CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems

Henry Hoffmann
Department of Computer Science, University of Chicago
hankhoffmann@cs.uchicago.edu

***Abstract*—We present CoAdapt, a runtime that coordinates accuracy-aware applications, which expose accuracy/performance tradeoffs, with power-aware systems, which manage power/performance tradeoffs. This is a challenging problem because decisions made at one level affect, and often counteract, decisions made at the other, leading to unpredictable behavior. Prior approaches have coordinated application and system, but provide a guarantee in only a single dimension (*i.e.,* just one of performance, power, or accuracy). In contrast, CoAdapt addresses the full performance/power/accuracy problem space with an adaptive runtime control system that guarantees behavior in any two of these dimensions, while optimizing the third. We implement CoAdapt on a Linux/x86 platform and show it predictably stabilizes to the desired behavior, adapts to phases in application workload, and supports dynamic changes to behavioral goals.**

I. Introduction

Predictable behavior is essential for the correct operation of many computer systems from the embedded space to webservers. Recently, techniques have emerged to create *accuracy-aware* applications, which automatically reduce accuracy (and computational requirements) to maintain performance despite fluctuations in system resource availability [2, 3, 7, 19, 37, 41]. Similarly, *power-aware* computing systems (including hardware and software) dynamically alter power consumption to maintain performance in the face of changing application workloads [4, 6, 11, 20, 21, 26, 27, 30, 44, 47, 50].

Accuracy-aware applications and power-aware systems adapt to provide predictable performance despite unpredictable changes in workload or resources. When deployed concurrently, however, they may interfere with each other, resulting in missed goals and unpredictable behavior. These problems arise because both application and system assume that changes in the other (*i.e.,* resources or workload) are rare, and will be long-lasting when they do occur. When application and system continuously react to each other, they miss performance goals, waste power, and lose accuracy.

Clearly, we must coordinate application and system to ensure predictable adaptation so that performance, power, or accuracy requirements are not violated. Several approaches have been proposed for managing such *cross-layer* adaptation [12, 23, 25, 28, 29, 33, 43]. This prior work coordinates application behavior with system resource usage to avoid potentially destructive interference. However, these approaches focus only on providing predictable behavior in a single dimension (typically performance). For example, GRACE-2 provides performance guarantees while minimizing energy, but does not explicitly guarantee accuracy [43]. However, many systems need guaranteed behavior in multiple dimensions, or even the flexibility to change what guarantees are provided in response to changing environmental conditions (*e.g.,* switching from wall power to battery power).

A. Cross-layer Adaptation with CoAdapt

This paper proposes a new approach to cross-layer adaptation. Specifically, we develop **CoAdapt**, a *runtime framework that dynamically coordinates accuracy-aware applications and power-aware systems*. Unlike prior approaches, CoAdapt's novel runtime control provides guarantees in any two out of the three dimensions of performance, power, and accuracy, while optimizing the third. CoAdapt avoids oscillation by selecting a *lead* and *subordinate* dimension of control based on the desired behavior (or *goals*) and the available adaptations. CoAdapt applies control in the lead dimension, and then dynamically customizes the control system for the subordinate dimension. This customization compensates for any potential interference, ensuring stable, predictable behavior in the two target dimensions. In addition, CoAdapt supports dynamic changes to both the magnitude and dimension of goals allowing the system to adapt to changing user desires (*e.g.,* switching from high-performance, high-accuracy to high-performance, low-power when battery life is low).

B. Summary of Results

We implement CoAdapt on a Linux/x86 system and evaluate its ability to manage existing accuracy-aware applications and power-aware systems. The applications include the x264 video encoder, the bodytrack video analysis application, the swaptions financial analysis application, and the swish++ search engine (all built with PowerDial [19]). The system platform supports adaptive resource management through allocation of cores to applications, injection of idle cycles, and management of dynamic voltage and frequency scaling (DVFS). Our results demonstrate:

- **Stability:** CoAdapt provides predictable behavior in *multiple* dimensions, quickly converging to externally specified goals and avoiding oscillation. (See Sec. V-B.)
- **Adaptation to Application Phases:** CoAdapt maintains performance during application workload fluctuations, while providing greater power savings than system-only adaptation and greater accuracy than application-only adaptation. (See Sec. V-C.)
- **Adaptation to Changing Goals:** We show CoAdapt dynamically transitioning between various goals (*e.g.,* high-performance, high-accuracy to high-performance, low-power). Supporting these transitions is a unique capability made possible by CoAdapt's support for multiple goal dimensions. (See Sec. V-D.)

C. Contributions

This paper makes the following contributions.

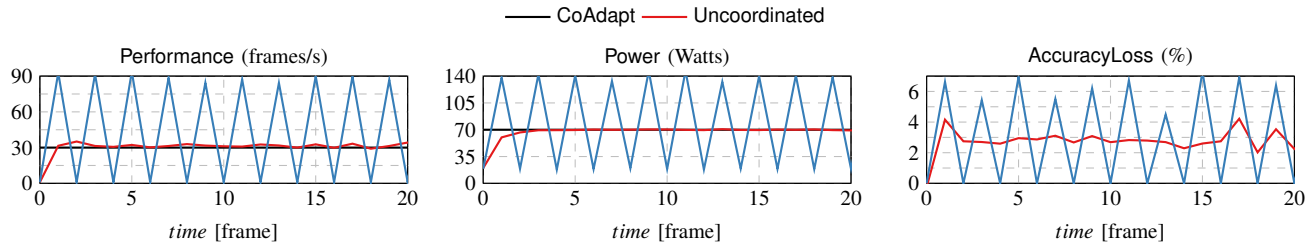

Fig. 1: Running an adaptive video encoder on an adaptive system with and without coordination. The goal is to maintain 30 frames per second performance. Without coordination, the performance goal is not met while power and accuracy fluctuate wildly. In contrast, CoAdapt's coordinated approach meets the performance goal while conserving power and accuracy.

- Demonstration of the oscillating behavior that can arise when accuracy-aware applications and power-aware systems act concurrently without coordination.
- Presentation of the CoAdapt runtime controller, which provides predictable behavior in multiple dimensions simultaneously. To the best of our knowledge, this is the first approach that guarantees dynamic behavior in any two of three dimensions of performance, power, and accuracy while optimizing the third.
- Evaluation of CoAdapt on a real system with several different applications.

The rest of this paper is organized as follows. Sec. II presents a motivational example demonstrating the need for CoAdapt. Sec. III describes CoAdapt's runtime control system. Sec. IV describes the applications and system used in this paper. Sec. V evaluates CoAdapt and compares it to other approaches. Sec. VI discusses related work in both application and system adaptation. Sec. VII concludes.

II. Motivational Example

This example demonstrates how uncoordinated adaptation of both system and application can fail to provide predictable performance. It then illustrates how CoAdapt overcomes this problem. This discussion provides an intuitive foundation for Sec. III's formal description of CoAdapt.

We run an accuracy-aware video encoder on a power-aware system. The encoder is built with *PowerDial* and automatically adjusts its algorithm to maintain soft real-time performance, with minimal accuracy loss; *i.e.,* increased bitrate [19]. The Sandy Bridge [38] Linux/x86 system runs an adaptive resource manager that allocates cores, clock-speed, and idle time to minimize energy while ensuring applications achieve predictable performance with minimal energy [20].

Both application and system collect performance feedback. If performance is low, the application switches to a simpler algorithm, while the system provides more resources. If performance is above the goal, the application increases accuracy, while the system reduces resource usage. In isolation, each approach is provably convergent to the performance goal [19, 20]. Unfortunately, these guarantees rely on assumptions about the relative rarity of changes; *i.e.,* systems assume application workload changes are infrequent, while applications assume system resource availability changes rarely. When each simultaneously reacts to the other, these assumptions are violated and performance can no longer be guaranteed.

Fig. 1 shows these problems. The application targets a performance of 30 frames per second and begins in its highest accuracy configuration, while the system initially allocates minimal resources. After encoding the first frame, both the application and system detect performance is close to 1 fps. Neither can achieve the goal individually, so the application switches to its least expensive algorithm and the system makes all of its resources available. These changes produce a performance of almost 90 fps, but both power consumption and accuracy loss are maximized. The application interprets the performance gain as the (long-term) availability of additional resources, while the system views it as a shift in application workload. Thus, the application reverts to its slowest (and most accurate) setting, while the system allocates minimal resources. In the third frame, performance is again below target, and the oscillations continue. As shown in Fig. 1, the uncoordinated adaptation of application and system fails to meet the performance requirement for every other frame. When the uncoordinated approach exceeds the performance goal, it wastes power and sacrifices accuracy.

