MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing

Anne Farrell
University of Chicago
amfarrell@cs.uchicago.edu

Henry Hoffmann
University of Chicago
hankhoffmann@cs.uchicago.edu

***Abstract*—Energy efficiency and timeliness (*i.e.,* predictable job latency) are two central concerns for real-time systems. While both are essential, they are opposing: hard timing guarantees require conservative resource allocation while energy minimization requires aggressively releasing resources and occasionally violating timing constraints. Recent work on approximate computing, however, opens up a new dimension of optimization: application accuracy. In this paper, we use approximate computing to achieve both hard timing guarantees and energy efficiency. Specifically, we propose MEANTIME: a runtime control system that delivers hard real-time latency guarantees and energy-minimal resource usage by sacrificing a small amount of application accuracy. We test MEANTIME on a real Linux/x86 system with seven applications. Overall, we find that MEANTIME never violates real-time deadlines and sacrifices a small amount (typically less than 2%) of accuracy yet consumes only 68% of the energy of a conservative, full accuracy approach.**

I. Introduction

Real-time systems require both predictable timing and energy-efficiency. When predictability takes the form of hard real-time constraints, these two goals are conflicting [16]. The conflict arises because hard real-time guarantees require reserving resources sufficient for worst case latency. In contrast, energy efficiency requires allocating resources that just meet the needs of the current job. Even if worst case resource allocation is coupled with aggressive energy reduction (*e.g.,* in the form of voltage scaling [19] or sleep states [32]), this strategy is less energy-efficient than allocating for the current case. This inefficiency has been demonstrated both analytically [1, 8, 35] and empirically [26, 33, 38, 44].

Recent research on *approximate computing* examines applications that can trade the *accuracy* of their results in exchange for reduced energy consumption [4, 6, 17, 30, 55]. These approximate computations open up a third dimension of optimization, making it possible to simultaneously tradeoff timing, energy, and accuracy [27]. Many real-time computations (for both embedded and web applications) are particularly well-suited for approximation as they typically involve large amounts of signal, image, media, and data processing, where it is easy to quantify application accuracy (*e.g.,* as signal-to-noise-ratio) and carefully trade it for other benefits. While approximate computing frameworks improve application performance and reduce energy [4, 27, 30, 51], they provide (at best) limited guarantees, and are thus unsuitable for hard real-time constraints. *This paper addresses the challenge of meeting both hard real-time constraints and low energy through careful application of approximate computing.*

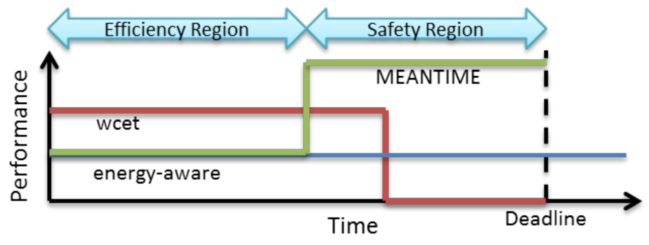

Fig. 1: Conceptual model of MEANTIME compared to worst-case and energy-aware resource allocation.

A. The MEANTIME Approach

Specifically, we develop MEANTIME[1], a runtime system that provides hard real-time guarantees with near minimal energy consumption by potentially reducing application accuracy. MEANTIME couples an existing state-of-the-art resource allocator [34] – which optimizes for the current job – with a novel *governor* which monitors job execution and reconfigures the application to execute in a less accurate configuration to avoid any timing violations. The resource allocator uses control theory to allocate for the current case. This control theoretical approach, by itself, will violate timing constraints. Therefore, the governor determines how much to manipulate application accuracy to ensure that the timing constraints are never violated despite the dynamics of the underlying controller and any application-level fluctuations.

Figure 1 illustrates the intuition behind MEANTIME and compares it to other resource allocation approaches. The figure shows time, with a deadline, on the x-axis and performance on the y-axis. Allocating for worst case execution time (wcet) requires using all resources in the system and then idling, or sleeping, (no performance) until the deadline. Energy-aware allocation estimates the resource needs of the current job, but it may miss the deadline due to variance in job requirements.

In contrast, MEANTIME reasons about wcet for both the application's nominal behavior and its acceptable approximate variants. Given this information, MEANTIME allocates for the current case, while computing two temporal regions within the deadline: an *efficiency region* and a *safety region* (shown in Figure 1). The efficiency region represents the time to run in the application's full-accuracy configuration using the resources specified by the energy-aware allocator. The safety region represents the time for which the application must switch to an approximate configuration (still using the assigned resources). MEANTIME's key idea is using timing analysis and approximate computation to determine the efficiency and

[1]The name stands for Minimal Energy ANd TIMEliness.

TABLE I: Server resources.

Processor	Cores	Speeds (GHz)	TurboBoost	HyperThreads	Memory Controllers	Sleep Power (W)	Max Power(W)	Configurations
Xeon E5-2690	16	1.2–2.9	yes	yes	2	70	350	1024

safety regions, ensuring both energy efficiency and timeliness.

B. Summary of Results

We implement MEANTIME and evaluate it on a Linux/x86 server. We test its timing guarantees, energy, and accuracy for seven different applications. These benchmarks represent a range of computations – including media processing, signal processing, web search, and financial analysis – all of which might require both timing guarantees and energy efficiency. These applications were not originally designed to provide any timing predictability, still MEANTIME achieves:

- **Efficacy with a range of behaviors**: Our benchmarks have a wide range of statistical behavior, including both high and low variance in job latency (see Section IV-D).
- **Predictable timing**: MEANTIME has no deadline misses for any benchmark (see Section V-A).
- **Energy Savings**: MEANTIME requires only 68% of the energy of allocating for worst case and aggressively sleeping the system (see Section V-B).
- **High Accuracy**: MEANTIME achieves accuracies that are typically very close to the nominal behavior (*i.e.,* almost no accuracy loss, see Section V-C).
- **Adapts to changing requirements**: MEANTIME supports changing users goals – *e.g.,* if desired, it can revert to full accuracy mode as the application runs (see Section V-D).
- **Adapts to application phases**: MEANTIME reacts to changing application behavior while maintaining timing guarantees (see Section V-E).

C. Scope

MEANTIME is not designed for all real-time applications – it is instead designed for applications that 1) have viable performance versus accuracy trade-off spaces and 2) must satisfy hard real-time constraints and minimize energy consumption despite large fluctuations in application workload. We believe a number of real-time applications meet these criteria (as evidenced by our use of existing benchmarks which exhibit performance and accuracy tradeoffs).

D. Contributions:

- Design of a runtime system that provides both hard real-time guarantees and energy efficiency by sacrificing computation accuracy.
- Experimental evaluation of the runtime on a real system with six different benchmarks.

The rest of the paper is organized as follows. Section II provides intuition for the MEANTIME framework. Section III describes the framework and how the efficiency and safety regions are calculated. Section IV describes the system, applications, and their timing statistics. Section V discusses MEANTIME's empirical evaluation. Section VI describes related work and Section VII concludes.

