CALOREE: Learning Control for Predictable Latency and Low Energy

Anonymous Author(s)

Abstract

Many modern computing systems must provide reliable latency 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*—latency must be maintained despite unpredictable changes in operating environment or input. Machine learning accurately models the latency of complex, interacting resources, but does not address system dynamics; control theory adjusts to dynamic changes, but struggles with complex resource interaction. We therefore propose CALOREE, a resource manager that learns key control parameters to meet latency requirements with minimal energy in complex, dynamic environments. CALOREE breaks resource allocation into two sub-tasks: learning how interacting resources affect speedup, and controlling speedup to meet latency requirements with minimal energy. CALOREE defines a general control system—whose parameters are customized by a learning framework—while maintaining control-theoretic formal guarantees that the latency goal will be met. We test CALOREE's ability to deliver reliable latency 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 60% while reducing energy consumption by 13%.

1 Introduction

Large classes of computing systems—from embedded to servers—must deliver reliable latency 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 latency and energy tradeoffs. Software must allocate these resources to guarantee latency requirements are met with minimal energy.

There are two primary difficulties in efficiently allocating 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.*, changes in application workload or operating environment. Prior work addresses each of these difficulties individually.

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

Machine learning handles complex modern processors, modeling an application's latency and power as a function of resource configurations [5, 11, 13, 28, 46, 51, 52, 59, 76]. 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 models of the difference between measured and expected behavior [8, 22, 23, 27, 39, 57, 62, 66, 72, 74]. Control provides formal guarantees that it will meet the latency goal in dynamic environments, but these guarantees are based on ground-truth models relating resources and latency. If these models are not known or there is error between the modeled and actual behavior, the controller will fail to deliver the required latency.

Intuitively, combining learninged models of complex hardware resources with control-theoretic resource management should produce predictable latency in complex, dynamic systems. To derive the benefit of both, however, requires addressing two major challenges:

- Dividing resource allocation into sub-problems that suit learning and control's different strengths.
- Defining abstractions that efficiently combine sub-problem solutions, while maintaining control's formal guarantees.

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* latency and the relative speedup is independent of these changes. Learning is well-suited to modeling 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 latency. Once the learner has found Pareto-optimal tradeoffs the problem is convex and well-suited to adaptive control solutions which guarantee the required speedup even in dynamic environments. Figure 1 illustrates the intuition: processor complexity creates local optima, where control solutions can get stuck; but learning finds true optimal tradeoffs—"convexifying"—the problem, allowing control techniques to handle dynamics while providing globally optimal energy.

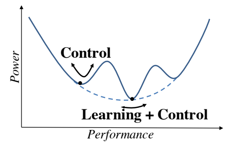


Figure 1. Learning smoothes the controller's domain.

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 model 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 guarantees even though the speedup is modeled by a noisy learning mechanism at runtime, rather than directly measured offline—as it would be in traditional control design.

Thus, we propose a general methodology where an abstract control system is customized at runtime by a learning mechanism to meet latency requirements with minimal energy. We refer to this approach as CALOREE[1]. Unlike previous work on control systems that required numerous user-specified models and parameters [8, 27, 39, 57, 74], CALOREE's learnner tunes the control parameters automatically; *i.e.*, *it requires no user-level inputs other than latency 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 [13, 59], collaborative filtering—*i.e.*, the Netflix algorithm[3, 11]—and a hierarchical Bayesian model [46]) and control (including proportional-integral-derivative [22] and adaptive, or self-tuning [38]) controllers. We set latency goals for benchmark applications and 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 enter the system and compete for resources. CALOREE achieves the:

- *Most reliable latency:*
  - In the *single-app* case, the best prior technique misses 10% of deadlines on average, while CALOREE misses only 6% on average. All other approaches miss 100% of deadlines for at least one application, but CALOREE misses 11% of deadlines in the worst case.
  - In the *multi-app* case, the best prior approach averages 40% deadline misses, but CALOREE misses just 20%.
- *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 18% more energy consumption than the oracle, but CALOREE consumes only 4% more.
  - In the *multi-app* case, the best prior approach averages 28% more energy than the oracle, while CALOREE consumes just 6% more.

In summary, *CALOREE is the first work to use learning to customize control systems at runtime, ensuring application latency—both formally and empirically—with no prior knowledge of the controlled application.* Its contributions are:

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

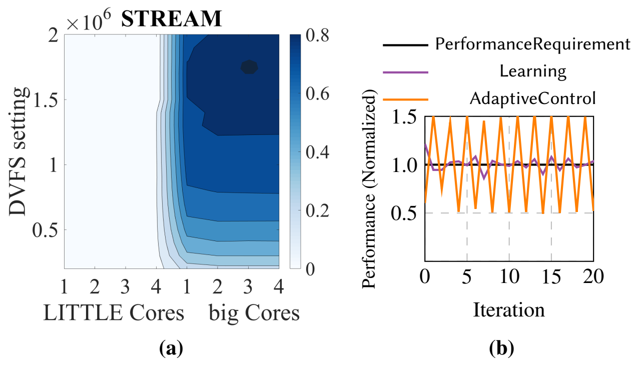


Figure 2. (a) STREAM performance vs. configuration. (b) Managing STREAM latency: *Learning* handles the complexity, but *control* oscillates.