CoAdapt overcomes these problems and, not only avoids oscillation in one dimension, but guarantees behavior in two of the three dimensions of performance, power, and accuracy. CoAdapt considers both the specified goals and available configurations of application and system. It then divides the goal dimensions into *lead* and *subordinate*. The lead dimension has fewer configurations affecting it, while the subordinate dimension has more. CoAdapt's control system selects a configuration that meets the goal in the lead dimension and predicts how that choice will affect the subordinate dimension. CoAdapt then dynamically adapts control for the subordinate dimension to ensure this goal is also met without oscillation.

Consider again the video encoder system with a real-time performance goal of 30 fps and a power budget of 70 Watts[1]. As before, the initial performance is just over 1 fps, while the power consumption is about 18 Watts. CoAdapt observes both values are below their goals and considers the configurations affecting each. Both application and system configurations affect performance, but only system configurations affect power. Thus, power is the lead dimension and performance is subordinate. CoAdapt selects the highest performance configuration within the 70 Watt budget, estimating it will provide a performance of 15 fps. CoAdapt then controls performance by selecting an application configuration that will further double the performance. CoAdapt slightly overestimates performance at first, but quickly corrects so performance and power settle to their target values (as shown in Fig. 1). In addition, accuracy is more stable with CoAdapt than without. In summary, CoAdapt

[1]We choose a point halfway between the maximum and minimum measured power consumption for our system.

coordinates application and system to meet the performance goal and avoid oscillations, while improving power consumption by over 10% and accuracy by over 20% compared to uncoordinated adaptation.

III. CoAdapt Framework

This section formally describes CoAdapt's cross-layer management of application and system. It begins by presenting terminology and notation. Then, it specifies how CoAdapt selects and controls the lead and subordinate dimensions. Finally, it discusses CoAdapt's support for changing goals at runtime, and how CoAdapt can use a combination of performance and power control to manage energy consumption.

CoAdapt is designed to provide predictability; *i.e.,* avoid the oscillating behavior shown in Fig. 1 and converge to the goals. A key design principle is generality. Rather than customizing a control system for each dimension, we design a generalized control system, capable of managing performance, power, and accuracy. The hope is that such generality will allow CoAdapt to control additional dimensions in the future.

A. Notation and Terminology

CoAdapt manages applications and system components supporting dynamic reconfiguration. Each component is represented by a *knob*, a discrete data structure supporting a finite number of *settings* represented as integers. The current setting of knob i is k_i. There are n knobs available for management, each may have a different number of settings. Each knob has a *baseline*, or nominal setting at $k_i = 0$.

The current *configuration* of the application and system is a vector $\vec{c} \in \mathbb{R}^n$, where the i^{th} element of $\vec{c}$ is the value k_i. We refer to the performance, power, and accuracy of a configuration as $s(\vec{c})$ (s for speed), $p(\vec{c})$, and $a(\vec{c})$.

To maintain generality with respect to applications, the runtime reasons about relative values rather than absolute values. Therefore, s, p, and a are measured as multipliers over the configuration where all knobs are set to their baseline, or nominal setting; *i.e.,* when $\vec{c} = \vec{0}$.

For example, suppose a system exposes one knob: processor speed. Further, two settings are supported: 1 and 2 GHz with power consumptions of 90 Watts at 1 GHz and 135 Watts at 2 GHz. This knob does not affect accuracy. In this example $\vec{c} \in \mathbb{R}$. $\vec{c} =< k_0 >$ represents the 1 GHz setting and $\vec{c} =< k_1 >$ represents the 2 GHz setting. Then $s(< k_0 >) = p(< k_0 >) = a(< k_0 >) = 1$. Further, $s(< k_1 >) = 2$, $p(< k_1 >) = 1.5$, and $a(< k_1 >) = 1$.

CoAdapt employs discrete time control. Therefore, it must reason about the value of variables at a given time. For any variable x, we denote its value at time t as $x(t)$.

B. Runtime Control

CoAdapt uses feedback control to guarantee behavior in up to two of the three dimensions of performance, power, and accuracy. Controlling multiple dimensions is a difficult problem because decisions made for one may affect the other. To account for this interference, CoAdapt selects *lead* and *subordinate* dimensions based on the available configurations of application and system. CoAdapt applies control in the lead dimension, predicts the affect that will have on the subordinate dimension, and uses this prediction to modify the control system for the subordinate dimension online.

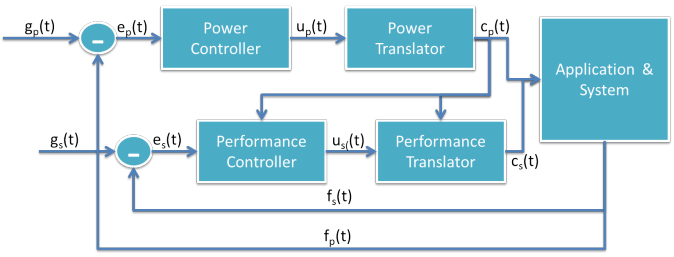

Fig. 2: Block diagram for controlling multiple goals.

Fig. 2 shows an example of CoAdapt's feedback control process when the lead dimension is power and the subordinate is performance. CoAdapt collects the current goals ($g_p(t)$, $g_s(t)$) and feedback ($f_p(t)$, $f_s(t)$) in the power and performance dimensions at time t. It then computes the *errors* (the difference between the goal and current value) $e_p(t)$ and $e_s(t)$. A control system produces $u_p(t)$, a signal indicating how to change power consumption to eliminate the error. This control signal is passed to a translator, which determines the highest performance configuration that achieves the control signal. The predicted performance of this configuration is used to modify a second control system that produces $u_s(t)$, a signal indicating how to drive performance error to zero. This signal is then translated into a configuration that achieves $u_s(t)$ with maximum accuracy *without* affecting power, thus avoiding oscillations. In the next iteration, feedback and goals are measured and new control signals are calculated. Through feedback, CoAdapt adapts to unpredictable dynamics such as changes in application workload or goals.

This section describes how CoAdapt selects lead and subordinate dimensions, turns error into control signals, and translates those signals into configurations of knob settings.

1) Selecting the Lead Dimension: Given goals in the set of dimensions $\{d_1, d_2\}$, where $d_i \in \{s, p, a\}$, the lead dimension is the one affected by fewer knob configurations. A configuration c affects dimension d, if $d(\vec{c}) \neq 1$; *i.e.,* if the configuration produces behavior other than nominal. Thus, the lead d_{lead} and subordinate d_{sub} dimensions are:

$$C_{d_1} = \{c | d_1(\vec{c}) \neq 1\} \quad (1)$$

$$C_{d_2} = \{c | d_2(\vec{c}) \neq 1\} \quad (2)$$

$$d_{lead} = \arg \min_{d \in \{d_1, d_2\}} |C_d| \quad (3)$$

$$d_{sub} = \arg \max_{d \in \{d_1, d_2\}} |C_d| \quad (4)$$

Eqns. 1 and 2 simply determine the set of knob configurations affecting the goal dimensions. Then, Eqn. 3 sets the lead dimension as the one with the smaller set of configurations, and the subordinate the one affected by more configurations.

2) Controlling the Lead Dimension: Let $g_{lead}(t)$ and $f_{lead}(t)$ be the goal and feedback in the lead dimension at time t. Then the runtime eliminates the error $e_{lead}(t) = g_{lead}(t) - f_{lead}(t)$ between these values by computing a control signal $u_{lead}(t)$ based on a prediction of the next feedback measurement:

$$f_{lead}(t+1) = b_{lead} \cdot u_{lead}(t) + dist_{lead}(t) \quad (5)$$

Where b is the baseline behavior in dimension *lead* (*i.e.,* behavior in the lead dimension with $k_i = 0, \forall i$) and $dist_{lead}(t)$ represents a transient disturbance in behavior. For example, if the lead dimension is performance, b_{lead} is the performance with all knobs at their default setting. $f_{lead}(t)$ represents

the measured performance at time t. $u_{lead}(t)$ represents the speedup (over baseline) the translator should achieve at time t, and $dist_{lead}(t)$ represents a momentary disruption in performance (*e.g.,* a page fault).

