(t)}.error(t) \tag{3}$$

which states that the speedup at time t is a function of the previous speedup, the error at time t, the base speed $b(t)$, and the controller's *pole*, $\rho(t)$. Standard control techniques statically determine the pole and the base speed, but CALOREE *dynamically sets the pole and base speed to account for error in the learner's predictions—an essential modification for providing formal guarantees of the combined control and learning systems.* For stable control, CALOREE ensures $0 \leq \rho(t) < 1$. Small values of $\rho(t)$ eliminate error quickly, but make the controller more sensitive to the learner's inaccuracies. Larger $\rho(t)$ makes the system more robust at the cost of increased convergence time. Section 3.5 describes how CALOREE sets the pole, but we first address converting speedup into a resource allocation.

3.2.2 Converting Speedup to Resource Schedules

CALOREE must map Eqn. 3's speedup into a resource allocation. On our example ARM big.LITTLE architecture we map speedup into an allocation of big and LITTLE cores as well as a speed for both (big and LITTLE cores are in separate clock domains).

The primary challenge is that speedups in real systems are discrete non-linear functions of resource usage, 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.

The assignment of time to resource configurations is a *schedule*; *e.g.*, spending 10 ms on the LITTLE cores at 0.6 GHz and then 15 ms on the big cores at 1 GHz. Typically many schedules can deliver a particular speedup and CALOREE must find one with minimal energy. Given a time interval T, the $speedup(t)$ from Eqn. 3, and C different resource configurations, CALOREE solves:

$$\underset{\tau \in \mathbb{R}^C}{\text{minimize}} \quad \sum_{c=0}^{C-1} \tau_c \cdot p_c \tag{4}$$

$$s.t. \quad \sum_{c=0}^{C-1} \tau_c \cdot s_c = speedup(t)T \tag{5}$$

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

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

where p_c and s_c are configuration c's estimated *powerup*—analogous to speedup—and speedup; τ_c is the time to spend in configuration c. Eqn. 4 is the objective: minimizing energy (power times time). Eqn. 5 states that the average speedup

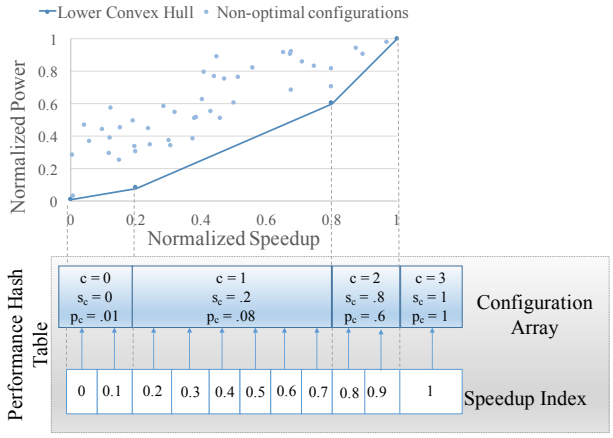


Figure 6. Data structure to efficiently convert required speedup into a resource configuration.

must be maintained, while Eqn. 6 requires the time to be fully utilized. Eqn. 7 simply avoids negative time.

3.3 Exploiting Problem Structure for Fast Solutions

By encoding the learner's predictions in the performance hash table (PHT), CALOREE solves Eqns. 4–7 in constant time.

Kim et al. analyze the problem of minimizing energy while meeting a performance constraint and observe that there must be an optimal solution with the following properties [24]:

- At most two of τ_c are non-zero, meaning that at most two configurations will be used in any time interval.
- If you chart the configurations in the power and performance tradeoff space (*e.g.*, the top half of Figure 6) the two configurations with non-zero τ_c lie on the lower convex hull of the points in that space.
- The two configurations with non-zero τ_c are adjacent on the convex hull: one above the constraint and one below.

The PHT (shown in Figure 6) provides constant time access to points on the lower convex hull. It consists of two arrays: the first being pointers into the second: a configuration array, which stores resource configurations on the lower convex hull sorted by speedup. Recall speedups are computed relative to the base speed, which uses all resources. The largest estimated speedup is therefore 1, so CALOREE needs only consider speedups between 0 and 1. The first array of pointers has a *resolution* indicating how many decimal points of precision it captures and it is indexed by speedup. The example in Figure 6 has a resolution of 0.1. Each pointer in the first array points to the configuration in the second array that has the largest speedup less than or equal to the index.

CALOREE computes $speedup(t)$ and uses the PHT to convert speedup into two configurations: *hi* and *lo*. To find the *hi* configuration, CALOREE clamps the desired speedup to the largest index lower than $speedup(t)$, indexes into the configuration array, and then walks forward until it finds the first configuration with speedup higher than $speedup(t)$. To find *lo*, it clamps the desired speedup to the smallest index higher than $speedup(t)$, indexes into the configuration array, and then walks backwards until it finds the configuration with the largest speedup less than $speedup(t)$.