- Separation of resource management into (1) *learning* complicated resource interactions and (2) *controlling* speedup.
- A generalized control design for use with multiple learners.
- A method for guaranteeing latency using learned—rather than measured—models.

2 Background and Motivation

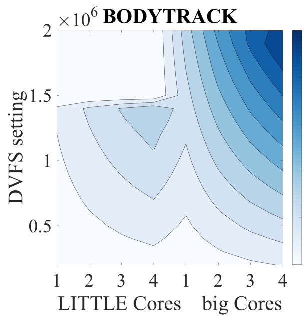
This section illustrates how learning handles complexity, how control handles dynamics, and then describes a key challenge that must be overcome to combine learning and control.

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 2a shows STREAM's performance for different resource configurations. STREAM has complicated behavior: the LITTLE cores' memory hierarchy cannot deliver performance. The big cores' more powerful memory system delivers greater performance, with a peak at 3 big cores. At low clockspeeds, 3 big cores cannot saturate the memory bandwidth, while at high clockspeeds thermal throttling creates performance loss. Thus, the peak speed occurs with 3 big cores at 1.2 GHz, and it is inefficient to use the LITTLE cores. STREAM, however, does not have distinct phases, so system dynamics are not an issue in this case.

Figure 2b shows 20 iterations of existing learning [46] and adaptive control [27] approaches allocating resources to STREAM. The x-axis shows iteration and the y-axis shows latency 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 latency. The *adaptive controller* begins with a generic notion of power/performance tradeoffs. As the controller runs, it measures latency and adjusts both the allocated resources and its own parameters. The adaptive controller dynamically adjusts to non-linearities with a series



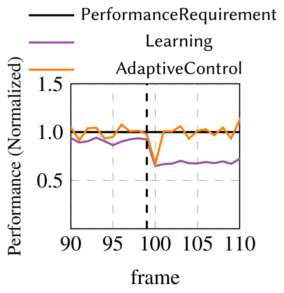


Figure 3. (a) bodytrack performance vs. configuration. (d) Managing bodytrack's latency with another application: *control* detects the change (at the vertical dashed line) and adjusts, but *learning* does not.

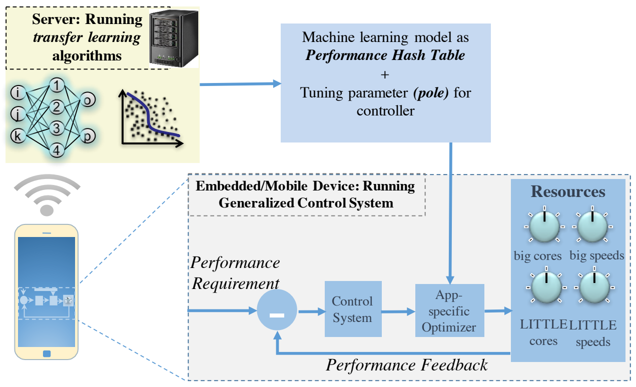

Server: Running *transfer learning* algorithms
Machine learning model as *Performance Hash Table* + Tuning parameter *(pole)* for controller
Embedded/Mobile Device: Running Generalized Control System
Resources
big cores
big speeds
LITTLE cores
LITTLE speeds
Performance Requirement
Control System
App-specific Optimizer
Performance Feedback

Figure 5. CALOREE overview.

of linear approximations; however, inaccuracies in the relationship between resources and latency cause oscillations that lead to latency violations. This behavior occurs because the controller's adaptive mechanisms cannot handle STREAM's complexity, a known limitation of adaptive control systems [14, 27, 74]. Hence, the *learner*'s ability to model complex behavior is crucial.

2.2 *Controlling* Dynamics

We now consider a dynamic environment. We begin with bodytrack running alone on the system. Figure 3a 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 latency requirements. Halfway through its execution, we launch a second application—STREAM—on a single big core, dynamically changing available resources.

Figure 3b shows the results. The vertical dashed line at frame 99 shows when the second application begins. At that point, the adaptive controller detects bodytrack's latency spike—rather than detecting the new application specifically—and it increases clockspeed and moves bodytrack from 4 to 3 big cores. The learner, however, does not have a mechanism to adapt to the altered environment. While we could theoretically add feedback to the learner and periodically re-estimate the configuration space, doing so is impractical due to high overhead for learners capable of handling this complexity [11, 12, 46]. Simpler reinforcement learners can adapt, but cannot guarantee reconvergence after the dynamic change [44, 64].