Given $e_{lead}(t)$ and the model of Eqn. 5, CoAdapt finds $u_{lead}(t)$ using the *integral control law* [17]:

$$u_{lead}(t) = u_{lead}(t-1) + \frac{e_{lead}(t)}{b_{lead}} \tag{6}$$

3) Translating the Lead Dimension: Having calculated $u_{lead}(t)$, CoAdapt must determine specific knob settings that will achieve this goal. Further, the knob settings must be drawn from the set C_{lead} (Eqns. 1–4). The control signal is a continuous variable that must be realized in a discrete configuration space. To approximate the continuous value, CoAdapt schedules configurations over a window of τ time units. τ is taken to be the time until the next feedback signal is available. The schedule consists of an amount of time to spend in each configuration such that the average behavior over the window is equal to the control signal. To minimize the affect on the subordinate dimension, this schedule should achieve $u_{lead}(t)$, while optimizing behavior in the subordinate dimension (*i.e.,* minimizing power consumption or maximizing accuracy and performance):

$$optimize \quad \sum_c \tau_c \cdot d_{sub}(\vec{c}) \tag{7}$$

$$s.t.$$

$$\sum_c \frac{\tau_c}{\tau} \cdot d_{lead}(\vec{c}) = u_{lead}(t) \tag{8}$$

$$\sum_c \tau_c = \tau \tag{9}$$

$$\tau_c \geq 0 \qquad \forall c \in C_{lead} \tag{10}$$

$$\tau_c = 0 \qquad \forall c \notin C_{lead} \tag{11}$$

This formulation assigns a time τ_c to each configuration, such that the behavior of the subordinate dimension is optimized (Eqn. 7), the control signal for the lead dimension is realized (Eqn. 8), and the total time does not exceed the time before the next feedback measurement (Eqn. 9).

It is impractical to solve Eqns. 7–11 online. Therefore, CoAdapt employs a heuristic solution ensuring the constraints (Eqns. 8–11) are met (guaranteeing behavior of the lead dimension), while providing close to optimal behavior for Eqn. 7 in practice. Specifically, CoAdapt uses two configurations: *over* and *under*. *over* exceeds the control signal in the lead dimension but has the smallest impact on the subordinate dimension. *under* is the most efficient state below the control signal. These two states and the corresponding values of τ_c are calculated as:

$$C_{over} = \{c \in C_{lead} | d_{lead}(\vec{c}) \geq u_{lead}(t)\} \tag{12}$$

$$over = \arg\min_{c \in C_{over}} d_{sub}(\vec{c}) \tag{13}$$

$$C_{under} = \{c \in C_{lead} | d_{lead}(\vec{c}) \leq u_{lead}(t)\} \tag{14}$$

$$under = \arg\max_{c \in C_{under}} d_{lead}(\vec{c}) / d_{sub}(\vec{c}) \tag{15}$$

$$\tau_{under} = \frac{u_{lead}(t) - d_{lead}(\vec{over})}{d_{lead}(\vec{under}) - d_{lead}(\vec{over})} \cdot \tau \tag{16}$$

$$\tau_{over} = \tau - \tau_{under} \tag{17}$$

$$\tau_c = 0 \qquad \forall c \notin \{under, over\} \tag{18}$$

A simple search over available configurations will find *over* and *under*, requiring $O(|C|)$ operations, where C is the set of all configurations. Alternatively, we can create a table of buckets for each dimension (essentially a hash table), where each bucket represents a range of behavior in that dimension. CoAdapt employs this approach for large C, confining the search to the small number of configurations that map to the same bucket, requiring $O(1)$ operations when the table is large enough to avoid bucket collisions.

4) Controlling the Subordinate Dimension: Once *over* and *under* have been collected CoAdapt estimates how they will affect the subordinate dimension by calculating $\hat{b}_{sub}(t)$:

$$\hat{b}_{sub}(t) = b_{sub} \cdot \frac{d_{sub}(\vec{over}) \cdot \tau_{over} + d_{sub}(\vec{under}) \cdot \tau_{under}}{\tau} \tag{19}$$

CoAdapt uses Eqn. 19 and the integral-control-law to calculate a control signal that eliminates error $e_{sub}(t) = g_{sub}(t) - f_{sub}(t)$ in the subordinate dimension:

$$u_{sub}(t+1) = u_{sub}(t) + \frac{e_{sub}(t)}{\hat{b}_{sub}(t)} \tag{20}$$

The value of $u_{sub}(t)$ is then passed to translation.

5) Translating the Subordinate Dimension: The value of $u_{sub}(t)$ must be translated into knob settings without affecting the behavior of d_{lead}, so CoAdapt computes the set of configurations affecting d_{sub}, but not d_{lead}, and uses those configurations to achieve $u_{sub}(t)$.

From Eqns. 1–4, CoAdapt has a set of configurations affecting the lead dimension, C_{lead}, and a possibly overlapping set of configurations affecting the subordinate dimension, C_{sub}. Furthermore, by construction, $|C_{lead}| \leq |C_{sub}|$. Therefore $C_{sub} \setminus C_{lead}$ is the set of all configurations that affect the subordinate dimension but do not affect the lead dimension. Let d_{free} be the dimension for which there is no goal. Then CoAdapt translates $u_{sub}(t)$ to knob configurations that optimize d_{free} by approximating a solution to:

$$optimize \quad \sum_c \tau_c \cdot d_{free}(\vec{c}) \tag{21}$$

$$s.t.$$

$$\sum_c \frac{\tau_c}{\tau} \cdot d_{sub}(\vec{c}) = u_{sub}(t) \tag{22}$$

$$\sum_c \tau_c = \tau \tag{23}$$

$$\tau_c \geq 1 \qquad \forall c \in C_{sub} \setminus C_{lead} \tag{24}$$

$$\tau_c = 0 \qquad \forall c \notin C_{sub} \setminus C_{lead} \tag{25}$$

This approximation follows the form of Eqns. 12–18:

$$C_{subover} = \{c \in C_{sub} \setminus C_{lead} | d_{sub}(\vec{c}) \geq u_{sub}(t)\} \tag{26}$$

$$subover = \arg\min_{c \in C_{subover}} d_{free}(\vec{c}) \tag{27}$$

$$C_{subunder} = \{c \in C_{sub} \setminus C_{lead} | d_{sub}(\vec{c}) \leq u_{sub}(t)\} \tag{28}$$

$$subunder = \arg\max_{c \in C_{subunder}} d_{sub}(\vec{c}) / d_{free}(\vec{c}) \tag{29}$$

$$\tau_{subunder} = \frac{u_{sub}(t) - d_{sub}(\vec{subover})}{d_{sub}(\vec{subunder}) - d_{sub}(\vec{subover})} \cdot \tau \tag{30}$$

$$\tau_{subover} = \tau - \tau_{subunder} \tag{31}$$

$$\tau_c = 0 \qquad \forall c \notin \{subunder, subover\} \tag{32}$$

Solving Eqns. 26–32 is analogous to solving Eqns. 12–18. Configurations *subunder* and *subover* do not affect the lead dimension, so CoAdapt schedules them independently of *over* and *under*.

6) Runtime Summary: Algorithm 1 shows the CoAdapt runtime. It runs in an infinite loop. At each iteration it receives feedback, selects the lead and subordinate dimensions, applies control, and configures the system. When the runtime is not computing new control or setting configurations, it puts itself to sleep to minimize its overhead.

CoAdapt's methodology is general with respect to the dimensions under control, and we hope this generality allows CoAdapt to be extended to control additional dimensions in the future. In addition, the methodology is stable because, by construction, control for the subordinate dimension does not affect the lead dimension. An analysis of CoAdapt's control system is provided in App. A. Finally, the proposed

methodology is close to optimal as it optimizes behavior in the unconstrained dimension using the heuristics described above.

C. Changing Goals at Runtime

CoAdapt allows applications to change goals dynamically. Both the magnitude (*e.g.,* changing from one performance goal to another) and the dimensionality (*e.g.,* changing from performance and accuracy goals to performance and power) can be changed. Allowing goal changes supports applications whose concerns change over their lifetime. For example, if a mobile user watching a video switches from wall power to battery power their primary concerns might switch from performance and accuracy to performance and power. To transition smoothly between goals, all state is maintained for all three dimensions regardless of the current goal. Specifically, the runtime maintains the value of $u_d(t)$ and $\hat{b}_d(t)$ for all d for the previous time step. This requires storing only six floating point values, so there is little overhead. Maintaining this state allows CoAdapt to rapidly transition to a new goal and still compute the correct value for $u_{lead}(t+1)$ and $u_{sub}(t+1)$, even when *lead* and *sub* change dynamically.

D. A Note on Energy Goals

For some systems *energy* is the primary goal as reducing energy will extend battery life or reduce costs; however, CoAdapt does not explicitly support energy goals. Energy consumption is a function of both performance and power consumption. CoAdapt, therefore, takes the approach energy control should be achieved by setting power and performance goals simultaneously. The authors believe that users will be better informed about whether it is preferable to increase performance or decrease power consumption (or some combination of both), but by supporting goal management in both power and performance, CoAdapt can meet the needs of users whose main concern is predictable energy consumption.