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

CALOREE sets τ_{hi} and τ_{lo} by solving:

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

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

where the controller provides $speedup(t)$ and the learner predicts s_c. By solving Eqns. 8 and 9, CALOREE has turned the controller's speedup into a resource schedule using predictions stored in the PHT.

3.4 CALOREE Learning Algorithms

The previous subsection describes a general control system, which can be customized with a number of different learning methods. The requirements on the learner are that it must produce 1) predictions of each resource configuration's speedup and powerup and 2) estimate of its own variance. This section describes the general class of learning mechanisms that meet these requirements.

We refer to application-specific predictors as *online* because they work for the current application and do not incorporate knowledge of other applications. We refer to general predictors as *offline* as they use prior observations of other applications to predict the behavior of a new application. A third class of *transfer learning* predictors combines information from the previously seen applications and current application to make the predictions on the application in hand [41]. Transfer learning obtains higher prediction accuracies since it augments online data with offline information from other applications. In this work we use transfer learning techniques because CALOREE's separation of learning and control makes it easy to incorporate data from other applications—the learner in Figure 5 can simply aggregate data from multiple controllers. We describe two examples of appropriate transfer learning algorithms.

Netflix Algorithm: The Netflix problem is a famous challenge posted by Netflix to predict users' movie preferences. The challenge was won by realizing that if 2 users both like some movies, they might have similar taste in other movies. This approach allows predictors to borrow large amounts of data from other users to answer some questions about a new user. Delimitriou and Kozyrakis use this algorithm to predict application response to heterogeneous resources in data centers [10, 11].

Bayesian Predictors: A hierarchical Bayesian model (HBM) provides a statistically sound framework for learning across

while *True* **do**
  Measure streaming application performance
  Compute required speedup (Equation (2))
  Lookup s_{hi} and s_{lo} with PHT
  Compute τ_{hi} and τ_{lo} (Equations 8 & 9)
  Configure to system to *hi* & sleep τ_{hi}.
  Configure to *lo* & sleep τ_{lo}.
end while

Algorithm 1: CALOREE's runtime control algorithm.

applications and devices [37]. In the HBM, 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. In practice, an HBM predicts behavior for a new application using a small number of observations and combining those with the large number of observations of other applications. Rather than over-generalizing, the HBM uses only similar applications to predict new application behavior. The HBM's accuracy increases as more applications are observed because increasingly diverse behaviors are represented in the pool of prior knowledge. Of course, the computational complexity of learning also increases with increasing applications.

3.5 Formal Analysis

Control System Complexity

CALOREE's control system (see Algorithm 1) runs on the local device along with the application under control, so its overhead must be minimal. In fact, each controller invocation is $O(1)$. The only parts that are not obviously constant time are the PHT lookups. Provided the PHT resolution is sufficiently high to avoid collisions, then each PHT lookup requires constant time.

Control Theoretic Formal Guarantees

The controller's pole $\rho(t)$ is critical to providing control theoretic guarantees in the presence of learned—rather than directly measured—data. CALOREE requires any learner estimate not only speedup and powerup, but also the variance σ. CALOREE uses this information to derive a lower bound for the pole which guarantees probabilistic convergence to the desired performance. Specifically, we prove that with probability 99.7% CALOREE converges to the desired performance if the pole is

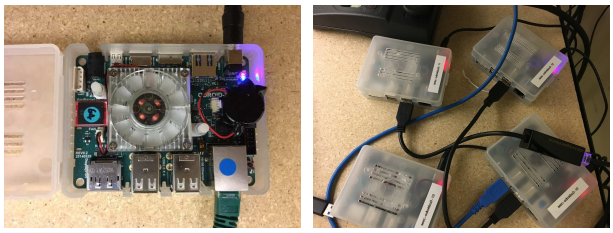
$$\lfloor 1 - \lfloor max(\hat{s})/(min(\hat{s}) - 3\sigma)\rfloor_0\rfloor_0 \leq \rho(t) < 1,$$

where $\lfloor x\rfloor_0 = \max(x, 0)$ and $\hat{s}$ is the estimated speedup. See appendix A for the proof. Users who need higher confidence can set the scalar multiplier on σ higher; *e.g.*, using 6 provides a 99.99966% probability of convergence.

Thus we provide a lower-bound on the value of $\rho(t)$ required for a user to be confident that CALOREE will converge to the desired performance. This pole value only considers performance, and not energy efficiency. In practice, we find it better to use a higher pole based on the *uncertainty* between the controller's observed energy efficiency and that predicted by the learner. We follow prior work [53] in quantifying uncertainty as $\beta(t)$, and setting the pole based on this uncertainty:

$$\begin{aligned} \beta(t) &= \exp\left(-\left(\left|\frac{\bar{s}(t)}{\bar{p}(t)} - \frac{\hat{s}(t)}{\hat{p}(t)}\right|\right)/5\right) \\ \rho(t) &= \frac{1-\beta(t)}{1+\beta(t)} \end{aligned} \tag{10}$$

where $\bar{s}$ and $\bar{p}$ are the measured values of speedup and power up and $\hat{s}$ and $\hat{p}$ are the estimated values from the learner. This measure of uncertainty captures both power and performance. We find that it is generally higher than the pole value given by our lower bound, so in practice CALOREE sets the pole dynamically to be the higher of the two values and CALOREE makes spot updates to the estimated speedup and power based on its observations.