2.3 Challenges Combining Learning and Control

Sections 2.1 and 2.2 motivate splitting the resource allocation problem into modeling—handled by learning—and dynamic management—handled by control. This subsection demonstrates the importance of defining principled techniques for controlling systems using learned models.

The controller's *pole* is a particularly important parameter [38]. 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* [22, 41]. CALOREE, however, must tune the pole based the learner's models, 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 using a ground truth model mapping resource usage to performance, we model 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 [27].

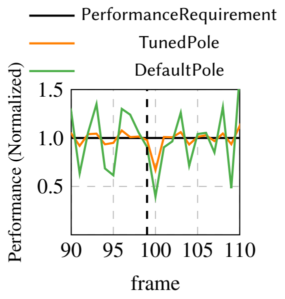
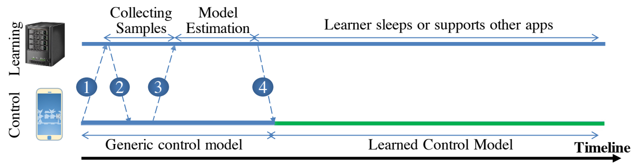


Figure 4. Comparison of carefully tuned and default poles.

As shown in Figure 4, the carefully tuned pole converges. The default pole, however, oscillates around the latency target, resulting in a number of missed deadlines. Additionally, the frames below the desired latency 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 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.*



1 An application starts and the controller begins with a generic model and queries the learner for the number of samples to take.

2 The learner responds with the number of samples needed, the controller continues.

3 The controller sends its samples back to the learner which asynchronously assembles a model.

4 The learner responds with a model customized for the application.

Figure 6. Temporal relationship of learning and control.

3 CALOREE: Learning Control

Figure 5 shows CALOREE's approach of splitting resource management into separate learning and control tasks and then composing these individual solutions. When a new application enters the system, an adaptive control system allocates resources using a generic model and records latency and power. The recorded values are sent to a learner, which predicts the application's latency and power in all other resource configurations. The learner extracts those that are predicted to be Pareto-optimal and packages them in a 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 latency. CALOREE's only user-specified parameter is the latency requirement.

Figure 6 illustrates the asynchronous interaction between CALOREE's learner and controller. The controller starts—using a conservative, generic speedup model—when a new application launches. The controller sends the learner the application's name and device type (message 1, Figure 6). 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 latency 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, it sends that data and the variance estimate to the controller (message 4), which uses the learned model from then on.

Figure 6 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.

We first describe adaptive control. We then generalize this approach, separating out parameters to be learned. Next, we discuss the class of learners 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.*, latency. 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]

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

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: latency. Thus, in 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 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 [38]. Such adaptive controllers provide formal guarantees that they will converge to the desired latency 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 [22, 57]. 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 latency [27, 38,

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

57, 74]. Therefore, typically 100s to 1000s of samples are taken at design time to ensure that the starting matrices are sufficient to ensure convergence [15, 41, 53].

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 latency) 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 latency 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 (latency), 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 latency. Using these substitutions, we eliminate $\mathbf{x}$ from Eqn. 1 to relate speedup to latency:

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

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 [67].

CALOREE must eliminate the error between the target latency and the goal: $error(t) = goal - 1/lat(t)$. Given Eqn. 2, CALOREE uses the integral control law [22]:

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

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 $0 \le \rho(t) < 1$. 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 big.LITTLE architecture an allocation includes big and LITTLE cores as well as a speed for both. 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:

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

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

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

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

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 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 latency constraint and observe that there must be an optimal solution with the following properties [31]:

- 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 speedup tradeoff space (*e.g.*, the top half of Figure 7) 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 7) provides constant time access to the lower convex hull. It consists of two arrays: the first being pointers into the second: a configuration array, which stores resource configurations the learner estimates to be 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. The first array of

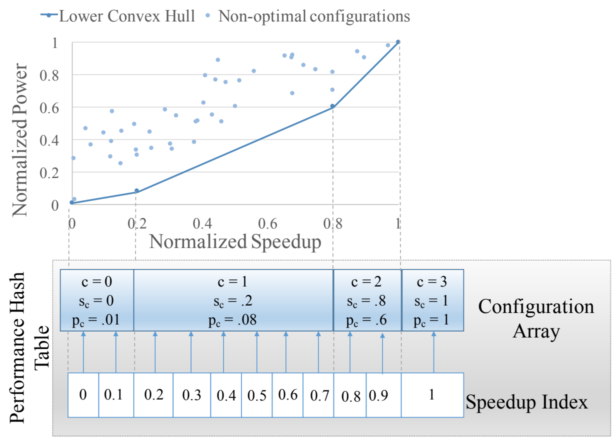


Figure 7. Data structure to efficiently convert required speedup into a resource configuration.

pointers has a *resolution* indicating how many decimal points of precision it captures and it is indexed by speedup. The example in Figure 7 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 7 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 σ^2. 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* combines information from the previously seen applications and current application to model the future behavior of the current application [50]. Transfer learning produces highly accurate models since it augments online data with offline information from other applications. CALOREE uses transfer learners because CALOREE's separation of learning and control makes it easy to incorporate data from other applications—the learner in Figure 6 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 [3]. This approach allows learners to borrow large amounts of data from other applications to answer questions about a new application. One formulation of this problem is to assume the matrix of resource-vs-speedup is low-rank and solve the problem using mathematical optimization techniques. The Netflix approach has been used to predict application response to heterogeneous resources in data centers [11, 12].