E. Impossible Goals

It is possible to specify goals that cannot be achieved with the available configurations of application and system. In this case, CoAdapt will not be able to eliminate the error between the feedback and the goal. To handle this situation for dimension d, CoAdapt checks whether the value of $u_d(t)$ is greater than the maximum value that can be achieved in that dimension. If so, CoAdapt sets the control signal to this maximum value. This check prevents infinite growth in the value of $u_d(t)$ and allows the system to default to best effort behavior when the goals are unachievable.

IV. Experimental Setup

We describe the applications and system used to evaluate CoAdapt.

A. Applications

We use four benchmark applications: x264, swaptions, and bodytrack (from the PARSEC benchmark suite [5]); as well as swish++, an open-source search engine [42]. All four benchmarks have been modified using the PowerDial compiler to become accuracy-aware [19]. PowerDial turns static configuration parameters into a data structure which can alter the dynamic behavior of the application. PowerDial's compiler also inserts an application-level runtime into the benchmarks that adjusts accuracy to guarantee performance. We disable the PowerDial runtime and replace it with CoAdapt.

Algorithm 1 The CoAdapt Runtime.

```
loop
    Read goal dimensions {d1, d2} and collect feedback.
    Determine d_lead and d_sub according to Eqns. 1–4.
    Compute u_lead(t) according to Eqn. 6.
    Translate u_lead(t) following Eqns. 12–18.
    Estimate b̂_sub(t) using Eqn. 19.
    Compute u_sub(t) with Eqn. 20.
    Translate u_sub(t) according to Eqns. 26–32.
    Set app and system to configurations c_over and c_subover.
    if τ_over < τ_subover then
        Sleep for τ_over time units.
        Set app and system to configurations c_under and c_subover.
        Sleep for τ_subover − τ_over time units.
        Set app and system to configurations c_under and c_subunder.
        Sleep for τ_subunder time units.
    else
        Sleep for τ_subover time units.
        Set app and system to configurations c_over and c_subunder.
        Sleep for τ_over − τ_subover time units.
        Set app and system to configurations c_under and c_subunder.
        Sleep for τ_under time units.
    end if
end loop
```

TABLE I: Accuracy-aware Application configurations.

Application	Configurations	Max Speedup	Max Acc. Loss (%)
x264	560	4.26	6.2
swaptions	100	100.35	1.5
bodytrack	200	7.38	14.4
swish++	6	1.52	83.4

We measure application accuracy loss or *distortion*, a unitless metric representing the percentage by which the result differs from a nominal setting [37]. Changes in performance are measured as speedup, the factor by which performance increases when moving from the nominal setting. This section briefly describes the tradeoffs exposed by each of these applications. Table I summarizes the application-level configurations, showing the total number of available configurations as well as the maximum speedup and accuracy loss. These applications represent a broad range of workloads that could be run in settings requiring predictable performance, power, or accuracy.

1) x264: This video encoder compresses a raw input according to the H.264 standard. The application can reduce the quality of the encoded video to increase the encoding frame rate. x264 searches for redundancy both within a frame and between frames. The accuracy-aware x264 exposes three separate knobs which trade the amount of work performed to find redundancy for the amount of redundancy identified. The total number of distinct application-level knob configurations is 560. The accuracy of the output video is measured by recording both the peak signal-to-noise ratio (PSNR) and the encoded bitrate, and weighting these two quantities equally.

2) swaptions: This financial analysis application uses Monte Carlo simulation to price a portfolio of swaptions. This application can reduce accuracy in the swaption price for increasing the rate of pricing. The application has a single knob, with 100 settings, controlling the number of Monte Carlo simulations performed per swaption.

3) bodytrack: This application uses an annealed particle filter to track a human moving through a video scene. The filter parameters control a tradeoff between the quality of the track and the tracking frame rate. The application exposes two

TABLE II: System configurations.

Configuration	Settings	Max Speedup	Max Powerup
clock speed	12	3.08	3.88
core usage	12	10.16	3.17
idle	n/a	1.0	1.0

knobs, one which changes the number of annealing layers and another which changes the number of particles used. Together, these knobs support 200 different configurations.

4) swish: This application is a search engine which indexes and searches files on web sites. Given an input query, it searches its index to find matching documents, ranks the matches, and returns the document list in ranked order. The application exposes a tradeoff in the number of documents returned per request and the time taken to serve the request. This tradeoff is represented as a single knob with six separate settings. The accuracy of the search engine is evaluated using F-measure, the harmonic mean of precision and recall[2].

B. System

Our hardware platform is a six-core Intel Xeon E5-1650 processor running Linux 3.2.0. The processor supports hyper-threading and has twelve different processor speeds including TurboBoost. This is a Sandy Bridge processor, which allows power and energy consumption to be read directly from registers at runtime in millisecond intervals [38]. This system supports three adaptations that alter performance and power consumption tradeoffs. The first uses the `cpufrequtils` package to control clock frequency (the highest setting actually turns control over to the hardware by enabling TurboBoost). Higher clock frequencies increase performance at the cost of increased power consumption. The second adaptation uses thread affinity to reduce the number of cores actively performing computation. The processor supports power-gating, so reducing core usage will reduce both power consumption and performance. The final adaptation is to idle the processor to take advantage of the low-power idle state. This adaptation supports *racing to idle*, where the system attempts to complete work as quickly as possible and maximize idle time [18]. Idling will not increase power consumption, but can decrease it for a decrease in performance.

These system knobs are summarized in Table II, which shows the total number of available settings and the maximum increase in speed and power measured on the machine for any benchmark. There are effectively an unlimited number of idle settings, as any application could be stalled arbitrarily. App. B contains a brief description of our implementation of CoAdapt on the target system.

V. Evaluation

We present several case studies evaluating CoAdapt managing the applications and system described in Sec. IV. We begin with a discussion of overhead. Sec. V-B shows that CoAdapt stabilizes to the goals while avoiding oscillations. Sec. V-C demonstrates CoAdapt coordinating system and application response to changes in application workload. Sec. V-D studies a unique capability of CoAdapt: reacting to changes in the goals themselves.

[2]Precision is the number of returned documents relevant to a query divided by the total number of returned documents. Recall is the number of relevant documents returned divided by the total number of relevant documents in the index

A. Overhead of CoAdapt

We measure CoAdapt's overhead by running the benchmarks both without the runtime and with the control system performing all calculations but not actually setting the knobs. Thus, any changes in performance or power consumption are due to its overhead. We find the power and performance overhead of CoAdapt is less than 1% for all applications.

Additionally, we account for overhead by measuring the maximum speed at which the runtime can execute the loop in Algorithm 1 when not waiting for feedback. We collect traces of feedback on the system and feed it through the runtime and measure the rate at which it selects new configurations for both the lead and subordinate dimensions. For x264 (with over 80,000 possible configurations), CoAdapt computes 48.6 million iterations of Algorithm 1 on our test platform. This is significantly faster than the rate at which any of the applications report feedback, verifying the runtime is low overhead. All results include the power and performance impact of the runtime itself.

B. Meeting Goals in Multiple Dimensions

This study demonstrates CoAdapt's ability to meet goals in multiple dimensions simultaneously while avoiding oscillations. This case study extends the motivational example from Sec. II to the three other benchmark applications. We compare CoAdapt to two other approaches: 1) uncoordinated application and system adaptation, using PowerDial [19] and PTRADE [20] and 2) a cross-layer approach based on the GRACE-2 system, which guarantees performance while minimizing energy, but does not directly monitor accuracy [43]. Applications are assigned performance goals and all three approaches attempt to guarantee predictable performance. In addition, CoAdapt targets a 70 Watt power consumption while minimizing accuracy loss.

The results for x264 are shown in Fig. 1. Results for swaptions, bodytrack, and swish are shown in Fig. 3. Each column of charts corresponds to an application. The first row shows performance (normalized to the goal), the second row shows power consumption, and the third row shows accuracy loss. Solid black lines show the desired behavior for performance and power. These results confirm the problem illustrated in the motivational example. Uncoordinated adaptation leads to oscillating behavior in three out the four benchmarks. This oscillation results in either missed deadlines or wasted power and accuracy loss. GRACE-2 can meet the target performance, but does so while minimizing energy – resulting in power consumptions below the target and increased accuracy loss. In contrast, CoAdapt converges to the desired behavior for both performance and power. In general, it may take an iteration or two to settle (due to non-linearities in the interaction of application and system); but after settling, behavior with CoAdapt quickly tracks the desired goals for all benchmarks.

swish++ represents an interesting exception where uncoordinated control does not lead to oscillation. This appears to be due to the noise in the application performance, which results in both application and system acting conservatively and converging to the desired performance. However, both uncoordinated adaptation and GRACE-2 result in lower than desired power consumption and high accuracy loss. In contrast, CoAdapt meets the performance goal while running at the desired power consumption, saving accuracy.