(a) **(b)**

Figure 7. ODROID-XU3 boards used in the evaluation.

4 Experimental Setup

4.1 Platform and Benchmarks

We run applications on four ODROID-XU3 devices with Ubuntu 14.04, as shown in Figure 7. The ODROIDs have Samsung Exynos 5 Octa processors using the ARM big.LITTLE architecture. Each has 19 speed settings for the 4 big cores and 13 for 4 LITTLE cores. Each board has an on-board power meter updated at 1/4 s intervals capturing core, GPU, and memory.

We use 20 benchmarks from different suites including PARSEC [4], Minebench [39], Rodinia [6], and STREAM [36]. Figure 8 shows the variety of workloads indicated by the *lack-of-fit*, or the absence of correlation between frequency and performance. Applications with high lack-of-fit do not speed up with increasing frequency—typical of memory bound applications. Applications with low lack-of-fit increase performance with increasing clock speed [38]. Applications with intermediate lack-of-fit tend to improve with increasing clock speed up to a point and then stop. Each application has an outer loop which processes one unit in a data stream (*e.g.*, a point for `kmeans` or a frame for `x264`). The application signals the completion of processing a stream element using a standard API [20]. Performance targets are specified as application-specific latencies for these stream elements.

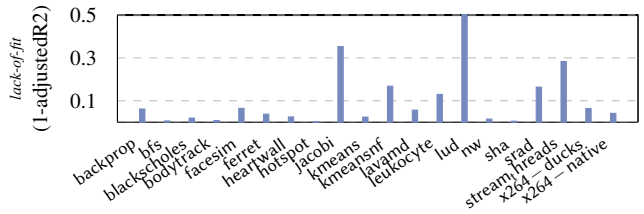


Figure 8. *Lack-of-fit* for performance vs clock-speed. Lower lack-of-fit indicates a more compute-bound application, higher values indicate a memory-bound one.

4.2 Evaluation Metrics

To test a variety of latency requirements, we run our applications with targets that represent 50-90% of the maximum speed and evaluate CALOREE under each constraint. For these targets, the key decision is how much work can be done on the little cores without violating the performance requirements. We quantify performance reliability by measuring the number of deadlines that were missed for each application and performance target. If the application processes n elements total and m of those elements took longer than the target latency we compute deadline misses as:

$$deadline\,misses = 100\% \cdot \frac{m}{n}. \tag{11}$$

We evaluate energy savings by constructing an oracle. We run every application in every resource configuration and record performance and power for every stream element. By post-processing this data we determine the optimal resource configuration for each stream element and performance target. To compare across applications, we normalize energy:

$$normalized\,energy = 100\%. \left(\frac{e_{measured}}{e_{optimal}} - 1 \right) \tag{12}$$

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

4.3 Points of Comparison

We compare to existing learning and control approaches:

1. *Race-to-idle*: This well-known heuristic allocates all resources to the application to complete each stream element as fast as possible, then idles until the next element is available [24, 26, 38]. This heuristic requires no knowledge of the application and never misses deadlines in a single-application scenario.
2. *PID-Control*: a standard single-input (performance), multiple-output (big/LITTLE core counts and speeds) proportional-integral-controller representative of several that have been proposed for computer resource management [18, 48]. This controller is tuned to provide the best average case behavior across all applications and targets.
3. *Online*: measures a few sample configurations then performs polynomial multivariate regression to estimate unobserved configurations' behavior [32, 37, 43].

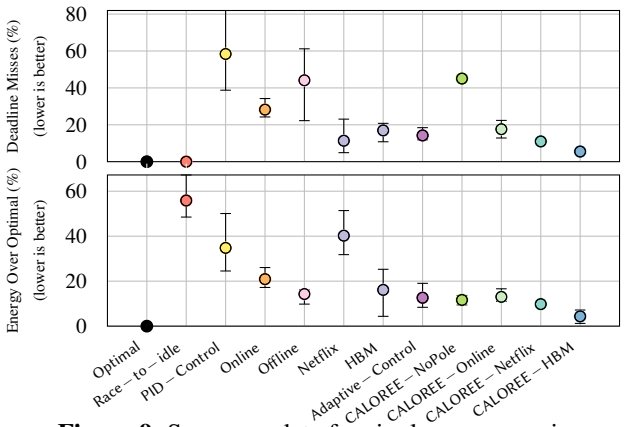


Figure 9. Summary data for single-app scenario.

4. *Offline*: does not observe any values for the current application—instead using previously observed applications to estimate power and performance as a linear regression [27, 29, 60].
5. *Netflix*: a matrix completion algorithm for the Netflix challenge. Variations of this approach allocate heterogeneous resources in data centers [10, 11].
6. *HBM*: a hierarchical Bayesian learner previously used to allocate resources to meet performance goals with minimal energy in server systems [37].
7. *Adaptive-Control*: a state-of-the-art, adaptive controller that meets application performance with minimal energy [22]. This approach requires a user-specified model relating resource configuration to performance and power. For this paper, we use the *Offline* learner's predictions.