Bayesian Predictors: A hierarchical Bayesian model (HBM) provides a statistically sound framework for learning across applications and devices [19, 48]. 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 [46]. 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.

```
while $True$ do
    Measure application latency
    Compute required speedup (Equation (2))
    Lookup $s_{hi}$ and $s_{lo}$ with PHT
    Compute $\tau_{hi}$ and $\tau_{lo}$ (Equations 8 & 9)
    Configure to system to $hi$ & sleep $\tau_{hi}$.
    Configure to $lo$ & sleep $\tau_{lo}$.
end while
```

Algorithm 1: CALOREE's runtime control algorithm.

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 latency. Specifically, we prove that with probability 99.7% CALOREE converges to the desired latency 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 confidence that CALOREE converges to the desired latency. This pole value only considers latency, 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 [65] in quantifying uncertainty as $\beta(t)$, and setting the pole based on this uncertainty:

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

where $\bar{s}$ and $\bar{p}$ are the measured values of speedup and powerup and $\hat{s}$ and $\hat{p}$ are the estimated values from the learner. This measure of uncertainty captures both power and latency. 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.

4 Experimental Setup

4.1 Platform and Benchmarks

We run applications on an ODROID-XU3 with a Samsung Exynos 5 Octa processor (an ARM big.LITTLE architecture), running Ubuntu 14.04. The 4 big cores support 19 clock-speeds, the 4 LITTLE ones have 13. An on-board power meter updated at 0.25s intervals captures core, GPU, network, and memory. We allocate cores using thread affinity and set speeds using `cpufrequtils`. The ODROID has no screen, but recent trends in mobile/embedded processor design and workloads have seen processor power become the dominant factor in energy consumption [21]. We run the learners on an Intel server with E5-2690 processors. The ODROID and the server are connected with Gigabit Ethernet.

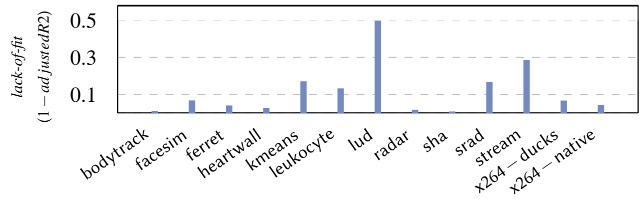


Figure 8. *Lack-of-fit* for performance vs clock-speed. Lower lack-of-fit indicates a more compute-bound application, higher values indicate a memory-bound one.

We use 12 benchmarks representing embedded and mobile sensor processing. These include video encoding (`x264`)), video analysis (`bodytrack`), image similarity search (`ferret`), and animation (`facesim`) from PARSEC [4]; medical imaging (`heartwall`, `leukocyte`), image processing (`srad`), and machine learning (`kmeans`) from Rodinia [7]; security (`sha`) from ParMiBench [29]; memory intensive processing (`stream`) [45]; and synthetic aperture radar (`radar`) [24].

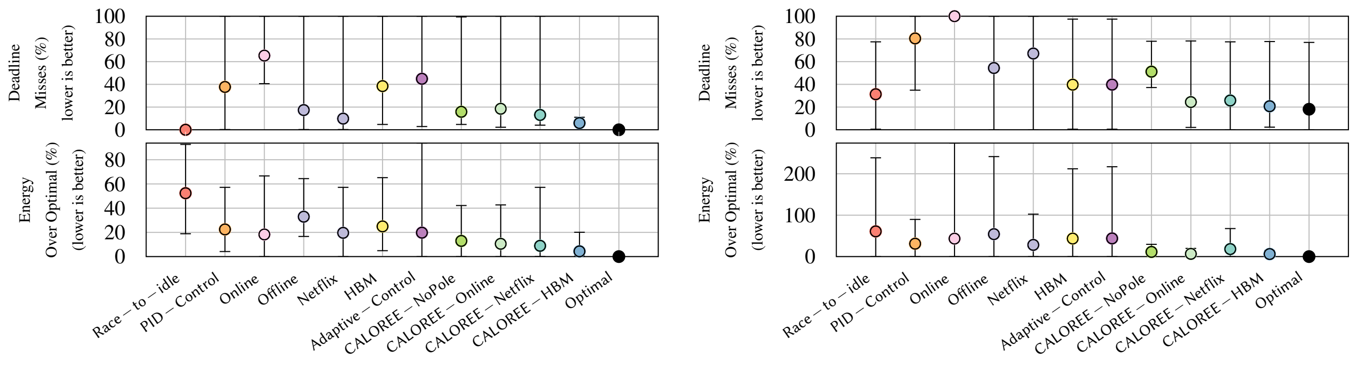
Figure 8 shows the variety of workloads indicated by the *lack-of-fit*—the absence of correlation between frequency and performance. Applications with high lack-of-fit do not speed up with increasing frequency—typical of memory bound applications. Applications with low lack-of-fit increase performance with increasing clock speed. 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 input (*e.g.*, a point for `kmeans` or a frame for `x264`). The application signals the completion of an input using a standard API [25]. Performance requirements are specified as latencies for these inputs.