II. Motivational Example

We motivate MEANTIME's combination of hard timing guarantees and energy efficiency through a radar signal processing example similar to what might be found on a unmanned autonomous vehicle (UAV). The radar system has timing constraints – if radar frames are not processed within a strict latency, the system cannot respond to external events. The UAV, however, is limited by available energy. In addition, the radar processing chain is amenable to approximate computation – its signal processing algorithms can trade reduced signal-to-noise ratio for performance gains.

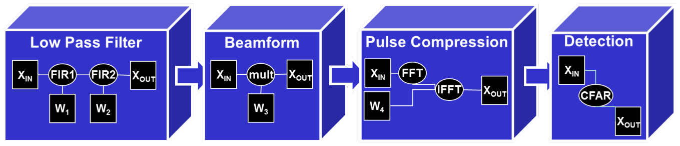

Fig. 2: Block diagram for radar application.

Figure 2 illustrates the radar's signal processing chain. The first block (LPF) performs a low-pass filter to eliminate high-frequency noise. The second block (BF) does beam-forming which allows a phased array radar to "look" in a particular direction. The third block (PC) performs pulse compression, which concentrates energy. The final block is constant false alarm rate detection (CFAR), which identifies targets. Several parameters control the tradeoff between signal-to-noise ratio (SNR) and performance (for more detail see Section IV).

Our experimental platform is a 16 core Linux/x86 server with 16 clock speeds, hyperthreads, and two memory controllers. The radar application is parallelized, so that each processing block can be executed across up to 32 threads. By allocating different resources to the radar, we achieve different tradeoffs between performance and power consumption. The available configurations are summarized in Table I.

We run the radar application on the server with all resources allocated and empirically determine the worst case latency for a single radar frame. Additionally, we gather other statistics including the minimum, mean, and standard deviation of latency. These values are summarized in Table II.

TABLE II: Radar timing.

Latency	Measurement (s)
Mean	0.054
Minimum	0.014
Maximum	0.067
STDEV	0.016
STDEV/Mean	0.302

The timing demonstrates the classic conflict between timeliness and energy efficiency. For timeliness, we can allocate for worst-case execution time. If a radar frame finishes early (*i.e.,* it is not a worst case frame), then we can simply sleep (or idle) the processor until the next frame is available [38]. To allocate for energy efficiency, we could use a state of the art energy-aware approach (*e.g.,* [42]).

The key idea behind MEANTIME is using energy-aware techniques to allocate for the average case, but avoid timing violations by quickly switching the application to an approximate configuration that is guaranteed to meet timing even for a worst case frame. Section III discusses in detail how MEANTIME calculates the times to spend in different configurations.

We demonstrate the benefits of MEANTIME for the radar application by comparing its latency, energy, and accuracy to

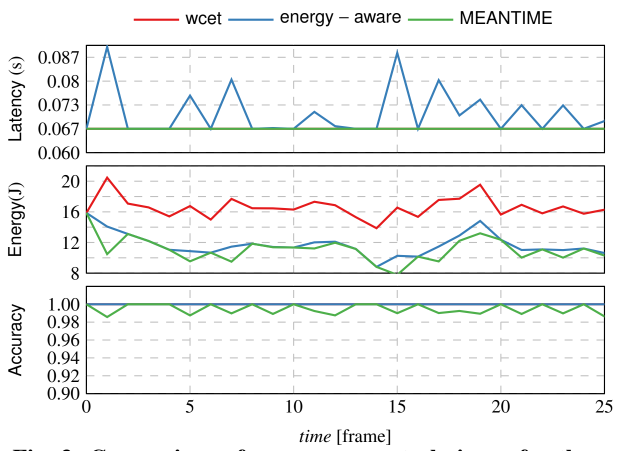

Fig. 3: Comparison of energy-aware techniques for the radar.

both allocating for worst case and allocating for average case using an energy aware resource allocator based on control theory [34, 42]. These results are illustrated in Figure 3. Each chart shows time (measured in radar frames) on the x-axis. The top chart shows latency normalized to the target latency (1/15 seconds per frame, the worst case latency measured on our system). The middle chart shows the energy per frame in Joules. The bottom chart shows the SNR normalized to the default configuration; *i.e.,* unity represents no accuracy loss.

The results in Figure 3 demonstrate that MEANTIME achieves both timeliness and energy efficiency. Indeed, MEANTIME achieves the same timing as allocating for worst case execution time while actually improving energy consumption compared to the energy-aware system. Specifically, the energy-aware approach consumes 12.2 Joules, allocating for wcet consumes 16.9 Joules, and MEANTIME consumes 11.4 Joules. These results come at the cost of a small amount of accuracy; in this case the SNR falls to 99.4% of the original application.

MEANTIME actually consumes less energy than even state-of-the-art energy-aware resource allocators. This savings arises because MEANTIME builds off the state-of-the-art; MEANTIME allocates the same resources as the energy-aware system, but it never violates latency constraints. Therefore, MEANTIME always takes time less than or equal to the energy-aware approach.

III. The MEANTIME Framework

This section provides the technical details on MEANTIME, illustrated in Figure 4. As shown in the figure, MEANTIME couples a *governor* with an *energy-aware* controller. The controller allocates system resources (*e.g.,* cores, clockspeed, memory bandwidth) and the governor ensures that the application's timing requirements are met. The controller receives a latency target from the governor and then computes the minimal energy set of resources needed to meet that target. The governor receives, from the user, timing requirements and a timing analysis of the application in both its default state and its approximate configurations. The user has complete control of which set of approximations MEANTIME considers. The governor translates these timing requirements and analysis into a latency target, which is passed to the controller. The timing information is also used to calculate the *efficiency* and *safety* regions. The efficiency region is the time to spend in the application's nominal configuration using the resource

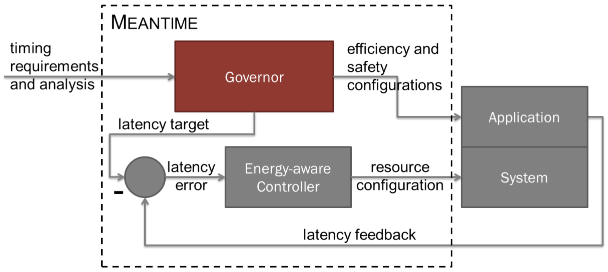

Fig. 4: Overview of the MEANTIME approach.

configuration suggested by the controller. The safety region is the time to spend in an approximate configuration, ensuring the job completes by the deadline.

This section describes MEANTIME's controller and governor. The controller itself is not a contribution of MEANTIME; we make use of existing research on energy-aware control for soft real-time requirements. MEANTIME's contributions are 1) the governor which ensures timeliness, and 2) the integration of the governor with existing control techniques. The remainder of this section provides an overview of energy-aware control schemes, then describes the governor.

A. Approximate Computing

A number of approximate computing frameworks exist (see Section VI for a detailed survey). The commonality between all of them is that they enable the application to expose tradeoffs between accuracy and resource usage (almost always quantified as execution time). Execution time is straightforward. Accuracy, however, is defined in an application-specific manner. In our radar example, accuracy is signal-to-noise ratio. In a search engine, it might be the number of results returned. We discuss the accuracy metrics used for our test applications in Section IV. The essential thing for MEANTIME is that the framework exposes the performance and accuracy tradeoffs for a particular application. Performance gains can be quantified in a standard way. Accuracy need only be a total order; MEANTIME works even if the exact accuracy of a configuration cannot be specified, as long as it can be ranked relative to other configurations. MEANTIME requires this ordering so that it can choose the highest accuracy configuration that meets a constraint.

