MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing

Anne Farrell *Henry Hoffmann*
Department of Computer Science, University of Chicago
{amfarrell, hankhoffmann}@cs.uchicago.edu

Abstract

Energy efficiency and timeliness (*i.e.,* predictable job latency) are two essential – yet opposing – concerns for embedded systems. 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 system that delivers hard latency guarantees and energy-minimal resource usage through small accuracy reductions. We test MEANTIME on a real Linux/ARM system with six applications. Overall, we find that MEANTIME never violates real-time deadlines and sacrifices a small amount (typically less than 2%) of accuracy while reducing energy to 54% of a conservative, full accuracy approach.

1 Introduction

Embedded 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 timing 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 [20] or sleep states [31]), this strategy is less energy-efficient than allocating for the current case, a fact which has been demonstrated both analytically [1, 8, 35] and empirically [17, 32, 38, 45].

Recent research on *approximate computing* examines applications that can trade *accuracy* for reduced energy consumption [4, 6, 18, 21, 29, 55]. These approximate computations open a third dimension for optimization, making it possible to simultaneously trade timing, energy, and accuracy [26]. Many embedded computations are well-suited for approximation as they 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. This process is often done statically; *e.g.,* by using fixed point arithmetic instead of floating point. While approximate computing frameworks produce a large trade-off space [4, 29, 51], they provide (at best) soft timing guarantees, and are thus unsuitable for hard real-time constraints. *This paper addresses the challenge of meeting both hard timing constraints and low energy through careful application of approximate computing.*

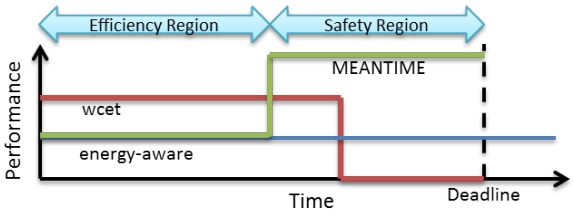

Figure 1: Conceptual model of MEANTIME compared to worst-case and energy-aware resource allocation.

Specifically, we develop MEANTIME[1], which couples a state-of-the-art resource allocator [42] – that optimizes for the current job, sometimes missing deadlines – with a novel *governor* which monitors timing and reconfigures the application to avoid any timing violations. The resource allocator uses control theory to allocate for the current case, an approach which, 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

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

Table 1: Embedded Platform Resources.

Platform	Processor	Cores	Core Types	Speeds (GHz)	Configurations
ODROID-XU3	Samsung Exynos5 Octa	8	2 (A15 & A7)	.2–2.0 (A15) .2-1.4 (A7)	128

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 (no performance) until the deadline. Energy-aware allocation estimates the resource needs of the current job, but it may miss the deadline due to job timing variance.

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). Note that switching to an approximate configuration does not require computation to be restarted; rather there are times while an application is running that it can be switched to an approximate configuration before the deadline. Approximation allows MEANTIME to potentially have even higher performance in the safety region than allocating for worst case execution time, as shown in Figure 1. MEANTIME's key idea is using timing analysis and approximate computation to determine the efficiency and safety regions, ensuring both energy efficiency and timeliness.

We implement MEANTIME on a Linux/ARM system. We test its timing guarantees, energy, and accuracy for six different applications, including media, image, signal processing, and financial analysis. These applications were not originally designed to provide any timing predictability, still MEANTIME achieves:

- **Efficacy with a range of behaviors**: The benchmarks include both high and low variance job latencies (see Section 4.3).
- **Predictable timing**: MEANTIME never misses a deadline for any benchmark (see Section 5.1).
- **Energy Savings**: MEANTIME requires only 46% of the energy of allocating for worst case and aggressively sleeping the system (see Section 5.2). These energy savings are comparable to a state-of-the-art energy-aware approach that provides only soft timing guarantees but near-optimal energy savings.
- **High Accuracy**: MEANTIME achieves accuracies that are typically very close to the nominal behavior (*i.e.,* almost no accuracy loss, see Section 5.3).
- **Adaptability**: MEANTIME automatically reacts to changing user goals (see Section 5.4) and fluctuations in application workload (see Section 5.5).

MEANTIME is not designed for all embedded applications, but for those that 1) have viable performance/accuracy trade-offs, 2) must satisfy hard real-time constraints and minimize energy consumption despite large fluctuations in application workload, and 3) have progress indicators and models of completion. We believe many embedded applications meet these criteria (as evidenced by our use of existing benchmarks).

MEANTIME makes the following 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 showing mean energy reductions of over $2\times$ compared to allocating for worst case.