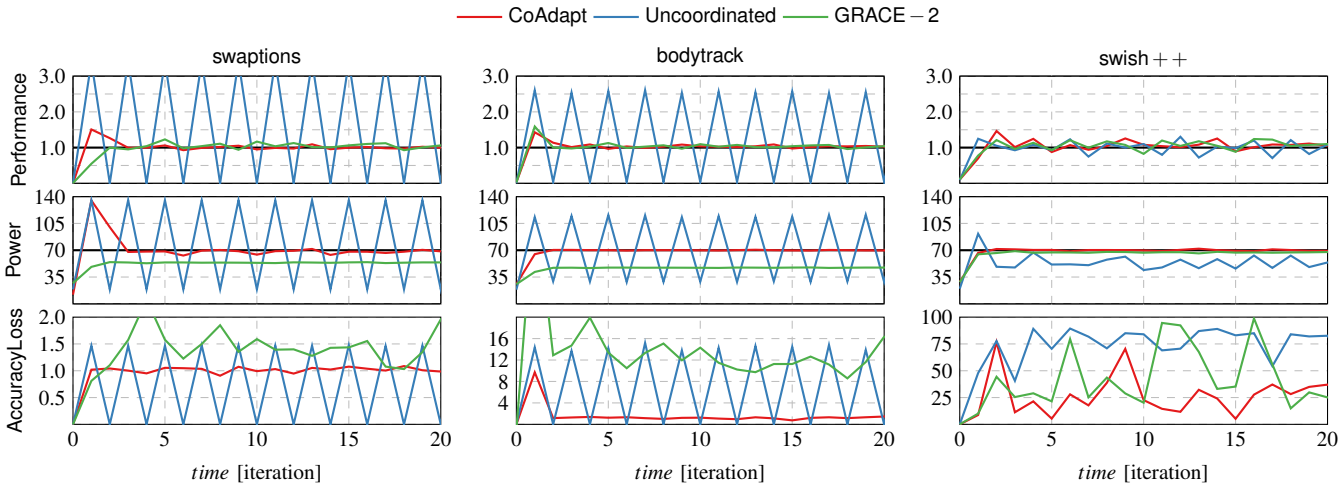

Fig. 3: CoAdapt prevents oscillations, while uncoordinated adaptation of application and system does not.

These results show CoAdapt managing the power and performance dimensions. Results controlling accuracy and performance have been omitted for space, but follow similar trends: 1) uncoordinated control leads to oscillation for all benchmarks but swish, 2) GRACE-2 meets the performance goal while exceeding the desired accuracy loss, 3) and CoAdapt quickly converges to the desired performance and accuracy.

C. Adapting to Application Phases

In this case study, we create a new input for the x264 benchmark by concatenating three different video scenes to form a single input with three distinct phases[3]. The top row of charts in Fig. 4 illustrates these phases using the default behavior of application and system (*i.e.,* encoding with the highest accuracy and all system resources). All figures show time on the x-axis; the upper row shows performance, power, and accuracy (measured in bitrate, lower is better) as a function of time. The three different phases (each 500 frames long) are demarcated by the vertical dashed lines. Each phase (or scene) has distinct characteristics. The first is the slowest, has lowest average power consumption and produces the largest bitrate. The second scene is faster and naturally has lower bitrate. The third scene is the fastest, but occupies a middle ground in bitrate.

To show the benefits of adaptation, we set a soft real-time performance goal (of 30 frames per second) and deploy the encoder with PTRADE, the PowerDial runtime, and with CoAdapt. We give CoAdapt an additional goal of maintaining an accuracy within 10% of the bitrate for the third scene, which precludes any accuracy loss in the first scene.

The behavior of all three management systems on this input is illustrated in the bottom row of Fig. 4. Clearly all three approaches meet the performance goal (with some short, deviations appearing at the scene changes); however, each approach achieves the goal differently, resulting in different power consumptions and accuracies. During easier scenes, PowerDial reduces accuracy, using the performance increase to increase idle time. PTRADE alters resource usage to save more power without sacrificing accuracy (about 75% of PowerDial's consumption). However, CoAdapt produces the lowest power consumption of the three, achieving almost half the power consumption of PowerDial's application-only approach. Furthermore, CoAdapt not only saves power, it also produces slightly better accuracy than PowerDial. Had we configured CoAdapt with a goal of zero accuracy loss, it would have produced the same behavior as PTRADE — highlighting the wider range of behaviors achievable with CoAdapt.

[3]These videos are the high-definition, 1080p videos `ducks_take_off`, `rush_hour`, and `old_town_cross` taken from xiph.org.

D. Reacting to Changing Goals

Our final experiment explores reaction to changing goals, which may occur due to changes in user priorities. For example, a mobile user's preference between performance, power consumption and accuracy might change over the life of long lived applications as battery life is diminished. Prior approaches, like PowerDial, PTRADE, and GRACE-2, support goals in only a single dimension, so this case study demonstrates a unique feature of CoAdapt.

We deploy each application with an initial set of goals, and then change those goals part way through execution. Fig. 5 shows the results of this experiment. There is a column for each application. The first row shows performance normalized to the initial performance goal. The second row shows power consumption. The third row shows accuracy loss. Solid horizontal black lines show the goals. Dashed vertical lines show the time that the goals change. The red line shows the behavior of the application and system managed by CoAdapt. We describe each application individually below.

1) x264: Initially, the goal is to maintain real-time performance (represented by 1.0 in the chart) and maximize accuracy subject to that performance goal. After 100 frames we simulate the switch from wall power to battery life by setting new goals that maintain real-time performance but increase energy efficiency by 1.5×. Thus, the initial goals are high accuracy and real-time performance, and they switch to the same performance with $1/3$ less power consumption.

CoAdapt begins with high power and high accuracy. At frame 100, the goals change. Power consumption immediately drops to the desired value. Performance quickly returns to real-time, and a small amount of accuracy is occasionally sacrificed.

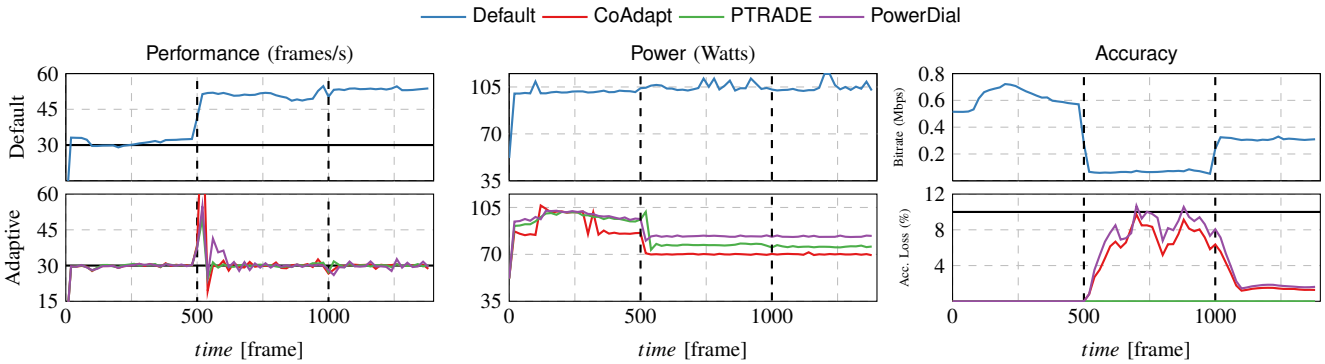

Fig. 4: Comparison of non-adaptive behavior and several adaptive schemes adjusting to phases in x264.