B. Control for Energy Efficiency

Control theory provides a powerful capability for computing systems in the form of formal guarantees about how a controlled system responds to dynamic events [20, 24, 61]. This powerful set of techniques has then been used to create computing systems that meet soft real-time deadlines while minimizing resource use. Examples can be found controlling latency in multi-tier webservers [11, 31], in media processing [58, 60], and on mobile platforms [34].

To minimize energy for a latency constraint, the controller is given a target latency. It then measures the current latency, computes the error between the target and current, and determines the most efficient resource allocation that will eliminate the error. Different control implementations have different tradeoffs between how fast they react and how stable they are in the face of noise. The important thing for MEANTIME is that control approaches have repeatedly proven capable of near optimal resource allocation [11, 25, 31, 34, 60].

The drawback is that these techniques, by themselves, cannot provide both hard real-time latency and energy efficiency. MEANTIME, therefore, relies on a control system to allocate resources for the current case and minimize energy. Several existing control systems might be appropriate for this task, but MEANTIME builds on POET, a portable, open-source energy-aware control implementation [34]. The concept is general, however, and could be applied to many different control systems.

Within MEANTIME, the controller is responsible for reacting to application phases; *e.g.,* reducing resource usage if the workload gets easier and increasing resources if it gets harder. Control systems are well-suited to this task, as their whole purpose is managing system dynamics. Of course, control systems react to changes by detecting errors between a target latency and an achieved latency. Such latency errors represent missed deadlines and are not acceptable for hard real-time. Therefore, MEANTIME's governor ensures that the user's target latency requirement is not violated.

C. Governor for Timeliness

The governor's primary responsibility is deriving the efficiency and safety regions. To perform this task, the governor requires the following inputs:

1) Worst case timing analysis for the application in its nominal (full accuracy) configuration.
2) Worst case timing analysis (expressed as speedup) for any variable accuracy application configurations.
3) Worst case timing analysis for switching application or system configurations.

This sections show how the governor derives the efficiency and safety regions from these inputs.

1) Notation: Table III summarizes the notation used in this section. We assume an application comprised of many jobs. Each job has a *workload* representing its processing requirements. We assume we know the worst case workload W and the deadline for completing the work t. We label the worst case latency and computation rate as t_{wc} and r_{wc} and best case values are t_{bc} and r_{bc}. Internally, MEANTIME records the relationship between the best and worst case as $\Delta = t_{bc}/t_{wc}$. Note that $0 < \Delta \leq 1$.

Both the machine and application are configurable. The machine's resources can be allocated and the application's accuracy can be changed to speed up computation. All speedups are measured relative to the full accuracy application running with all machine resources. We assume we know s_0, the maximum worst case speedup available from changing application accuracy. That is, s_0 is the minimum speedup that will be observed from approximation.

2) Goal: Given the above assumptions and notation, the governor computes the safety and efficiency regions such that the workload of all jobs is completed by their deadlines. We write these requirements as three constraints:

$$t_s \cdot r_s + t_e \cdot r_e \geq W \quad (1)$$

$$t_s + t_e \leq t \quad (2)$$

$$t_s,\ t_e \geq 0 \quad (3)$$

TABLE III: Notation.

	Variable	Meaning
Input	W	job workload in worst case
	t	deadline for completing work
	t_{wc}	worst case latency
	t_{bc}	best case latency
	r_{wc}	worst case computation rate with full accuracy
	r_{bc}	best case computation rate with full accuracy
	s_0	minimum speedup from approximation
	t_{switch}	worst case time to switch app. & sys. config.
Internal	t_s	time to spend in safety region
	t_e	time to spend in efficiency region
	r_s	computation rate in safety region
	r_e	computation rate in efficiency region
	Δ	ratio of best to worst case, t_{bc}/t_{wc}

Eqn. 1 states that total capacity for work should be greater or equal to the worst case workload[2]. This constraint is conservative, but if it holds then we know the worst case work will be accomplished. The second constraint ensures that the total time is less than or equal to the deadline time. The third constraint ensures that the times are non-negative, which means the deadline can be met in practice. The governor determines the values of t_e and t_s such that the constraints will be respected, ensuring hard real-time guarantees.

3) Deriving Efficiency and Safety Regions: MEANTIME considers the most difficult situation, which occurs when the application transitions from a best case latency job to a worst case job. The controller will detect the best case and allocate a commensurate amount of resources. Thus, the controller allocated resources for best case. The combination of worst case workload and resources allocated for best case will severely degrade measured performance. MEANTIME calculates this degraded performance as Δr_{wc}. Therefore, the computation rate in the efficiency region can be as low as $r_e = \Delta r_{wc}$. Furthermore, we can rewrite r_s in terms of known quantities by recognizing that the worst case speed in the safety region is $r_s = \Delta r_{wc} \cdot s_0$. To ensure the constraints are satisfied even in this situation, we substitute into Eqns. 1 and 2 to obtain:

$$t_s \cdot \Delta r_{wc} \cdot s_0 + t_e \cdot \Delta r_{wc} = W \quad (4)$$

$$t_s + t_e + t_{switch} = t \quad (5)$$

We now have a system of two equations with two unknowns: t_s and t_e, the time to spend in the safety and efficiency regions, respectively. All other quantities are known based on timing analysis as stated in the above assumptions. We therefore rewrite Eqn. 5 as $t_s = t - t_e - t_{switch}$ and substitute into Eqn. 4 to obtain:

$$W = (t - t_e - t_{switch}) \cdot \Delta r_{wc} \cdot s_0 + t_e \cdot \Delta r_{wc} \quad (6)$$

$$t_{wc} = \frac{W}{\Delta r_{wc}} = (t - t_e - t_{switch}) \cdot s_0 + t_e \quad (7)$$

$$t_e = \frac{t_{wc} - s_0 \cdot (t - t_{switch})}{1 - s_0} \quad (8)$$

Thus, Eqn. 8 gives the efficiency region; *i.e.,* the largest amount of time we can spend using the resources specified by the controller. We then transition to the variable accuracy configuration (keeping resource usage the same) for $t_s = t - t_e - t_{switch}$ time. A negative value of either t_s or t_e indicates that the application is not schedulable.

[2]In practice, we can always idle or sleep if we reserve too much capacity and finish early.

An additional quick check for schedulability can be done by checking whether $s_0 \geq \frac{1}{\Delta}$. If the available speedup cannot overcome the potential difference between best and worst case, then this approach may miss deadlines. If this is the case, MEANTIME sets a stricter latency goal than the user specified, forcing the controller to be more conservative. MEANTIME then aggressively sleeps or idles so that the user sees the desired latency.