4.2 Evaluation Metrics

For each application, we measure its worst-case execution time (wcet) running without management; *i.e.*, the highest latency for any input. We set a latency goal—or *deadline*—for each input equal to its wcet; the standard approach for ensuring real-time latency guarantees or maximum responsiveness [6]. We quantify performance reliability by measuring the missed deadlines. If the application processes n total inputs and m exceeded the target latency the deadline misses are:

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

We evaluate energy savings by running every application in every resource configuration and recording performance and power for every input. By post-processing this data we determine the minimal energy resource configuration that meets



(a) Single-App

(b) Mulit-App

Figure 9. Summary data for (a) single- and (b) multi-app scenarios. The top row shows deadline misses, the bottom energy consumption.

the latency for each input. To compare across applications, we normalize energy:

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

where $e_{measured}$ is measured energy and $e_{optimal}$ is the optimal energy. 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 input as fast as possible, then idles until the next input is available [31, 33, 47]. This heuristic is a standard way to meet hard deadlines, but it requires conservative resource allocation [6].
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 [22, 57]. 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 [40, 46, 52].
4. *Offline*: does not observe the current application—instead using previously observed applications to estimate power and performance as a linear regression [34, 36, 73, 76].
5. *Netflix*: a matrix completion algorithm for the Netflix challenge. Variations of this approach allocate heterogeneous resources in data centers [11, 12].
6. *HBM*: a hierarchical Bayesian learner previously used to allocate resources to meet performance goals with minimal energy in server systems [46].
7. *Adaptive-Control*: a state-of-the-art, adaptive controller that meets application performance with minimal energy [27]. This approach requires a user-specified model relating resource configuration to performance and power. For this paper, we use the *Offline* learner's model.

We compare the above baselines to:

1. *CALOREE-NoPole*: uses the HBM learner, but sets the pole to 0, which shows 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.

We use leave-one-out cross validation. To test application x, we train the learners on all other applications, then test on x.

5 Experimental Evaluation

5.1 Performance and Energy for Single App

Figure 9a summarizes the average error across all targets for the single application scenario. The figure shows deadline misses in the top chart and energy over optimal in the bottom. The dots show the average, while the error bars show the minimum and maximum values.

Race-to-idle meets all deadlines, but its conservative resource allocation has the highest average energy consumption. Among the prior approaches HBM has the lowest average deadline misses (9%) and lowest energy (20% more than optimal). CALOREE with no pole misses 15% of all deadlines, which is worse than prior approaches. Note that all prior approaches—other than racing—have at least one application that misses all deadlines. In many cases these approaches are close to the latency (within 10%), but not close enough to deliver reliable performance.

When CALOREE adaptively tune its pole, the results greatly improve. The best combination is CALOREE-HBM, which averages 6.0% missed deadlines, while consuming just 4.3% more energy than optimal. Thus, CALOREE-HBM reduces average deadline misses by 65% and energy consumption by 13% compared to the best prior approach. The error bars on the CALOREE-HBM approach demonstrate that it is the only approach—besides racing—that handles every test application; all others see at least 100% deadline