CALOREE is a framework for combining different learners with the CALOREE control system. We compare the above baselines to four different versions of CALOREE:

1. *CALOREE-NoPole*: uses the HBM learner, but sets the pole to 0, which will show the importance of incorporating the learned variance into control. All other versions of CALOREE set the pole according to Section 3.5.
2. *CALOREE-online*: uses the online learner.
3. *CALOREE-Netflix*: uses the Netflix learner.
4. *CALOREE-HBM*: uses the HBM learner.

In all cases that require prior knowledge, we ensure that knowledge of the application under test is never included in that set of prior knowledge. Specifically, we use leave-one-out cross validation: to test application x, we form a set of all other applications, train the learners, and then test on x.

5 Experimental Evaluation

5.1 Performance and Energy for Single App

We set a range of performance targets from 50-90% of the maximum achievable performance and measure the deadline misses and energy over optimal for all points of comparison. Figure 9 represents the summary results as an average error across all targets for the single application scenario. This figure shows two charts with the percentage of deadline misses

in the top chart and the energy over optimal in the bottom. The dots show the average for each technique, while the error bars show the minimum and maximum values.

Not surprisingly, race-to-idle meets all deadlines, but its conservative resource allocation has the highest average energy consumption. Among the prior learning approaches Netflix has the lowest average deadline misses (11%), but with high energy (40% more than optimal), while the HBM has higher deadline misses (17%) but with significantly lower energy consumption (16%). Adaptive control achieves similar deadline misses (14%) with lower average energy than any of the prior learning approaches (12%). CALOREE with no pole misses 45% of all deadlines, which is clearly unacceptable.

When we allow CALOREE to adaptively tune its pole, however, we see greatly improved results. The best combination is CALOREE with the HBM, which misses only 5.5% of deadlines on average, while consuming just 4.4% more energy than optimal. These numbers represent large improvements in both performance reliability and energy efficiency compared to prior approaches. The other learners paired with CALOREE achieve similar results to the prior adaptive control approach.

The summary data shows us that the best prior approaches are race-to-idle, HBM, and adaptive control. Figure 10 shows the detailed results for the 60, 75, and 90% targets comparing the best of these prior approaches to CALOREE with no pole and CALOREE coupled with the HBM—other data has been omitted for space. The benchmarks are shown on the x-axis; the y-axis shows the number of deadline misses and the normalized energy, respectively.

5.2 Performance and Energy for Multiple Apps

We again launch each benchmark with a performance target (the same targets as the prior study). Halfway through execution, we start another application randomly drawn from our benchmark set—bound to one big core—which interferes with the application CALOREE is controlling. Meeting the required performance in this dynamic scenario tests CALOREE's ability to react to environmental changes.

Figure 11 summarizes the results as the average number of deadline misses and energy over optimal for all approaches. We note that some targets are unachievable for some applications. Due to these unachievable targets, both optimal and race-to-idle show some deadline misses. Race-to-idle misses more deadlines than optimal because it cannot make use of LITTLE cores to do some work, it simply continues using all big cores despite the degraded performance due to the second application. In fact, most approaches do badly in this scenario—even adaptive control misses 50% of the deadlines. CALOREE with the HBM produces the lowest deadline misses with an average of 20%, which is only 5% more than optimal. It also produces the second lowest energy, using slightly more than race-to-idle because it uses LITTLE cores to make up for some work that cannot be done on a big core due to the second application. Figure 12 shows the detailed results. 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.

5.3 Adapting to Phase Changes

We compare CALOREE and Adaptive-Control reacting to input variations. Figure 13 shows the `x264` video encoder application with 2 different phases caused by a scene change in the input occurring at the 500^{th} frame. The first scene is difficult and the second one is much easier. In the first, CALOREE is closer to the desired performance (1 in the figure) and operates at a lower power state compared to Adaptive-Control. Adaptive-Control is not operating at a configuration on the Pareto frontier of power and performance. When the input changes, CALOREE still meets the performance target with fewer fluctuations compared to Adaptive-Control.

5.4 The Pole's Importance

LAVAMD has the most complicated responses to resource usage on our system, with multiple local optima as shown in Figure 14a. Section 3.5 presents an analytical argument that tuning the controller to learned variance prevents oscillation and provides probabilistic control theoretic guarantees despite using noisy, learned data to control such complicated behavior. We now demonstrate this empirically by showing LAVAMD's single-app behavior using both CALOREE-NoPole and CALOREE-HBM to meet the 80% target.

Figure 14b shows the results, with time on the x-axis and normalized performance and power on the respective y-axes. CALOREE-NoPole oscillates around the desired performance and causes wide fluctuations in power consumption. In contrast, after receiving the predictions from its learner, CALOREE provides reliable performance right at the target value (normalized to 1). CALOREE also saves tremendous energy because it does not oscillate but uses a mixture of big and LITTLE cores to keep energy near minimal.