4) Minimizing Accuracy Loss for Schedulability: The prior section derived the time to spend in the efficiency region for an application with a single approximate configuration. In practice, however, approximate computations expose a wide range of tradeoffs between accuracy and speedup [30, 53]. We denote the accuracy and speedup of application configuration i as a^i and s^i, respectively. To maximize the accuracy and maintain hard real-time constraints, we formulate the following optimization problem:

$$maximize \sum_i \frac{t_s^i}{t} \cdot a^i \tag{9}$$

$$s.t.$$

$$\sum_i t_s^i \cdot r_{wc} \cdot s^i + t_e \cdot \Delta r_{wc} \quad \geq \quad W \tag{10}$$

$$t_{switch} + \sum_i t_s^i + t_e \quad \leq \quad t \tag{11}$$

$$t_s^i, \;\; t_e \quad \geq \quad 0 \tag{12}$$

Here, we assume that the user has supplied some set of approximations, all of which are acceptable, if not preferable. The goal is to find the maximum accuracy that guarantees the constraints in Eqns. 10–12. If this linear program has no feasible solution, the application is not schedulable on the system. MEANTIME, in general, does not need to solve this program often. If the acceptable accuracy never changes, then the program can be solved once at initialization time. If acceptable accuracy may change as the program runs, MEANTIME will solve a new program with a new set of variable accuracy configurations each time it changes.

Eqns. 10–12 has two non-trivial constraints. By the theory of linear programming, that means that there is an optimal solution with at most two t_s^i greater than 0 and all other equal to 0 [15]. Thus, during any interval, MEANTIME will switch configurations at most twice. This limitation on switching time means that we can provide a fairly tight bound on t_{switch}, meaning that switching time will have little impact on energy efficiency in practice.

5) Providing Feedback to the Controller: Control systems have been widely used to improve energy efficiency because control theory provides formal guarantees concerning how the system will react to dynamic events. For example, when an application shifts from a computationally intensive phase to an easy phase, the controller reacts to this change by reducing resource usage. The problem for hard real-time guarantees is that the control system reacts by detecting an error between the desired behavior and observed behavior. If the behavior under control is job latency, this error may result in missed deadlines.

This is an example of the fundamental tension between hard real-time and energy efficiency. We would like the runtime to reduce resource usage, but the control system will not detect the phase shift if every deadline is respected. MEANTIME overcomes this drawback by intercepting the latency feedback and altering it. Instead of providing the actual latency feedback (which will always be the same as all deadlines are guaranteed), MEANTIME estimates what the latency would have been if the application had not switched to a less accurate configuration and passes this latency to the controller. This latency estimate is denoted as $\hat{t}$ and calculated as:

$$\hat{t} \quad = \quad t_e + t_s \cdot s_0 + t_{switch} \tag{13}$$

Passing this estimated latency to the controller allows MEANTIME to maintain responsiveness and energy-efficiency while still meeting hard real-time deadlines.

TABLE IV: System configurations.

Configuration	Settings	Max Speedup	Max Powerup
clock speed	16	3.23	2.05
core usage	16	15.99	2.03
hyperthreading	2	1.92	1.11
sleep	n/a	1.00	1.00
mem controllers	2	1.84	1.11

D. Summary of MEANTIME

This section has provided the technical overview for MEANTIME. MEANTIME works with existing control theoretic approaches that provide soft real-time support while minimizing energy consumption. MEANTIME augments these control based approaches in two novel ways. First, it computes the safety and efficiency regions as described above – thus combining hard real-time with energy efficiency. Second, it alters the feedback passed to the controller to maintain responsiveness to application phases despite the fact that deadlines are never violated.

IV. Experimental Setup

A. Hardware Platform

Our hardware platform is a is a dual socket system with two eight-core Intel Xeon E5-2690 processors, supporting hyperthreading, TurboBoost, and 16 configurable processor speeds (summarized in Table I). This is a Sandy Bridge processor, which allows power and energy consumption to be read directly from registers at runtime in millisecond intervals [49]. We run Linux 3.2.0 with the `cpufrequtils` package to change clock speeds and the `numalib` package to manipulate memory controller usage.

This system supports five configurable resources that alter performance and power tradeoffs. The first uses the `cpufrequtils` package to control voltage and frequency. 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 is power-gated, so reducing core usage reduces both power consumption and performance. In addition, hyperthreading can be turned on or off to provide further power/performance tuning. The processor also supports a low-power *sleep* state allowing it to *race-to-sleep*; *i.e.,* complete work as quickly as possible and maximize sleep time [26, 32, 35]. The final adaptation changes the virtual to physical page mapping to alter memory controller usage. For applications which make significant amounts of memory references, using fewer memory controllers decreases performance and power consumption.

TABLE V: Approximate Application configurations.

App.	Configs.	Min. Spdup	Max Acc. Loss (%)
x264	560	3.96	6.2
bodytrack	200	5.24	14.4
swaptions	100	50.43	1.5
ferret	8	1.24	18.24
streamcluster	16	3.82	54.8
swish++	6	1.47	83.4
radar	512	3.95	73.4

TABLE VI: Application Input Details.

Application	Input	Jobs
x264	native	512 frames
bodytrack	sequenceB	261 frames
swaptions	randomized parameters	256 swaptions
ferret	corel	2000 queries
streamcluster	7 card poker hands	1000 hands
swish++	search strings	100,000 queries
radar	radar pulses	100 pulses

These system knobs are summarized in Table IV, 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 sleep settings, as any application could be stalled arbitrarily. In addition to the processor's support for power measurements, we measure total system power consumption with a WattsUp meter. The system sleeps at 70 Watts. The maximum measured power consumption is 350 Watts. *All energy and power numbers reported in this paper consider total system usage.*

B. Applications

We use seven benchmark applications: x264, bodytrack, swaptions, ferret, and streamcluster (from the PARSEC benchmark suite [12]); as well as swish++, an open-source search engine [57] and the radar signal processing application from the example section [28]. We use the PowerDial compiler to modify all seven benchmarks so they support dynamic approximation [30]. PowerDial turns static configuration parameters into a data structure which controls the application's runtime behavior. PowerDial also instruments the applications, providing latency feedback for every outer loop iteration [30].

We measure application accuracy as a normalized distance from full accuracy. This is a standard metric allowing comparison across applications [48]. Changes in application performance are measured as speedup, the factor by which latency decreases when moving from the nominal setting. Unlike previous work on approximation that maximized average achievable speedup, MEANTIME cares about the worst case speedup. Thus, we measure the minimum speedup achievable with these transformations. This section describes the tradeoffs exposed by each of these applications. Table V summarizes the application-level configurations, showing the total number of available configurations as well as the minimum speedup and maximum accuracy loss. These applications represent a range of workloads which might be run on an embedded system with both timing and energy constraints.

x264: This video encoder compresses a raw input according to the H.264 standard. It can decrease the frame latency at a cost of increased noise. 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 encoder's accuracy is measured by recording both the peak signal-to-noise ratio (PSNR) and the encoded bitrate, and weighting these two quantities equally.

bodytrack: This application uses an annealed particle filter to track a human moving through a video. The filter parameters trade the track's quality and the frame latency. The application exposes two knobs, one changes the number of annealing layers and another changes the number of particles. Together, these knobs support 200 different configurations.

ferret: This application performs content-based similarity search for images; *i.e.,* it finds images similar to a query image. Both the image analysis and the location sensitive hash used to find similar images are amenable to approximation. In total, ferret supports 8 different approximate configurations. The search accuracy is evaluated using F-measure, the harmonic mean of precision and recall[3].

streamcluster: This application is an online approximation of the k-means clustering algorithm. It has two knobs which trade clustering accuracy (measured using the B^3 score [2]) and clustering latency. The first knob changes the number of iterations used in the approximation, the second knob changes the distance metric used to assign a sample to a cluster. Together, these knobs expose 16 different configurations.