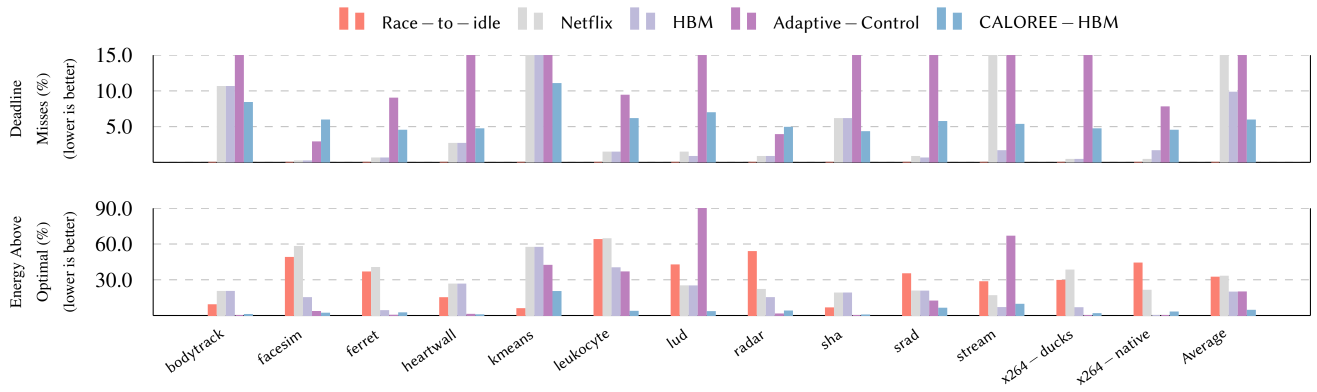


Figure 10. Comparison of application performance error and energy for single application scenario.

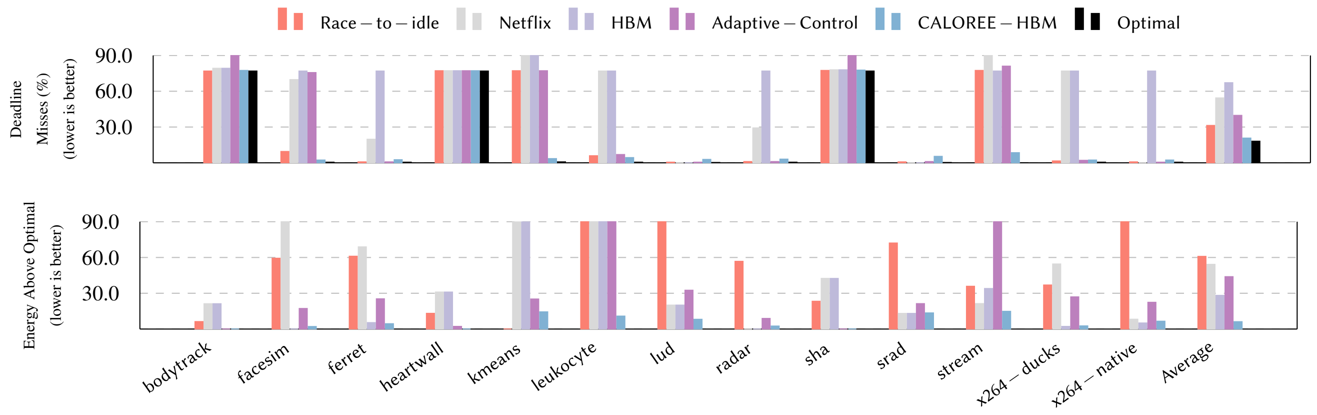


Figure 11. Comparison of application performance error and energy for multiple application scenario.

misses for one test case. Yet, CALOREE-HBM reduces energy consumption by 27% compared to race-to-idle. The energy savings comes because most inputs are not worst case, leaving slack for smart resource allocators to save energy. *Among many smart approaches CALOREE-HBM provides highly reliable performance with very low energy.*

Figure 10 presents a detailed, per-application comparison between CALOREE-HBM and selected prior approaches which have performed well in other scenarios: race-to-idle, Netflix, HBM, and adaptive control. 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 goal of meeting its worst case latency. Halfway through execution, we start another application randomly drawn from our benchmark set—bound to one big core—which interferes with the original application. Delivering the required latency tests the ability to react to environmental changes.

Figure 9b shows the average number of deadline misses and energy over optimal for all approaches. Some targets are unachievable for some applications; specifically, `bodytrack`, `heartwall`, and `sha`. 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. Most approaches do badly in this scenario—even adaptive control has 40% deadline misses. CALOREE-HBM produces the lowest deadline misses with an average of 20%, which is only 2 points more than optimal. It also produces the lowest energy, just 6% more than optimal. Figure 11 shows the detailed results.

5.3 Adapting to Phase Changes

We compare CALOREE and Adaptive-Control reacting to input variations. Figure 12 shows the `x264` video encoder with 2 different phases caused by a scene change 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

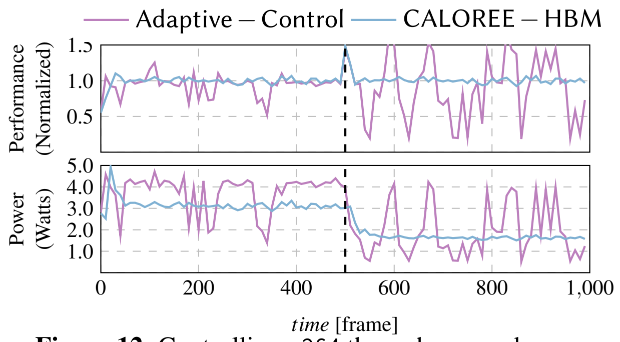


Figure 12. Controlling `x264` through scene changes.