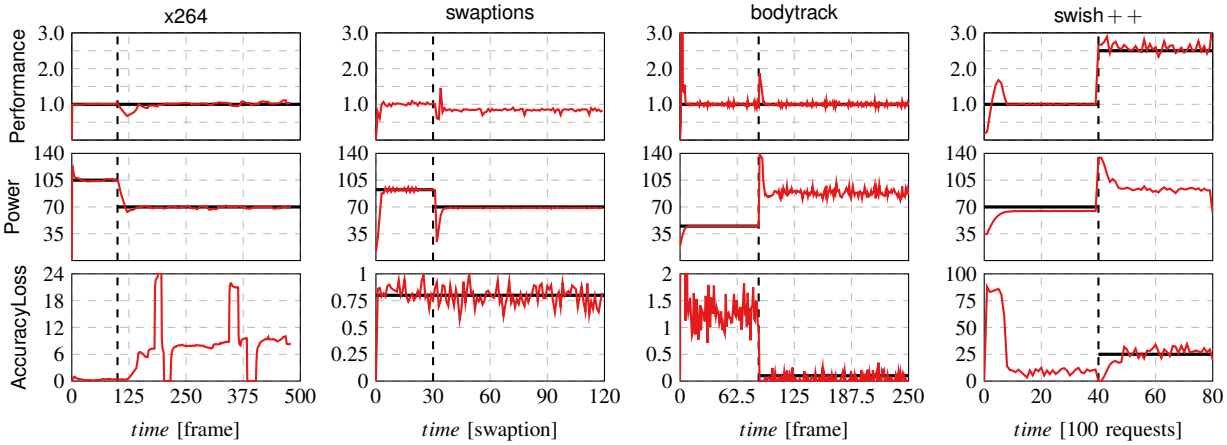

Fig. 5: CoAdapt provides stability while managing applications through goal changes.

2) swaptions: We deploy the swaptions benchmark with a power goal of 90 Watts and an accuracy goal of 0.8% accuracy loss. After 32 swaptions, we set a new power goal of 70 Watts, but keep the accuracy goal the same. This represents a software imposed power cap, but an unwillingness to further distort the output.

CoAdapt begins within the accuracy and power limits. When the goals change, power consumption immediately drops to the desired value and the accuracy goal is maintained at the cost of some performance. This experiment represents a capability that is not possible with systems which can can only control performance. For this experiment, we want performance to fall because we have hard goals in the power and accuracy dimensions.

3) bodytrack: We deploy the bodytrack benchmark with an initial goal of real-time performance (1.0 in the chart) and a power goal of 40 Watts. One third through the benchmark we transition to a new set of goals: real-time performance and less than 0.25% accuracy loss. These goals represent a transition that might occur when a real tracking application goes from monitoring an empty scene to monitoring a scene with a person. When the scene is empty, there is no need for an accurate track — it is best to keep power low. When there is a human to track, it is worthwhile to spend power to provide predictably high accuracy. In both cases, performance needs to be predictable to keep up with the camera.

CoAdapt initially meets the power and performance goals. The vertical dashed line represents when the goals change. Clearly performance maintains real-time, with a spike immediately following the goal change. At the same time, the accuracy loss drops to the second, smaller limit. Consequently power rises, eventually settling to be about twice the initial power consumption. This experiment represents yet another scenario that is not possible with PowerDial or other existing systems.

4) swish++: Finally, we deploy swish++ with a goal of real-time performance and a power goal of 70 Watts. Halfway through execution, we transition to a new set of goals that increases performance by 2.5× while allowing an accuracy loss of 25%. This goal change represents the search engine responding to a burst of search requests, and attempting to satisfy them while bounding the number of dropped results.

CoAdapt initially meets the power and performance goals. Again, the vertical dashed line represents when the goals change. Performance quickly jumps to meet the new requirement and accuracy loss rises as well. Consequently power rises, but that is acceptable because the new goals are performance and accuracy based.

5) Summary: These results represent a powerful capability: dynamically and automatically adapting behavior based on changes in desired behavior. This is possible because CoAdapt can meet goals in any of the dimensions of power, performance, and accuracy. Furthermore, CoAdapt's control system can switch goals and quickly stabilize to the new targets in new dimensions.

E. Discussion

These results show the benefits achievable with a framework, like CoAdapt, that coordinates system- and application-level adaptation. CoAdapt manages application and system without oscillation. When dynamically adapting in response to unpredictable changes (such as changes in input difficulty), CoAdapt's coordinated approach provides better outcomes than either system-only or application-only approaches. Finally, CoAdapt provides a new capability, reaction to changing goals, that was not possible with prior approaches because they guarantee behavior in only a single dimension.

VI. Related Work

We describe several related efforts in providing predictable behavior including accuracy-aware applications, power-aware systems, and cross-layer approaches.

A. Accuracy-aware Applications

Accuracy-aware applications trade accuracy for performance, power, energy, or other benefits. Such approaches include both static, compile-time analysis of tradeoffs [1, 39, 40] and dynamic, runtime support for tradeoff management [2, 3, 7, 19, 37, 41]. Static analysis guarantees that accuracy bounds are never violated, but it is conservative and may miss chances for additional savings through dynamic customization.

Dynamic management systems tailor behavior online in response to specific inputs. For example, Green maintains accuracy goals while minimizing energy [3], while Eon meets energy goals while maximizing accuracy [41]. Both Green and Eon use heuristic techniques for managing the tradeoff space. PowerDial [19], uses control theoretic techniques to provide performance guarantees while maximizing accuracy. Each of these dynamic management approaches supports a goal in only one, fixed dimension. For example, Green guarantees accuracy, but not power or performance, while PowerDial guarantees performance, but not accuracy or power. Applications requiring support for multiple goals, or working in environments where goals may change cannot benefit from these approaches.

B. Power-aware Systems

There are several system-level frameworks that provide predictable performance while minimizing power or energy consumption. Many approaches manage a single system resource; *e.g.,* processor speed [47], processor duty cycle [49], caches [4], DRAM [50], and disks [26]. Other approaches coordinate multiple resources. For example, Li et al. manage memory and processor speed [27], while Dubach et al. coordinate several microarchitectural features [11], and Maggio et al. coordinate core allocation and clock speed [30]. Meisner et al. propose coordinating low-power states of CPUs, memories, and disks in servers to meet performance goals while minimizing power consumption [31]. Bitirgen et al. coordinate clockspeed, cache, and memory bandwidth in a multicore [6]. Still other approaches focus on managing a general set of system-level components [20, 32, 35, 48]; however, none of these approaches can coordinate system resource usage with application-level adaptation.

Other system-level approaches provide power consumption guarantees while maximizing performance in both clusters and nodes. Cluster-level examples include those proposed by Wang et al. [45] and Raghavendra et al. [34]. Examples of node-level systems include those that manage DVFS for a processor [24], per-core DVFS in a multicore [22], processor idle-time [16], and DRAM [10]. Other approaches provide power guarantees while maximizing performance through management of multiple components including processor and DRAM [8, 14], or DVFS and core allocation [9, 36, 46].

C. Cross-layer Approaches

Static approaches coordinate application and system by marking regions of an application as candidates for accuracy loss and then statically determine when the system can turn that loss into performance or energy savings. The Truffle architecture [12] supports applications which explicitly mark some computations and data as "approximate," allowing accuracy to be exchanged for reduced energy consumption. A similar approach, Parrot, replaces approximate regions of an application with a neural network implementation, which is then executed on a special neural processing unit in hardware [13]. Flikker, allows applications to explicitly mark some data as "non-critical," storing this data in a DRAM that trades accuracy for energy savings [28].

CoAdapt most resembles other approaches that coordinate application and system dynamically. Flinn and Satyanarayanan build a framework for coordinating operating systems and applications to meet user defined energy goals [15]. This system trades application quality for reduced energy consumption, providing up to 30% increase in battery life. GRACE-2 employs hierarchy to provide predictable performance for multimedia applications, making system-level adaptations first and then application-level adaptations second [43]. Like GRACE-2, Agilos uses hierarchy, combined with fuzzy control, to coordinate multimedia applications and system to meet a performance goal [25]. Maggio et al. propose a game-theoretic approach for a decentralized coordination of application and system adaptation which provides real-time guarantees [29]. xTune uses static analysis to build a model of application and system interaction and then refines that model with dynamic feedback [23].

These prior dynamic approaches support behavior guarantees in a single dimension only (energy for Flinn and Satyanarayanan, performance for the others). To the best of our knowledge, CoAdapt is the first approach that can coordinate application and system to meet performance, power, or accuracy goals. This allows CoAdapt to provide higher accuracy than GRACE-2 for the same performance goal (as shown in Fig. 3). In addition, CoAdapt's support for multidimensional goals provides a new capability: ensuring predictable behavior even when goals change at runtime (as shown in Sec. V-D).

Many approaches discussed above use control theoretical designs to provide behavioral guarantees in closed-loop systems [17]. These closed loop systems are robust and capable of maintaining predictable behavior in the presence of unmodeled disturbances. CoAdapt builds on standard control techniques; however, the idea of splitting control into lead and subordinate dimensions and dynamically constructing the controller for the subordinate dimension is a unique contribution of CoAdapt. This technique allows CoAdapt to apply simple linear control systems to what would otherwise be a multidimensional, non-linear control problem.