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 decreased pricing latency. The application has a single knob, with 100 settings, controlling the number of Monte Carlo simulations performed per swaption.

swish: This search engine indexes and searches files. Given an input query, it finds matching documents, ranks them, and returns a formatted, ranked document list. The application supports a single knob which trades the number of documents returned per request for request latency. Accuracy is again evaluated using F-measure.

radar: This application is the centerpiece of our example (see Section II). It is the front-end of a radar signal processing system and is responsible for turning raw radar returns into lists of targets. The application supports four different knobs that tradeoff signal-to-noise ratio (SNR) [4]. The first two knobs change the decimation ratios in the finite impulse response filters that make up the LPF stage. The third knob changes the number of beams used in the beamformer. The fourth knob changes the range resolution. The application can enter 512 separate configurations using these four knobs.

C. Tradeoffs Between Accuracy and Energy

These applications all expose tradeoffs between their runtime and their result accuracy. The presence of these tradeoffs is fundamental to the ideas in this paper. We illustrate the tradeoff spaces in Figure 5. The figure consists of seven charts for each of our seven applications. The x-axis of each chart shows the speedup and the y-axis shows the resulting accuracy. Speedup an accuracy are both reported normalized to the

[3]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.

[4]Higher SNR makes targets easier to detect, lower SNR makes targets harder to detect. All configurations used in this paper still detect all targets with more than 10dB of margin.

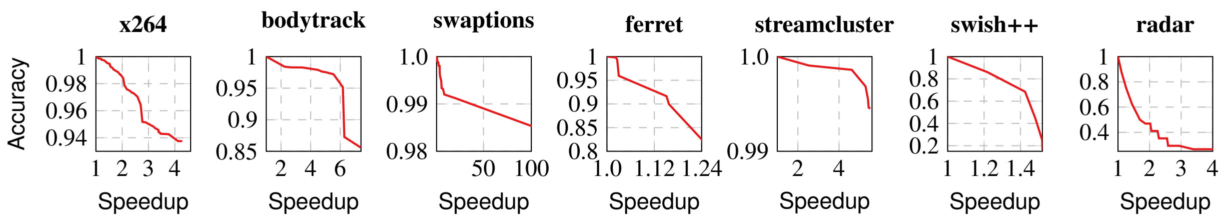

Fig. 5: Speedup and accuracy tradeoffs for test applications.

TABLE VII: Application Timing Statistics.

	Latency Statistics (ms)				
Application	Mean	Min	Max	STDEV	STDEV/Mean
x264	36.4	8.80	174.0	15.3	0.420
bodytrack	96.6	86.5	107.2	3.60	0.037
swaptions	50.8	0.002	582.6	117.9	2.322
ferret	13.4	0.04	73.4	12.2	0.913
streamcluster	623.5	445.6	786.7	73.8	0.118
swish++	6.21	0.79	51.1	3.82	0.614
radar	53.9	10.4	66.7	16.3	0.302

default configuration. For clarity of presentation, we show only Pareto-optimal tradeoffs.

D. *Statistical Timing Characteristics*

Table VI shows the inputs for each benchmark. These benchmarks were not originally designed to provide predictable timing. We quantify this inherent unpredictability by measuring the latency of each job and computing the mean, minimum, maximum, standard deviation, and standard deviation over mean for all jobs in a benchmark. This data – summarized in Table VII – shows that our benchmarks have a range of natural behavior from low variance (natural predictability; *e.g.,* bodytrack) to high variance (widely distributed job latencies; *e.g.,* swaptions).

This timing variability further motivates the need for MEANTIME, which meets hard real-time guarantees even for such off-the-shelf applications. The inherent variability means that allocating resources for worst case execution time requires reserving resources that are not used much of the time – the worst case is typically far from the average case. MEANTIME adapts to this variability by allocating for the average case to save energy and using a (typically) small amount of approximation to meet the hard real-time deadlines.

V. Experimental Evaluation

This section evaluates MEANTIME's ability to provide timeliness and energy savings. We first measure both latency and deadline misses. Next, we quantify MEANTIME's energy savings and then the effect of approximation. Next, we show how MEANTIME reacts to both changes in user goals (*e.g.,* transitioning to perfect accuracy) and application workload phases. We end by measuring MEANTIME's overhead.

We compare MEANTIME's timeliness, energy, and accuracy to two existing approaches:

- **wcet**: meets hard real-time constraints by reserving resources sufficient for worst case latency and intelligently sleeping if a job finishes early [32, 38].
- **energy-aware**: uses state-of-the-art control and optimization techniques to allocate minimal energy resources for the current job [34, 42].

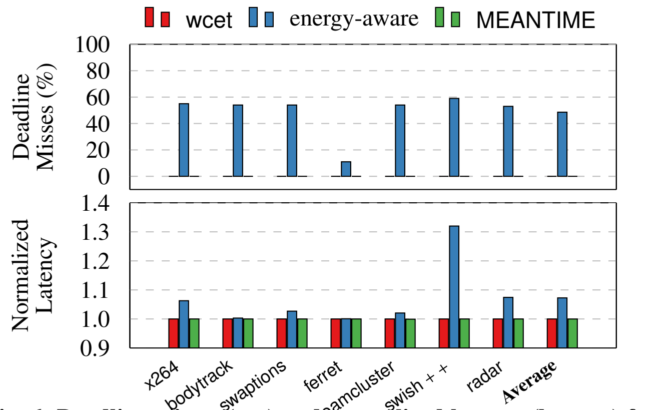

Fig. 6: Deadline misses (top) and normalized latency (bottom) for different resource allocation techniques.

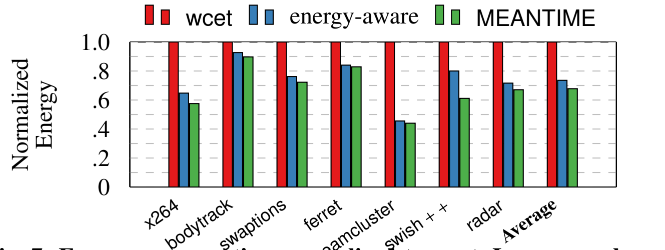

Fig. 7: Energy consumption normalized to wcet. Lower numbers represent reduced energy consumption.

To test MEANTIME, we deploy it on our test server and run each application with a target latency equal to its maximum latency (as reported in Table VII). We then measure, for each application, the number of missed deadlines and the average latency, energy, and accuracy for each job.

A. *Timing Properties*

Figure 6 shows the timing results for these experiments with deadline misses shown on the top and latency (normalized to the target) shown on the bottom. These results confirm that neither wcet nor MEANTIME miss deadlines, thus they provide hard real-time guarantees. The energy-aware approach does not provide hard guarantees, so it is not surprising that it misses deadlines. Instead, this approach supports soft real-time goals, reflected by the average latency results in the bottom of Figure 6. Overall, the energy-aware approach provides good average timing; it is just not as predictable as wcet and MEANTIME. These results confirm MEANTIME's ability to meet hard timing constraints.