5.5 Sensitivity to the Measured Samples

We vary the number of samples taken and show how it affects learning accuracy for the Online, Netflix, and HBM learners. We quantify accuracy as how close the learner is to ground truth (found through exhaustive exploration), with 1 meaning the learner perfectly predicted the real performance or power. Accuracy is significant because the smaller the number of samples, the faster the controller can switch to the learner's application-specific predictions.

Figure 15 compares the Online, HBM, and Netflix learners for both performance (top) and power (bottom). The figure shows sample size on the x-axis and accuracy on the y-axis. The HBM initially performs as well as Offline. As sample size increases, HBM's accuracy uniformly improves, exceeding 90% after 20 samples. The Online approach needs at

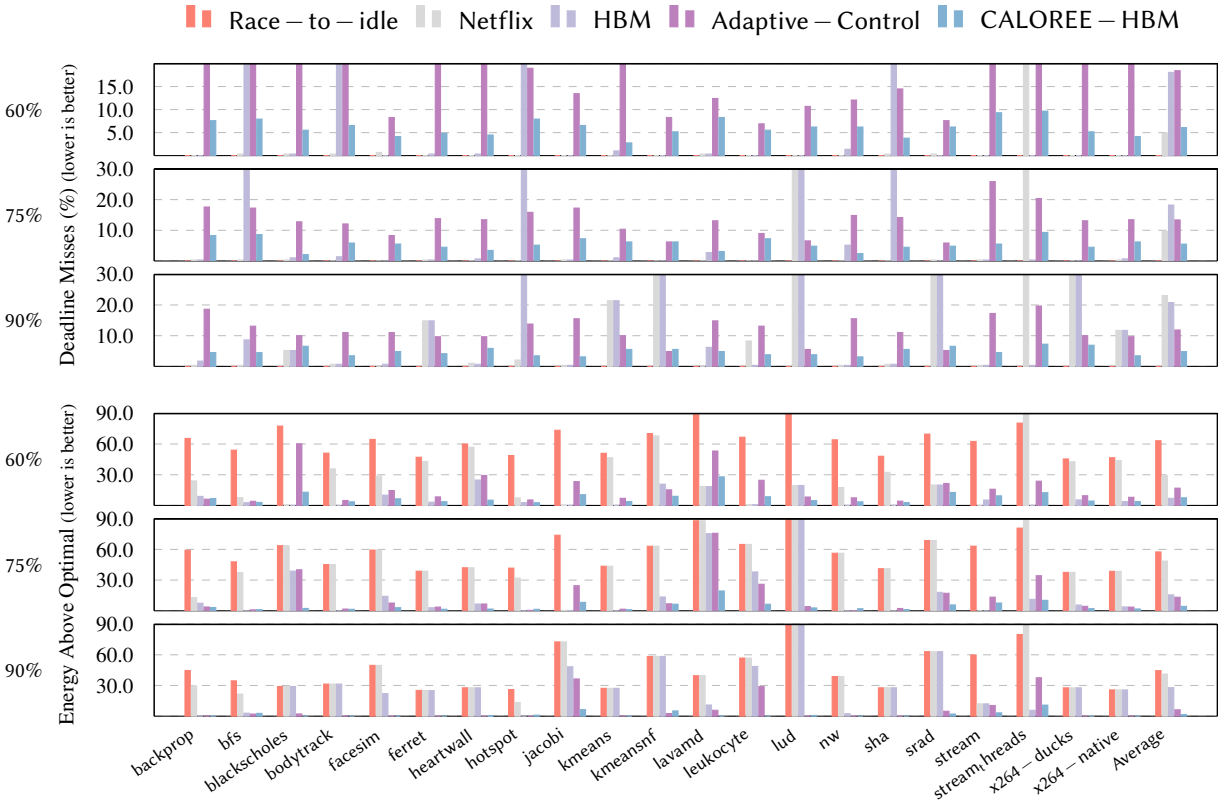


Figure 10. Comparison of application performance error and energy for single application scenario.

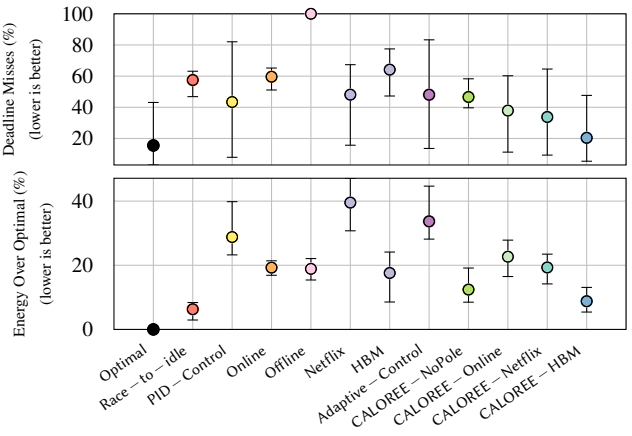


Figure 11. Summary data for multi-app scenario.

least 7 samples to even generate a prediction. As Online receives more samples, its accuracy improves but never exceeds HBM's for the same number of samples. Netflix is very noisy for small sample sizes, but after about 50, it is competitive with HBM. These results not only demonstrate the sensitivity to sample size, they show why CALOREE-HBM achieves better results than the other learners.