VII. Conclusion

CoAdapt coordinates the management of application and system, dynamically positioning both in a three-dimensional

performance/power/accuracy tradeoff space. CoAdapt allows applications to continue executing successfully despite changes in application workload, changes in available resources, or even changes in user goals. Our results show that CoAdapt meets goals in multiple dimensions while avoiding oscillations. Unlike prior approaches to cross-layer adaptation, CoAdapt is capable of guaranteeing predictable behavior in any two of the three dimensions of performance, power, and accuracy. CoAdapt responds to phases within an application by ensuring goals are met and adjusting to new optimal configurations. Additionally, CoAdapt enables applications and systems to successfully respond to changes in goals (*e.g.,* switching from high-performance, high-accuracy to high-performance, low-power). CoAdapt is an example from a class of adaptive or autonomic systems designed to overcome the challenges of meeting multiple design constraints while operating in unpredictable environments characterized by fluctuating application workload, resource availability, and user goals.

REFERENCES

[1] J. Ansel et al. "Language and compiler support for auto-tuning variable-accuracy algorithms". In: *CGO*. 2011.

[2] J. Ansel et al. "Siblingrivalry: online autotuning through local competitions". In: *CASES*. 2012.

[3] W. Baek and T. Chilimbi. "Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation". In: *PLDI*. June 2010.

[4] R. Balasubramonian et al. "Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures". In: *MICRO*. 2000.

[5] C. Bienia et al. "The PARSEC Benchmark Suite: Characterization and Architectural Implications". In: *PACT*. 2008.

[6] R. Bitirgen et al. "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach". In: *MICRO*. 2008.

[7] F. Chang and V. Karamcheti. "Automatic Configuration and Run-time Adaptation of Distributed Applications". In: *HPDC*. 2000.

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

[9] R. Cochran et al. "Pack & Cap: adaptive DVFS and thread packing under power caps". In: *MICRO*. 2011.

[10] B. Diniz et al. "Limiting the power consumption of main memory". In: *ISCA*. 2007.

[11] C. Dubach et al. "A Predictive Model for Dynamic Microarchitectural Adaptivity Control". In: *MICRO*. 2010.

[12] H. Esmaeilzadeh et al. "Architecture support for disciplined approximate programming". In: *ASPLOS*. 2012.

[13] H. Esmaeilzadeh et al. "Neural Acceleration for General-Purpose Approximate Programs". In: *MICRO*. 2012.

[14] W. Felter et al. "A performance-conserving approach for reducing peak power consumption in server systems". In: *ICS*. 2005.

[15] J. Flinn and M. Satyanarayanan. "Managing battery lifetime with energy-aware adaptation". In: *ACM Trans. Comput. Syst.* 22.2 (May 2004), pp. 137–179. ISSN: 0734-2071. DOI: 10.1145/986533.986534. URL: http://doi.acm.org/10.1145/986533.986534.

[16] A. Gandhi et al. "Power capping via forced idleness". In: *Workshop on Energy-Efficient Design*. Austin, TX, 2009.

[17] J. L. Hellerstein et al. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004. ISBN: 047126637X.

[18] H. Hoffmann. "Racing vs. Pacing to Idle: A Comparison of Heuristics for Energy-aware Resource Allocation". In: *HotPower*. 2013.

[19] H. Hoffmann et al. "Dynamic Knobs for Responsive Power-Aware Computing". In: *ASPLOS*. 2011.

[20] H. Hoffmann et al. "A Generalized Software Framework for Accurate and Efficient Managment of Performance Goals". In: *EMSOFT*. 2013.

[21] T. Horvath et al. "Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control". In: *Computers, IEEE Transactions on* 56.4 (2007), pp. 444 –458.

[22] C. Isci et al. "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget". In: *MICRO*. 2006.

[23] M. Kim et al. "xTune: A Formal Methodology for Cross-layer Tuning of Mobile Embedded Systems". In: *ACM Trans. Embed. Comput. Syst.* 11.4 (Jan. 2013).

[24] C. Lefurgy et al. "Power capping: a prelude to power shifting". In: *Cluster Computing* 11.2 (2008), pp. 183–195.

[25] B. Li and K. Nahrstedt. "A control-based middleware framework for quality-of-service adaptations". In: *IEEE Journal on Selected Areas in Communications* 17.9 (Sept. 1999), pp. 1632 –1650. ISSN: 0733-8716. DOI: 10.1109/49.790486.

[26] X. Li et al. "Performance directed energy management for main memory and disks". In: *Trans. Storage* 1.3 (Aug. 2005).

[27] X. Li et al. "Cross-component energy management: Joint adaptation of processor and memory". In: *ACM Trans. Archit. Code Optim.* 4.3 (Sept. 2007).

[28] S. Liu et al. "Flikker: saving DRAM refresh-power through critical data partitioning". In: *ASPLOS*. 2011.

[29] M. Maggio et al. "A Game-Theoretic Resource Manager for RT Applications". In: *ECRTS*. 2013.

[30] M. Maggio et al. "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation". In: *IEEE Trans. on Control Systems Technology* 21.1 (2013).

[31] D. Meisner et al. "Power management of online data-intensive services". In: *ISCA* (2011).

[32] "Modular software architecture for flexible reservation mechanisms on heterogeneous resources". In: *Journal of Systems Architecture* 57.4 (2011).

[33] S. Mohapatra et al. "A cross-layer approach for power-performance optimization in distributed mobile systems". In: *IPDPS*. 2005.

[34] R. Raghavendra et al. "No "power" struggles: coordinated multi-level power management for the data center". In: *ASPLOS*. 2008.

[35] R. Rajkumar et al. "A resource allocation model for QoS management". In: *RTSS*. 1997.

[36] K. K. Rangan et al. "Thread motion: fine-grained power management for multi-core systems". In: *ISCA*. 2009.

[37] M. Rinard. "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks". In: *ICS*. 2006.

[38] E. Rotem et al. "Power management architecture of the 2nd generation Intel Core microarchitecture, formerly codenamed Sandy Bridge". In: *Hot Chips*. Aug. 2011.

[39] A. Sampson et al. "EnerJ: approximate data types for safe and general low-power computation". In: *PLDI*. 2011.

[40] S. Sidiroglou-Douskos et al. "Managing performance vs. accuracy trade-offs with loop perforation". In: *ESEC/FSE*. 2011.

[41] J. Sorber et al. "Eon: a language and runtime system for perpetual systems". In: *SenSys*. 2007.

[42] SWISH++. http://swishplusplus.sourceforge.net/.

[43] V. Vardhan et al. "GRACE-2: integrating fine-grained application adaptation with global adaptation for saving energy". In: *IJES* 4.2 (2009).

[44] A. Verma et al. "Server workload analysis for power minimization using consolidation". In: *USENIX Annual technical conference*. 2009.

[45] X. Wang et al. "MIMO Power Control for High-Density Servers in an Enclosure". In: *IEEE Transactions on Parallel and Distributed Systems* 21.10 (2010), pp. 1412–1426.

[46] J. A. Winter et al. "Scalable thread scheduling and global power management for heterogeneous many-core architectures". In: *PACT*. 2010.

[47] Q. Wu et al. "Formal online methods for voltage/frequency control in multiple clock domain microprocessors". In: *ASPLOS*. 2004.

[48] R. Zhang et al. "ControlWare: A middleware architecture for Feedback Control of Software Performance". In: *ICDCS*. 2002.

[49] X. Zhang et al. "A Flexible Framework for Throttling-Enabled Multi-core Management (TEMM)". In: *ICPP*. 2012.

[50] H. Zheng et al. "Mini-rank: Adaptive DRAM architecture for improving memory power efficiency". In: *MICRO*. 2008.

APPENDIX

A. Analysis of CoAdapt Control

CoAdapt uses a control theoretic runtime decision engine to adapt application and system behavior in response to unpredictable events. One advantage of the control theoretic approach is its properties can be described analytically. Thus, CoAdapt can provide provable guarantees that it will meet the desired goals. In particular, one can describe CoAdapt's behavior in terms of the SASO properties defined by Hellerstein et al. [17] and illustrated in Fig. 6: *stability, accuracy, settling time, and maximum overshoot*. This section first describes these properties and then uses them to evaluate CoAdapt's behavior and ability to meet goals.