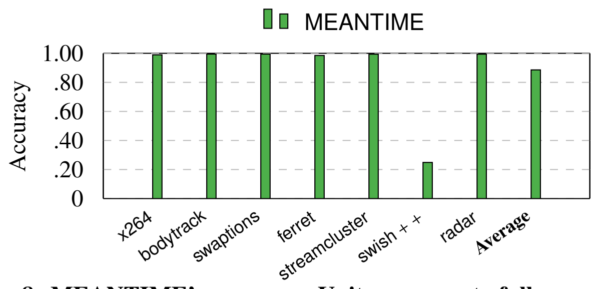

Fig. 8: MEANTIME's accuracy. Unity represents full accuracy.

B. Energy

Figure 7 shows the energy consumption of each benchmark normalized to wcet. We normalize so that results are comparable across benchmarks. The results show that the energy-aware approach saves significant energy compared to allocating for worst case execution time (confirming prior results [34, 38]). On average, MEANTIME consumes only 68% of the energy of allocating for worst case and aggressively sleeping. The POET approach, which we use as our representative energy-aware approach, has been shown to provide energy consumptions only a few percent above the optimal determined by an oracle that acts with perfect knowledge and no overhead [34]. Thus, we conclude that MEANTIME is also near optimal energy.

Some benchmarks exhibit greater energy savings than others. In general, two factors predict the energy savings compared to wcet. First, the greater the variance in timing, the greater the energy saving potential – if an application spends most of its time near worst case latency, then there is limited opportunity to reduce resource usage. Second, if an application's approximate configurations do not provide much speedup, then the potential for energy savings is also reduced. Comparing the timing statistics (Table VII) and the speedups from approximation (Table V) to the energy reduction in Figure 7 validates these observations.

Somewhat surprisingly, MEANTIME consumes slightly less energy than the energy-aware approach – 8% on average. While this seems counter-intuitive, the explanation is straightforward. Both approaches select the same resource configuration (as MEANTIME builds on prior energy-aware techniques). Because they both select the same resources, their average power is the same. MEANTIME, however, never violates timing constraints, so its latency is always less than or equal to the energy-aware approach. Thus, both approaches have the same power consumption, but MEANTIME's time is always less than or equal to energy-aware. Therefore, MEANTIME's energy (power times time) is always less than or equal to energy-aware. As the average latency for energy-aware approaches is very close to the target, this energy savings is small, but real. Such energy-aware approaches have already been shown to provide close to minimal energy [34, 42], so we conclude that MEANTIME meets its design goals of timeliness with near-minimal energy consumption.

C. Accuracy

Figure 8 shows MEANTIME's accuracy. These results are normalized to nominal application behavior, so unity represents full accuracy. The results show generally high accuracy. Six of the seven benchmarks have accuracies above 0.98 representing accuracy loss of less than 2%. The one exception is swish++, whose accuracy is 0.24.

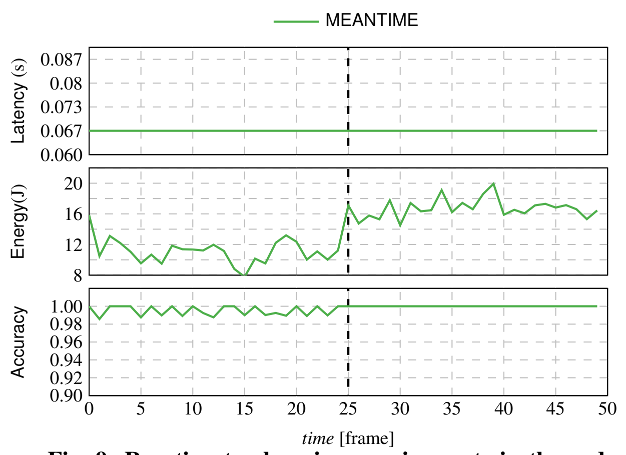

Fig. 9: Reacting to changing requirements in the radar.

For swish++ the low accuracy comes from a combination of factors. The first is the application's high variance (see Table VII). The second is the low speedup relative to the difference between minimum and maximum measured latency (see Table V). These two factors lead the application to spend most of its time in a severely reduced accuracy state. For swish++ this accuracy reduction results in fewer matches returned for a given search. The top (most relevant) matches are still returned, but less relevant ones are dropped. We emphasize that for users who do not want to lose this level of accuracy, they could simply specify fewer application knobs. The energy consumption would be higher, but that might be the tradeoff the user wants. The next section shows an example of an application transitioning from reduced accuracy to perfect accuracy to support changing user goals.

D. Adapting to Changing Requirements

This section demonstrate how MEANTIME reacts to changes in accuracy requirements. Specifically, we transition a running application from energy minimization to accuracy maximization. This demonstrates how MEANTIME can overcome the issue described in the previous section.

To demonstrate this capability, we extend the radar example from Section II. We now launch the radar application using MEANTIME to meet real-time goals and minimize energy consumption. After 25 frames we change the radar's goal to real-time with maximum accuracy. Practically, this transitions from MEANTIME to wcet resource allocation.

The results of this experiment are illustrated in Figure 9. The figure has three charts which show latency, energy, and accuracy as functions of time (measured in radar pulses). The vertical dashed lines represent the time when the acceptable approximation changes.

The results demonstrate that MEANTIME easily transitions between requirements. The first half of the input is processed exactly as in the example section. The second half transitions to a higher energy configuration, but with no approximation. This demonstrates that the accuracy loss incurred by MEANTIME is optional and can be eliminated as needed. In this sense, MEANTIME represents a strictly greater set of capabilities than wcet.

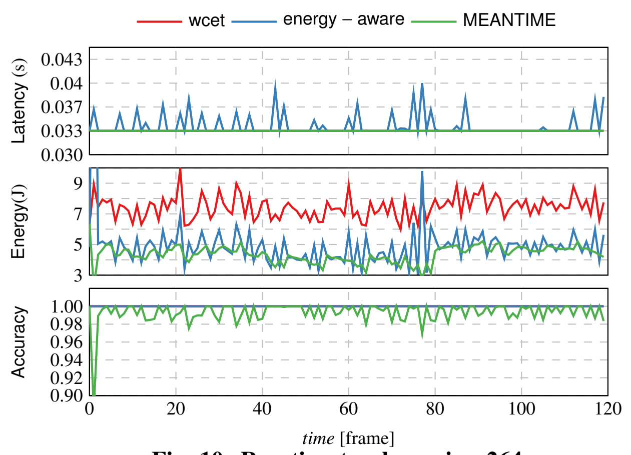

Fig. 10: Reacting to phases in x264.

E. Adapting to Phases

This section demonstrates how MEANTIME adapts to application phases, while maintaining hard real-time deadlines and energy efficiency. In this experiment, we run the x264 video encoder on an input consisting of three distinct video scenes. The difficulty (computational resources required) varies from frame to frame, but on average, the first scene is the hardest, the second scene is the easiest, and the third scene is close to the first, but with a lower variance in frame-to-frame performance. To demonstrate the benefits of MEANTIME we compare to wcet and energy-aware allocation.