5.6 Overhead

CALOREE's main source of overhead is sampling, where the applications need to run through a few configurations before CALOREE can reliably estimate the entire power and performance frontier. 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 to run any of our applications. Additionally, CALOREE's asynchronous communication means that the controller never waits for the learner. Using the four ODROIDs in our experimental system, each board only needs to contribute 4 samples to achieve 90% accuracy. In the worst case (`facesim`), this sampling overhead is less than 2%. For all other benchmarks it is lower, and for most it is negligible.

The controller requires only a few floating point operations to execute, plus the table lookups in the PHT. To evaluate its overhead, we time 1000 iterations. 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 controller has negligible impact on performance and energy consumption of the controlled device.

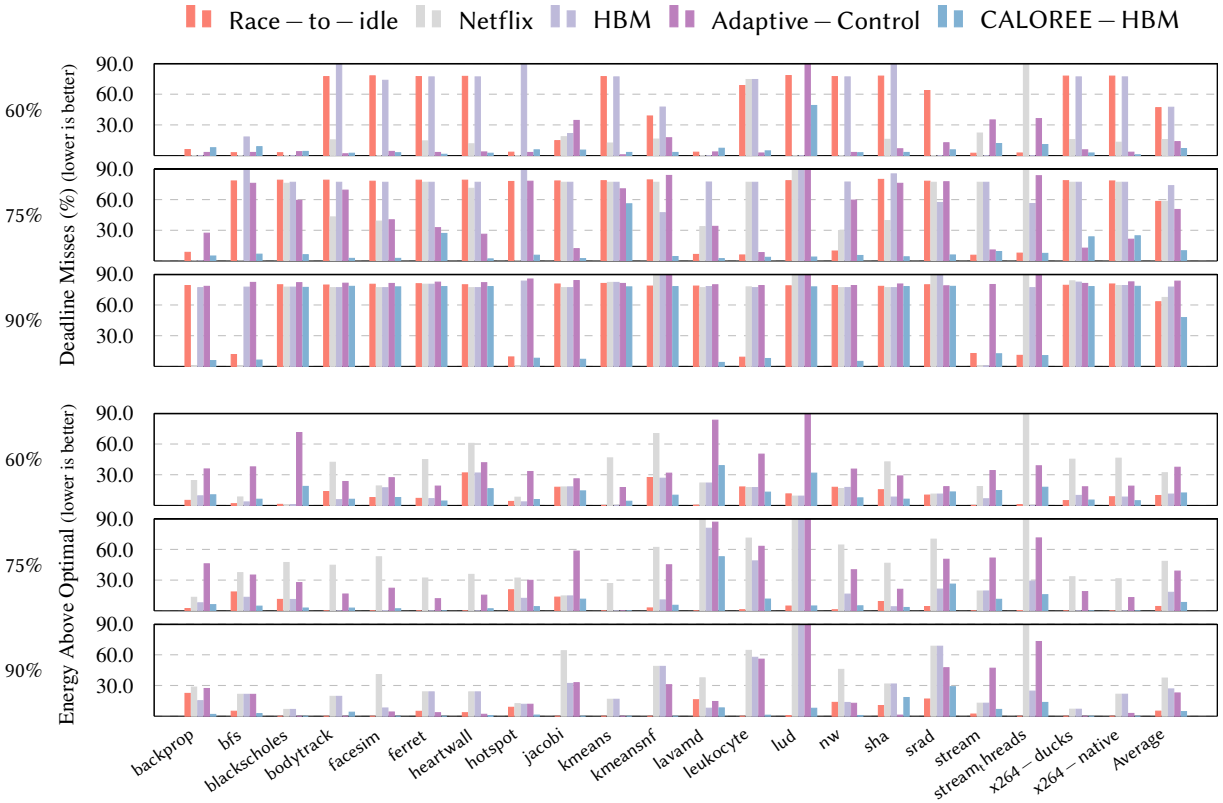


Figure 12. Comparison of application performance error and energy for multiple application scenario.

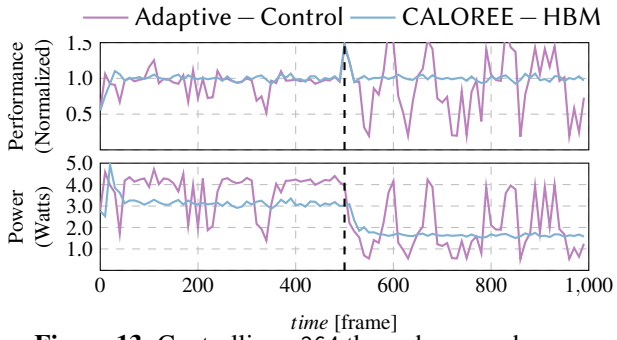


Figure 13. Controlling x264 through scene changes.