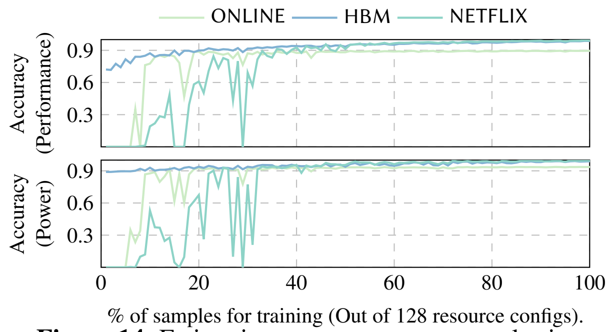


Figure 14. Estimation accuracy versus sample size.

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

Section 3.5 argues that tuning the controller to learned variance prevents oscillation and provides probabilistic guarantees despite using noisy, learned data to control unseen applications. We demonstrate this empirically by showing LUD using both CALOREE-NoPole and CALOREE-HBM. Figure 13 shows time on the x-axis and normalized performance and power on the y-axes. CALOREE-NoPole oscillates and causes wide power fluctuations. In contrast, CALOREE provides reliable performance and saves tremendous energy because it does not oscillate but uses a mixture of big and LITTLE cores to keep energy near minimal.

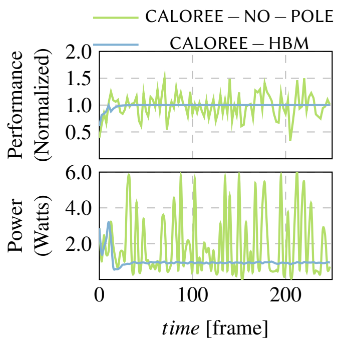


Figure 13. Comparison of learned and default poles.

5.5 Sensitivity to the Measured Samples

We show how the number of samples affects model 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 models the real performance or power. Accuracy matters because the fewer the samples, the faster the controller switches to the learner's application-specific model.

Figure 14 shows the accuracy vs sample count for both performance (top) and power (bottom). The HBM incorporates prior knowledge and its accuracy uniformly improves with more samples—exceeding 0.9 after 20 samples. The Online approach needs at least 7 samples to even generate a prediction. As Online receives more samples, its accuracy improves but never exceeds HBM's for the same sample count. 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 the best results.

5.6 Overhead

CALOREE's main overhead is sampling, where the controller tests 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 generates an estimate as fast as 500 ms, which is significantly smaller than the time required to run any of our applications. In the worst case (`facesim`), the controller sends 320B of sample data to the learner, which sends back 1KB. In this case, the sampling overhead and communication cost is less than 2% of total execution time. CALOREE's asynchronous communication means that the controller never waits for the learner. 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; the controller has negligible impact on performance and energy consumption of the controlled device.

6 Related Work

Energy has long been an important resource for mobile and embedded computing. Several OSs make energy an allocatable resource [56, 58, 59]. Others have specialized OS constructs to monitor [18] and reduce [16, 23, 37, 66, 72] energy for mobile and embedded applications. We examine related work applying learning and control to energy management.

Offline Learning approaches build predictors before deployment and then use those fixed predictors to allocate resources [2, 9, 34, 36, 71]. The training requires 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 reconfigure their device for energy savings [49]. 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, 35, 40, 51, 52, 61]. For example, Flicker is a configurable architecture and optimization framework that uses online prediction to maximize performance under a power limitation [51]. Another example, ParallelismDial, uses online adaptation to tailor parallelism to application workload [61].

Hybrid Approaches combine offline predictions with online adaptation [10, 13, 56, 59, 68, 70, 75]. For example, Dubach et al. use a hybrid scheme to optimize the microarchitecture of a single core [13]. Such predictors have also been employed at the operating system level to manage system energy consumption [56, 59, 70]. Other approaches combine offline prediction with online updates [5, 23, 28]. 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 to maximizes performance but without considering energy.

Control solutions can be thought of as a combination of offline prediction with online adaptation. Their formal properties make them attractive for managing resources in operating systems [22, 30, 62]. The offline phase involves substantial empirical measurement that is used to synthesize a control system [8, 27, 39, 54, 55, 60, 69, 72, 74]. Control solutions work well over a narrow range of applications, as the rigorous offline measurement captures the general behavior of a class of application and require negligible online overhead. This focused approach is extremely effective for multimedia applications [16, 17, 32, 43, 66, 72] and web-servers [26, 42, 63] because the workloads can be characterized ahead of time to produce sound control.