The results of this experiment are shown in Figure 10. The charts show latency, energy, and accuracy as a function of time (measured in frames). Overall, these results are consistent with prior findings – MEANTIME achieves perfect timeliness, equivalent to wcet, but an energy consumption slightly better than the energy-aware approach. These results also demonstrate that MEANTIME provides predictable timing even as the application transitions through phases, something that prior energy-aware approaches which only support soft real-time cannot do [34].

Finally, these results show that the accuracy and energy are tailored to the application workload. During the middle scene (between frames 40 and 80) the resource demand is lessened and the average accuracy improves and the energy decreases. The first scene has an accuracy of .98 and averages 9.0 Joules/frame, the second scene has an accuracy of .99 and averages 7.6 Joules/frame, while the third scene has an accuracy of .98 and averages 9.4 Joules/frame. These results demonstrate that MEANTIME does not reduce accuracy needlessly, but tailors it to the workload requirements.

F. Overhead

MEANTIME's runtime is of constant – $O(1)$ – computational complexity. To make the system work in practice, however, it needs to know its own worst case latency so that it can ensure it does not interfere with the application's timing. Therefore, we measured the latency of the MEANTIME runtime on our system. To do this, we deploy the runtime with no application to manage and "dummy resources" which are allocated through the same system calls, but do not change the timing. We execute 1000 iterations of the runtime and measure the worst case latency. On our target system, the worst case latency is .02 μs. In practice, we account for this as part of the switching overhead as discussed in Section III.

G. Discussion

MEANTIME provides:

- the **timeliness** of conservatively allocating and sleeping.
- the **energy efficiency** of allocating for the current case.
- support for widely **varying timing** characteristics.
- support for **changing goals**, allowing applications to be dynamically reverted to full accuracy.
- support for applications with distinct **phases** and different processing requirements per phase.

Using MEANTIME, however, requires:

- sacrificing accuracy. While we believe there are many embedded applications that can make a small sacrifice in accuracy for a large gain in energy savings, it is clearly not appropriate for all applications.
- increased amount of worst case analysis. Typically, allocating for worst case execution time requires just a single analysis step to be performed on the whole application. To use MEANTIME, worst case analysis must be performed on the default application, as well as any approximate configurations that will be used in the runtime.

VI. Related Work

The problem of scheduling for timeliness and energy efficiency has been widely studied in the literature. A complete survey is beyond the scope of this paper, but we mention some related highlights. At the coarsest level, scheduling and resource allocation can be done to provide either hard (*e.g.,* [3, 7, 9, 13, 19, 32, 37, 39, 46, 50, 52]) or soft (*e.g.,* [10, 11, 23, 25, 29, 31, 34, 45]) timing guarantees, and in both cases it is beneficial to save as much energy as possible. While all these techniques differ in terms of the mechanisms they manipulate and the assumptions they make, we can draw some general conclusions. First, all techniques (whether hard or soft) manipulate *slack* – the time when the system is not busy. Some approaches scale voltage and frequency to reduce slack (*e.g.,* [13, 19, 25]); others manipulate processor sleep states (*e.g.,* [9, 32]). Still others work on general sets of resources specified at runtime (*e.g.,* [34, 47, 54, 61]). For this paper, however, the important distinction is that soft guarantees allow the system to be much more aggressive about eliminating slack as they have the freedom to occasionally miss deadlines. Hard real-time guarantees, in contrast, require the system to be conservative and never remove so much slack that timing could be violated. This conflict has been documented in the literature [16]. It has even been noted that, in mixed criticality systems, aggressively removing slack for non-critical jobs can cause critical jobs to violate timing [59].

MEANTIME does not overturn these prior results. Rather, MEANTIME is based on the observation that it may be possible to achieve timing and energy efficiency if we make sacrifices in a third dimension. Specifically, MEANTIME exploits the growing domain of approximate computing. Approximate applications trade accuracy for performance, power, energy, or other benefits. Such approaches include both static, compile-time analysis of tradeoffs [5, 51, 53] and dynamic, runtime support for tradeoff management [4, 6, 17, 30, 48, 55]. 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 [6], while Eon meets energy goals while maximizing accuracy [55]. Both Green and Eon use heuristic techniques for managing the tradeoff space. PowerDial [30], uses control theoretic techniques to provide performance guarantees while maximizing accuracy. PowerDial is the only technique that attempts to guarantee performance. Its control-theoretic guarantees may be appropriate for soft real-time, but it cannot provide hard real-time guarantees. None of these approaches combine hard real-time with energy reduction.

In contrast to these application-level approaches, many system-level designs work to reduce energy consumption by tailoring system resource usage. To reduce total system power, most of these approaches coordinate multiple resources [18]. For example, Meisner et al. propose coordinating CPU power states, memories, and disks to meet soft latency goals while minimizing power consumption [43]. Bitirgen et al. coordinate clockspeed, cache, and memory bandwidth in a multicore [14]. Still other approaches focus on managing general sets of system-level components [29, 34, 47, 54, 61]. Finally, recent approaches manage system resources to provide both real-time and temperature guarantees, but do not minimize energy [22]. In fact, these systems provide either hard real-time guarantees (making no claims about energy), or they provide soft real-time guarantees (enforced through a variety of mechanisms) with energy savings.

Cross-layer optimization combines application-level approximation and system-level resource allocation. In that sense, MEANTIME most resembles other cross-layer approaches. Early cross-layer systems were designed for media processing on mobile systems. For example, Flinn and Satyanarayanan build a framework for coordinating operating systems and applications to meet user defined energy goals [21]. This system trades application quality for reduced energy consumption. GRACE [60] and GRACE-2 [58] use hierarchy to provide soft real-time guarantees for multimedia applications, making system-level adaptations first and then application-level adaptations. Like GRACE-2, Agilos uses hierarchy, combined with fuzzy control, to coordinate multimedia applications and systems to meet a performance goal [40]. Maggio et al. propose a game-theoretic approach for a decentralized coordination of application and system adaptation which provides soft real-time guarantees [41]. xTune uses static analysis to model application and system interaction and then refines that model with dynamic feedback [36]. CoAdapt allows users to pick two out of three of performance, power, and accuracy; it then provides soft guarantees in those two dimensions while optimizing the third [27].

These prior cross-layer approaches coordinate application accuracy and system energy, but none provide the combination of hard real-time and energy reduction. *This combination is the unique contribution of MEANTIME.*

A number of the approaches listed above use feedback control to manage timing constraints [11, 22, 25, 27, 29, 34, 40–42, 56, 58, 61]. Control theory provides a formal framework for reasoning about the dynamic behavior of the system. Prior shows these techniques can meet soft real-time constraints while providing a solid formal foundation for reacting to dynamic events (like application workload changes). They are generally insufficient for hard real-time, however, as they work towards the common case. MEANTIME is one example of modifying a control system to maintain the benefits of its dynamics (reacting to changes), while augmenting it with the ability to meet hard real-time constraints.