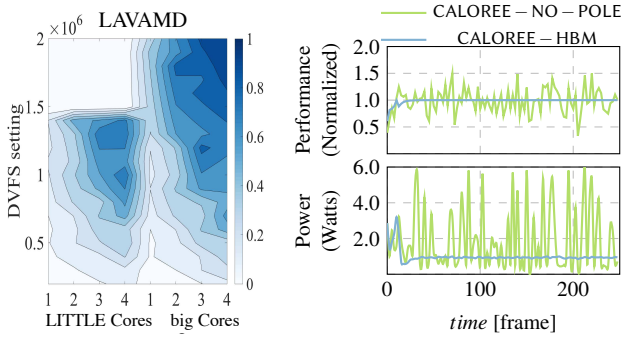


Figure 14. (a) LAVAMD's performance with different resources. (b) The pole's effects on LAVAMD's behaviors.

6 Related Work

Offline Learning approaches build predictors before deployment and then use those fixed predictors to allocate resources [2, 8, 27, 29, 59]. The training phase is expensive, requiring both a large number of samples and substantial computation. Applying the predictor online, however, is low overhead. The main drawback is that the predictions are not updated as the system runs: a problem for adapting workloads. Carat is an offline learner that aggregates data across multiple devices to generate a report for human users about how to configure their device to increase battery life [40]. While both Carat and CALOREE learn across devices, they have very different goals. Carat returns very high-level information to human users; *e.g.*, update a driver to extend battery life. CALOREE automatically builds and applies low-level predictions to save energy.

Online Learning techniques observe the current application to tune system resource usage for that application [1, 28, 32, 42, 43, 51]. For example, Flicker is a configurable architecture and optimization framework that uses online prediction

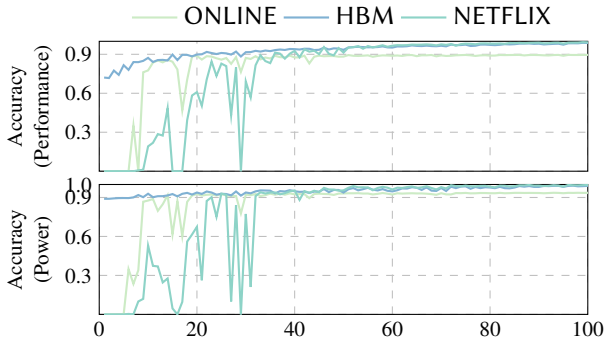


Figure 15. Estimation accuracy versus sample size.

to maximize performance under a power limitation [42]. Another example, ParallelismDial, uses online adaptation to tailor parallelism to application workload [51].

Hybrid Approaches combine offline predictions with online adaptation [9, 12, 47, 49, 56, 58, 62]. For example, Dubach et al. use a hybrid scheme to optimize the microarchitecture of a single core [12]. Such predictors have also been employed at the operating system level to manage system energy consumption [47, 49, 58]. Other approaches combine offline prediction with online updates [5, 19, 23]. For example, Bitirgen et al use an artificial neural network to allocate resources to multiple applications in a multicore [5]. The neural network is trained offline and then adapted online using measured feedback to maximizes performance but without consideration for energy minimization.

Control solutions can be thought of as a combination of offline prediction with online adaptation. The offline phase involves substantial empirical measurement that is used to synthesize a control system [7, 22, 31, 45, 46, 50, 57, 61]. The combination of offline learning and control works well over a narrow range of applications, as the offline data captures the general behavior of a class of application and require negligible online overhead. This focused approach is extremely effective for multimedia applications [15, 16, 25, 35, 54] and web-servers [21, 34, 52] because the workloads can be characterized ahead of time to produce sound control.

Indeed, the need for good predictions 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 specializing for that pair. More general controllers, which work with a range of applications, have addressed the need for accurate predictions in various ways. Some provide libraries that encapsulate control functionality and require user-specified models [17, 22, 46, 50, 61]. Others automatically synthesize both a predictor and a controller for either hardware [44] or software [13, 14]. JouleGuard combines learning for energy efficiency with control for managing application parameters [19]. In JouleGuard, a learner adapts the controller's coefficients to model uncertainty, but JouleGuard's learner does not produce a new set of predictions. 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 separate learner generates an application-specific predictions automatically. By offloading the learning task, CALOREE (1) combines data from many applications and systems and (2) applies computationally expensive, but highly accurate learning techniques.

7 Conclusion

While much recent work has built systems to support learning and data science, in this work we use learning and data to build better systems. Specifically we propose CALOREE, a combination of machine learning and control for managing resources to meet performance requirements with minimal energy. CALOREE's unique contributions are (1) breaking resource allocation into two sub-tasks: learning complexity and controlling dynamics; and (2) proposing an interface that combines the two efficiently while maintaining formal guarantees that the combination will converge to the desired performance. Our empirical results show that this combination produces more reliable performance and lower energy than either learning or control alone.