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. Prior work addresses the need for accurate predictions in various ways. Some provides control libraries that require user-specified models [20, 27, 55, 60, 74]. Others automatically synthesize both a predictor and a controller for either hardware [53] or software [14, 15]. JouleGuard combines learning for energy efficiency with control for managing application parameters [23]. In JouleGuard, a learner adapts the controller's coefficients to uncertainty, but JouleGuard does not produce a new set of predictions. JouleGuard's computationally efficient learner runs on the same device as the controlled application, but 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

Much recent work builds systems to support learning, CALOREE uses learning to build better systems. CALOREE is a resource manager that meets application latency requirements with minimal energy, even without prior knowledge of the application. CALOREE is the first work that provides formal guarantees that it will converge to the required latency despite not having prior knowledge. CALOREE achieves this breakthrough by using learning to model complex resource interaction and control theory to manage system dynamics. CALOREE proposes foundational techniques that allow control to be applied using noisy learned models—instead of ground truth models—while maintaining formal guarantees. We demonstrate CALOREE's effectiveness with a case study using embedded applications on a heterogeneous processor. Compared to prior learning and control approaches, CALOREE is the only approach that provides reliable latency for all applications with near minimal energy.

A Probabilistic Convergence Guarantees

Theorem A.1. *Let* $\mathbf{s_c}$ *and* $\hat{\mathbf{s_c}}$ *denote the true and estimated speedups of various configurations in set C as* $\mathbf{c} \in \mathbb{R}^{|C|}$*. Let* σ *denote the estimation error for speedups such that,* $\hat{s}_i \sim N(s_i, \sigma^2)\ \forall\ i$*. We 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 $\widehat{s_c}\Delta = s_c$. To guarantee convergence the value of pole, $\rho(t)$ can vary in the range $\lfloor 1 - \frac{2}{\Delta})\rfloor_0, 1)$ [14]. The lower $\rho(t)$, the faster the convergence. Equations 8 & 9 show that any $s(t)$ is a linear combination of two speedups:

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

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

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

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

The speedup estimates 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, $\hat{s}(t) \geq T(\hat{s}_{lo} - 3\sigma)$. 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] G. C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency: Predictability vs. Efficiency*. Springer, 2006.

[7] 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.

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

[9] 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.

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

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

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

[13] 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.

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

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

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

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

[18] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. "Quanto: Tracking Energy in Networked Embedded Systems". In: *OSDI*. 2008.

[19] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. CRC press, 2013.

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

[21] M. Halpern, Y. Zhu, and V. J. Reddi. "Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction". In: *HPCA*.

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

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

[24] H. Hoffmann, A. Agarwal, and S. Devadas. "Selecting Spatiotemporal Patterns for Development of Parallel Applications". In: *IEEE Trans. Parallel Distrib. Syst.* 23.10 (2012).

[25] 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.

[26] 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).

[27] 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.

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

[29] S. M. Z. Iqbal, Y. Liang, and H. Grahn. "ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems". In: *IEEE Comput. Archit. Lett.* 9.2 (July 2010).

[30] C. Karamanolis, M. Karlsson, and X. Zhu. "Designing controllable computer systems". In: *HotOS*. Berkeley, CA, USA, 2005.

[31] 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.

[32] 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).

[33] 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.

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

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

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

[37] M. Lentz, J. Litton, and B. Bhattacharjee. "Drowsy Power Management". In: *SOSP*. 2015.

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

[39] 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).

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

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

[42] 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.

[43] 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)* ().

[44] M. Maggio, H. Hoffmann, A. V. Papadopoulos, J. Panerati, M. D. Santambrogio, A. Agarwal, and A. Leva. "Comparison of Decision-Making Strategies for Self-Optimization in Autonomic Computing Systems". In: *ACM Trans. Auton. Adapt. Syst.* 7.4 (Dec. 2012), 36:1–36:32. ISSN: 1556-4665. DOI: 10.1145/2382570.2382572. URL: http://doi.acm.org/10.1145/2382570.2382572.

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

[46] 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.

[47] 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.

[48] C. N. Morris. "Parametric empirical Bayes inference: theory and applications". In: *Journal of the American Statistical Association* 78.381 (1983), pp. 47–55.

[49] 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.

[50] 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.

[51] 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.

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

[53] 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.

[54] 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.

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

[56] 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.

[57] 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.

[58] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. "Power Containers: An OS Facility

for Fine-grained Power and Energy Management on Multicore Servers". In: *SIGPLAN Not.* 48.4 (Mar. 2013), pp. 65–76. ISSN: 0362-1340. DOI: 10.1145/2499368.2451124. URL: http://doi.acm.org/10.1145/2499368.2451124.

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

[60] 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).

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

[62] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. "A Feedback-driven Proportion Allocator for Real-rate Scheduling". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. OSDI '99. New Orleans, Louisiana, USA: USENIX Association, 1999, pp. 145–158. ISBN: 1-880446-39-1. URL: http://dl.acm.org/citation.cfm?id=296806.296820.

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

[64] G. Tesauro. "Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies". In: *IEEE Internet Computing* 11 (1 2007).

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

[66] 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).

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

[68] 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.

[69] 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.

[70] 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.

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

[72] W. Yuan and K. Nahrstedt. "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems". In: *SOSP*. 2003.

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

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

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

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