VII. Conclusion

This paper presents MEANTIME, a runtime control methodology. Its unique contribution is using application approximation to provide both hard real-time guarantees and energy efficiency. While not appropriate for all applications, MEANTIME provides a large benefit for those that can reduce accuracy. Our experimental results verify the claims made in the paper's introduction: MEANTIME achieves the timeliness of allocating for worst case and the energy efficiency of allocating for current case (actually, MEANTIME does slightly better). The capability provided by MEANTIME is strictly greater than allocating for worst case and racing to sleep. Therefore, we believe this is an important contribution, which can greatly increase energy efficiency for a class of embedded applications that must meet hard real-time constraints but can sacrifice a small amount of accuracy.

References

[1] S. Albers. "Algorithms for Dynamic Speed Scaling". In: *STACS*. 2011, pp. 1–11.

[2] E. Amig, J. Gonzalo, J. Artiles, and F. Verdejo. "A comparison of extrinsic clustering evaluation metrics based on formal constraints". English. In: *Information Retrieval* 12.4 (2009), pp. 461–486.

[3] J. Anderson and S. Baruah. "Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms". In: *ICDCS*. 2004.

[4] 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.

[5] 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.

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

[7] M. Bambagini, M. Bertogna, and G. Buttazzo. "On the effectiveness of energy-aware real-time scheduling algorithms on single-core platforms". In: *ETFA*. 2014.

[8] N. Bansal, D. P. Bunde, H.-L. Chan, and K. Pruhs. "Average Rate Speed Scaling". In: *Algorithmica* 60.4 (2011).

[9] P. Baptiste. "Scheduling Unit Tasks to Minimize the Number of Idle Periods: A Polynomial Time Algorithm for Offline Dynamic Power Management". In: *SODA*. 2006.

[10] M. Becker, A. Schmidt, M. Orehek, and T. Nolte. "Saving energy by means of dynamic load management in embedded multicore systems". In: *SIES*. 2014.

[11] L. Bertini, J. Leite, and D. Mosse. "Statistical QoS Guarantee and Energy-Efficiency in Web Server Clusters". In: *ECRTS*. 2007.

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

[13] E. Bini, G. Buttazzo, and G. Lipari. "Minimizing CPU Energy in Real-time Systems with Discrete Speed Management". In: *ACM Trans. Embed. Comput. Syst.* 8.4 (July 2009), 31:1–31:23.

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

[15] S. Bradley, A. Hax, and T. Magnanti. *Applied mathematical programming*. 1977.

[16] G. C. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency: Predictability vs. Efficiency*. Springer, 2006.

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

[18] H. Cheng and S. Goddard. "SYS-EDF: a system-wide energy-efficient scheduling algorithm for hard real-time systems". In: *International Journal of Embedded Systems* 4.2 (2009).

[19] A. Dudani, F. Mueller, and Y. Zhu. "Energy-conserving Feedback EDF Scheduling for Embedded Systems with Real-time Constraints". In: *LCTES/SCOPES '02*. 2002.

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

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

[22] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos. "Feedback thermal control of real-time systems on multicore processors". In: *EMSOFT*. 2012.

[23] R. Guerra, J. C. B. Leite, and G. Fohler. "Attaining soft real-time constraint and energy-efficiency in web servers". In: *SAC*. 2008.

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

[25] J. Heo, P. Jayachandran, I. Shin, D. Wang, T. Abdelzaher, and X. Liu. "OptiTuner: On Performance Composition and Server Farm Energy Minimization Application". In: *IEEE Transactions on Parallel and Distributed Systems* 22.11 (2011).

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

[27] H. Hoffmann. *CoAdapt*. Tech. rep. TR-2014-12. University of Chicago, Dept. of Comp. Sci., 2014.

[28] H. Hoffmann, A. Agarwal, and S. Devadas. "Selecting Spatiotemporal Patterns for Development of Parallel Applications". In: *IEEE Trans. Parallel Distrib. Syst.* 23.10 (2012), pp. 1970–1982.

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

[30] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. "Dynamic Knobs for Responsive Power-Aware Computing". In: *ASPLOS*. 2011.

[31] 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), pp. 444 –458.

[32] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. Buttazzo. "Adaptive Dynamic Power Management for Hard Real-Time Systems". In: *RTSS*. 2009.

[33] C. Imes and H. Hoffmann. "Minimizing energy under performance constraints on embedded platforms: resource allocation heuristics for homogeneous and single-ISA heterogeneous multi-cores". In: *SIGBED Review* 11.4 (2014), pp. 49–54.

[34] 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.

[35] S. Irani, S. Shukla, and R. Gupta. "Algorithms for Power Savings". In: *ACM Trans. Algorithms* 3.4 (Nov. 2007).

[36] 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).

[37] F. Kong, Y. Wang, Q. Deng, and W. Yi. "Minimizing Multi-resource Energy for Real-Time Systems with Discrete Operation Modes". In: *ECRTS*. 2010.

[38] E. Le Sueur and G. Heiser. "Slow Down or Sleep, That is the Question". In: *USENIX ATC*. 2011.

[39] Y.-H. Lee, K. Reddy, and C. Krishna. "Scheduling techniques for reducing leakage power in hard real-time systems". In: *ECRTS*. 2003.

[40] 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).

[41] M. Maggio, E. Bini, G. C. Chasparis, and K.-E. Årzén. "A Game-Theoretic Resource Manager for RT Applications". In: *ECRTS*. 2013.

[42] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation". In: *IEEE Trans. on Control Systems Technology* 21.1 (2013).

[43] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. "Power management of online data-intensive services". In: *ISCA* (2011).

[44] 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.

[45] R. Nassiffe, E. Camponogara, and G. Lima. "Optimizing QoS in Energy-aware Real-time Systems". In: *SIGBED Rev.* 10.2 (July 2013).

[46] R. Racu, A. Hamann, R. Ernst, B. Mochocki, and X. S. Hu. "Methods for power optimization in distributed embedded systems with real-time requirements". In: *CASES*. 2006.

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

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

[49] E. Rotem, A. Naveh, D. R. amd Avinash Ananthakrishnan, and E. Weissmann. "Power management architecture of the 2nd generation Intel Core microarchitecture, formerly codenamed Sandy Bridge". In: *Hot Chips*. Aug. 2011.

[50] S. Saha, J. S. Deogun, and Y. Lu. "Adaptive energy-efficient task partitioning for heterogeneous multi-core multiprocessor real-time systems". In: *HPCS*. 2012.

[51] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. "EnerJ: approximate data types for safe and general low-power computation". In: *PLDI*. 2011.

[52] A. Shrivastava, E. Earlie, N. Dutt, and A. Nicolau. "Aggregating Processor Free Time for Energy Reduction". In: *CODES+ISSS*. 2005.

[53] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. "Managing performance vs. accuracy trade-offs with loop perforation". In: *ESEC/FSE*. 2011.

[54] 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).

[55] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. "Eon: a language and runtime system for perpetual systems". In: *SenSys*. 2007.

[56] M. H. Suzer and K. Kang. "Predictive Thermal Control for Real-Time Video Decoding". In: *ECRTS*. 2014.

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

[58] 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).

[59] M. Volp, M. Hahnel, and A. Lackorzynski. "Has energy surpassed timeliness? Scheduling energy-constrained mixed-criticality systems". In: *RTAS*. 2014.

[60] W. Yuan and K. Nahrstedt. "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems". In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), pp. 149–163.

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