A Probabilistic Convergence Guarantees

Theorem A.1. *Let* $\mathbf{s}$ *and* $\hat{\mathbf{s}}$ *denote the true and estimated speedups of various configurations in set* C *as* $\mathbf{s} \in \mathbb{R}^{|C|}$. *Let* σ *denote the estimation error for speedups such that,* $\hat{s}_i \sim N(s_i, \sigma^2)\ \forall\ i$. *We can show that with probability greater than 99.7%, the pole* $\rho(t)$ *can be chosen to lie in the range,* $\lfloor 1 - \lfloor max(\hat{s})/(min(\hat{s}) - 3\sigma)\rfloor_0\rfloor_0, 1)$, *where* $\lfloor x\rfloor_0 = \max(x, 0)$.

Proof. Let Δ denote the multiplicative error over speedups, such that $\widehat{speedup}(t)\Delta = speedup(t)$. To guarantee convergence the value of pole, $\rho(t)$ can vary in the range $\lfloor 1 - \frac{2}{\Delta})\rfloor_0, 1)$[13]. The lowest value of $\rho(t)$ offer the fastest convergence. We have described in Equations 8 & 9 that any speedup can be written as a linear combination of two configuration speedups as,

$$speedup(t) = \hat{s}_{hi} \cdot \tau_{hi} + \hat{s}_{lo} \cdot (T - \tau_{hi}) \tag{13}$$

$$\widehat{speedup}(t) = s_{hi} \cdot \tau_{hi} + s_{lo} \cdot (T - \tau_{hi}) \tag{14}$$

We can upper bound and lower bound each of these terms,

$$speedup(t) \leq T\hat{s}_{hi} \text{ and } \widehat{speedup}(t) \geq Ts_{lo} \tag{15}$$

The estimates of speedups are close to the actual speedups since $\hat{s} \sim N(s, \sigma^2)$, therefore with probability greater than 99.7% and the speedups can be given by, $s_{lo} \geq \hat{s}_{lo} - 3\sigma$. Hence, $\widehat{speedup}(t) \geq T(\hat{s}_{lo} - 3\sigma)$. Since, over all configurations, $\Delta \leq \lfloor max(\hat{s})/(min(\hat{s}) - 3\sigma)\rfloor_0$, we can choose the pole from the range, $(\lfloor 1 - \lfloor max(\hat{s})/(min(\hat{s}) - 3\sigma)\rfloor_0\rfloor_0, 1)$.

□

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] R. M. Bell, Y. Koren, and C. Volinsky. *The BellKor 2008 solution to the Netflix Prize*. Tech. rep. ATandT Labs, 2008.

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

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

[6] 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.

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

[8] 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.

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

[10] C. Delimitrou and C. Kozyrakis. "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters". In: *ASPLOS*. 2013.

[11] C. Delimitrou and C. Kozyrakis. "Quasar: Resource-efficient and QoS-aware Cluster Management". In: *ASPLOS*. 2014.

[12] 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.

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

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

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

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

[17] A. Goel, D. Steere, C. Pu, and J. Walpole. "SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit". In: *2nd USENIX Windows NT Symposium*. 1998.

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

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

[20] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. "Application Heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments". In: *ICAC*. 2010.

[21] 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).

[22] 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.

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

[24] 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.

[25] 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).

[26] E. Le Sueur and G. Heiser. "Slow Down or Sleep, That is the Question". In: *Proceedings of the 2011 USENIX Annual Technical Conference*. Portland, OR, USA, 2011.

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

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

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

[30] W. Levine. *The control handbook*. CRC Press, 2005.

[31] 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).

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

[33] L. Ljung. *System Identification: Theory for the User*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN: 0-13-656695-2.

[34] 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.

[35] 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)* ().

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

[37] 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.

[38] 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.

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

[40] 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.

[41] S. J. Pan and Q. Yang. "A Survey on Transfer Learning". In: *IEEE Trans. on Knowl. and Data Eng.* 22.10 (Oct. 2010), pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191. URL: http://dx.doi.org/10.1109/TKDE.2009.191.

[42] 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.

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

[44] 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.

[45] 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.

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

[47] 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.

[48] A. Sharifi, S. Srikantaiah, A. K. Mishra, M. Kandemir, and C. R. Das. "METE: meeting end-to-end QoS in multicores through system-wide resource management". In: *SIGMETRICS*. 2011.

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

[50] 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).

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

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

[53] M. Tokic. "Adaptive ε-Greedy Exploration in Reinforcement Learning Based on Value Differences". In: *KI*. 2010.

[54] 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).

[55] G. Welch and G. Bishop. *An Introduction to the Kalman Filter*. Tech. rep. TR 95-041. UNC Chapel Hill, Department of Computer Science.

[56] 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.

[57] 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.

[58] 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.

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

[60] H. Zhang and H. Hoffmann. "Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques". In: *ASPLOS*. 2016.

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

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

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