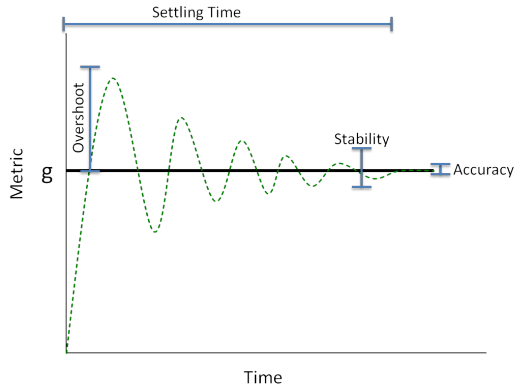

Fig. 6: Illustration of the SASO (stability, accuracy, settling time, and overshoot) properties.

1) Definition of SASO Properties: **Stability** Stability refers to the feedback measured in the goal dimension over time. A *stable* system converges to a single value; *i.e.,* the derivative of the signal eventually becomes zero. For example, if an application has a performance goal, stability refers to the property that the performance converges to a steady value. The remaining SASO properties are defined in terms of a stable system.

Accuracy Unfortunately, the term accuracy is overloaded. In this section accuracy refers to a property of the control system rather than a property of an application's output. An accurate controller converges to the goal. This property is demonstrated empirically in Secs. V-B and V-D, which show applications converging to goals in various dimensions. Control accuracy may be the most important property for CoAdapt users, because accurate control means that goals are met. Accuracy implies that for a goal $g_d(t)$, there exists a time t_a, such that $e_d(t) = 0 \quad \forall t > t_a$.

Settling Time Settling time refers to the time that passes from application start (or a change in phase or goals) to the point where the system becomes stable. That is settling time is the difference between the start time t_0 and t_a.

Max Overshoot The maximum overshoot refers to the largest amount by which CoAdapt might fail to miss the target on its way to becoming stable.

2) Properties of CoAdapt: This analysis makes two assumptions. First, we assume that behavior in all dimensions is *bounded*; *i.e.,* an uncontrolled application cannot continually increase or decrease its performance, power consumption, or output accuracy. For example, these properties do not apply to an application whose performance continually increases (or decreases) for its duration. Second, we assume that any changes in goals occur at least 2τ time units apart. Faster goal changes will not allow the controller time to adapt to the new goal. Given these assumptions, we evaluate CoAdapt following the standard procedure outlined by Hellerstein et al. [17]. All analysis is performed in the Z-domain.

Stability We analyze stability by examining the Z-transform $L(z)$ of the closed loop control system that maps the goal $g_d(t)$ to feedback $f_d(t)$:

$$\begin{aligned} F_d(z) &= \frac{b_d}{z} \\ U_d(z) &= \frac{z}{b_d \cdot (z-1)} \\ L_d(z) &= \frac{F_d(z) \cdot U_d(z)}{1 + F_d(z) \cdot U_d(z)} \\ &= \frac{1/(z-1)}{1 + 1/(z-1)} \\ &= \frac{1}{z} \end{aligned} \tag{33}$$

Given $L_d(z)$, we show that it is stable by demonstrating its poles lie within the unit circle. $L_d(z)$ has a single pole at 0, so the control system is stable.

Accuracy A control system is accurate if its steady state gain is unity, a property which can be determined by evaluating the Z-transform at $z = 1$. As $L_d(1) = 1$, CoAdapt has a unit steady state gain and thus is expected to converge to the goal.

Settling Time The time required to reach steady state can be approximated as $-4/\log(|p|)$, where p is the largest pole in the system. Thus, CoAdapt can be expected to converge almost instantly. In practice the convergence time will be limited by τ, the time quantum over which CoAdapt schedules knob settings. The practical settling time is depicted in the charts detailing the various behaviors as a function of time.

Maximum Overshoot Maximum overshoot is derived from the poles. Again, the only pole occurs at zero, so we do not expect the system to overshoot the goal. In practice overshoots can occur when the application changes phase, when goals change, or due to unpredicted dynamics in the interaction between application and system configurations. We expect these periods of over or undershoot to be short because of the short settling time.

3) Properties of Multi-goal Control: CoAdapt handles goals in multiple dimensions by coupling controllers and creating a lead and subordinate control system. The lead dimension is not affected by the subordinate dimension, so the above analysis applies to the lead controller. The subordinate controller, however, is no longer a time-invariant system because its coefficient $\hat{b}_d(t)$ is time-varying. However, $\hat{b}_d(t)$ is a function of the lead controller, and when the lead controller stabilizes, $\hat{b}_d(t)$ also becomes stable. Therefore, once the lead controller stabilizes, the subordinate controller will also stabilize accurately with a short settling time and small overshoot. Thus, there may be a period of 2τ time units (one τ for the lead controller to stabilize and one τ for the subordinate to subsequently stabilize) where the subordinate controller is inaccurate. Overall, we believe this is a reasonable cost for providing guarantees for goals in multiple dimensions.

4) Optimality of CoAdapt: Eqns. 7–11 and Eqns. 21–25 describe optimization problems used to translate the control signals for the lead and subordinate dimensions into actual configurations of available knobs. As mentioned in the body of the paper, solving these optimization problems online is impractical. However, CoAdapt's greatest priority is ensuring predictable behavior in the goal dimensions. Therefore,

CoAdapt employs the heuristics (described in Eqns. 12–18 and Eqns. 26–32). These heuristics guarantee that the goal behavior is achieved, but may sacrifice some optimality; *i.e.,* they are feasible solutions to the constraint equations, but may not be optimal solutions. Nevertheless, similar heuristics have been studied for energy optimization under performance constraints and they have been found to be near optimal [18].

B. Implementation

This section briefly describes our implementation of CoAdapt on the target system. We cover specification of goals and feedback, specification of available configurations, and the implementation of the runtime itself.

1) Specifying Feedback and Goals: CoAdapt reads goals and feedback, using these values to drive adaptation. To specify goals, the runtime exports a POSIX shared memory segment that holds three floating point values for storing the performance, power, and accuracy goal. We have a command line tool that attaches to this memory segment and sets the goals. The tool can be used to change goals online. It is beyond the scope of the paper, but in the future we hope to replace the command line tool with a graphical user interface for performing the same function.

In addition to the goals, CoAdapt needs feedback on performance, power, and accuracy. Performance and power consumption can be read directly off hardware performance counters on our target system. In general, CoAdapt is agnostic about the source of its power and performance feedback. The only requirement is that the performance metric increase with increasing performance. Alternatively, PowerDial already instruments the target applications to provide application specific performance feedback. [19]. These applications emit heartbeats at important intervals and timestamps for each heartbeat are stored in a separate shared memory segment. By reading these timestamps, CoAdapt can obtain an application specific metric of performance. We augment this capability by reading the energy data available on the Sandy Bridge processor every time a heartbeat is emitted, allowing CoAdapt to calculate power data for the application. We note that power and performance can be controlled without any further modification to the application, other than that done by PowerDial to expose application configurations. However, controlling accuracy does require a small additional modification. We modify PowerDial so that each heartbeat emits the current accuracy as well as its time-stamp.

Thus, goals are set by writing to one POSIX shared memory segment, while feedback is acquired by reading from another.

2) Specifying Configurations: The PowerDial system uses an internal interface for specifying the application components, or *knobs*, which can be configured at runtime [19]. Each knob is specified by a number of settings, a set of addresses that can be written to change application behavior, and a table detailing the values to be written for each setting and their expected speedups and accuracy loss. CoAdapt extends this interface by adding the ability to specify the expected power consumption of a configuration. We then implement the system level knobs by creating a separate routine that checks a shared memory location for the desired setting and reconfigures the system component accordingly.

We note that PowerDial exhaustively profiles all possible application configurations to determine initial estimates of their performance and accuracy. As additional system configurations become available, the total number of configurations grows exponentially, and it quickly becomes impractical to profile all possible configurations. Therefore, we profile system components individually and rely on the control system to correct any errors that arise. The effects of this policy are visible in the results. In the example section, x264 initially exceeds its performance goal (see Fig. 1) before settling to the target. Other results are similar, and present empirical evidence that the control system can handle the potential errors in the model. This handling could later be improved by either 1) taking the time to profile all possible configurations or 2) making the controller itself adaptive [20].

3) Implementing the Runtime: We have two implementations of the runtime system, both written in C. One is a standalone process and the other gets compiled into applications and runs as a separate thread. In both cases, the behavior is similar. The runtime executes the algorithm from Algorithm 1. All goals and feedback are read from the appropriate shared memory segments. Configurations are set by writing to the values specified for each knob. The runtime uses the `nanosleep` command to put itself to sleep to minimize interference in between setting configurations and calculating new configurations. Having the runtime put itself to sleep is essential for reducing overhead – it would be prohibitively expensive for the runtime to poll for feedback.