2 Motivational Example

We motivate MEANTIME's combination of hard timing guarantees and energy efficiency through a radar processing example similar to what might be found on an unmanned autonomous vehicle (UAV). The radar must process frames with a strict latency or the system cannot respond to external events. The UAV, however, is energy limited. Radar processing is amenable to approximate computation – it can trade reduced signal-to-noise ratio for performance.

The radar processing consists of four *blocks* of processing. The first block (LPF) performs a low-pass filter to eliminate high-frequency noise. The second block (BF) does beam-forming, steering the 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 (CFAR) detection, which identifies targets. Several parameters control the tradeoff between signal-to-noise ratio (SNR) and performance (see Section 4). The radar processes frames in batches of 8. Each batch is one job.

Our experimental platform is an 8-core Linux/ARM big.LITTLE system, which has four high power, high performance "big" cores, and four low power, low performance "LITTLE" cores. The radar application is parallelized, so that each processing block can be executed

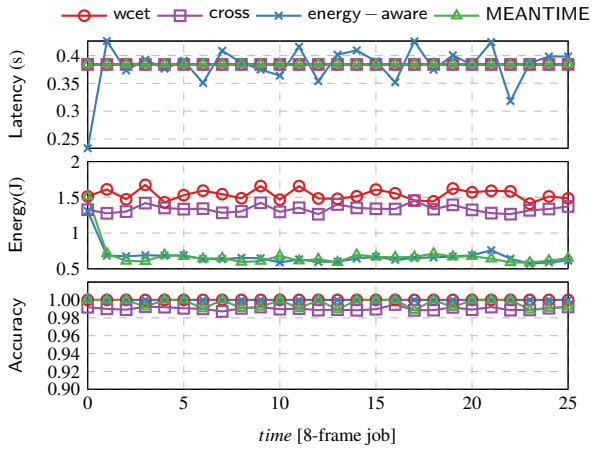

Figure 2: Comparison of techniques for the radar.

across multiple threads. Different resource configurations achieve different power/performance tradeoffs; Table 1 lists available configurations.

We run the radar application on the ARM with all resources allocated and empirically determine the worst case latency for a single radar frame (this is technically worst *observed* latency). Additionally, we report the minimum, mean, and standard deviation over mean latency in Table 2.

Table 2: Radar timing.

Latency	Measurement (s)
Mean	0.032
Minimum	0.031
Maximum	0.048
STDEV/Mean	0.025

The timing data demonstrates the classic conflict between timeliness and energy efficiency. For timeliness, we can allocate for worst-case execution time. The table, however, shows that the average latency is very close to the minimum latency. So most jobs are not close to worst case. If a job finishes early (*i.e.,* it is not a worst case), then we can simply sleep (or idle) the processor until the next job is available [17, 38, 45]. To allocate for energy efficiency and meet average case timing, 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 3 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 three other approaches: allocating resources for worst case (wcet), allocating both resources and algorithm parameters for worst case based on existing *cross-layer* approaches (cross) [21, 59], and allocating for average case using an *energy-aware* resource allocator based on control theory [33, 42]. Figure 2 shows the results. Each chart shows time (measured in 8-frame jobs) on the x-axis. The top chart shows latency compared to the target latency (0.384 seconds per 8-frame job, the worst latency measured on our system). The middle chart shows the energy per job in Joules. The bottom chart shows the SNR normalized to the default configuration; *i.e.,* unity represents no accuracy loss.

The results demonstrate that MEANTIME achieves both timeliness and energy efficiency. Indeed, MEANTIME achieves the same timing as allocating for worst case, the same energy consumption as the energy-aware system, and the same accuracy as the cross-layer approach. All approaches except for the energy-aware approach have flat latency curves because they always meet the deadline (possibly finishing early and idling until the next job starts) – these techniques are hard to distinguish in the latency chart because all three are on top of each other.. Specifically, the energy-aware approach consumes 20.2 Joules, allocating for wcet consumes 46.1 Joules, the cross-layer approach consumes 36.7 Joules, and MEANTIME consumes 20.3 Joules. These results come at the cost of a small amount of accuracy; in this case the SNR falls to 99.7% of the original application.

3 The MEANTIME Framework

This section provides the technical details on MEANTIME, illustrated in Figure 3. 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 computes the minimal energy set of resources needed to meet that target. The governor receives, from the user, timing requirements and application's timing analysis in both its default and approximate configurations. The user has complete control over 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 governor calculates the *efficiency* and *safety* regions. The efficiency region is the time to spend in the application's default configuration using the resource configuration suggested by the controller. The safety region is the time to spend in an approximate configuration, ensuring the job meets its 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 gover-

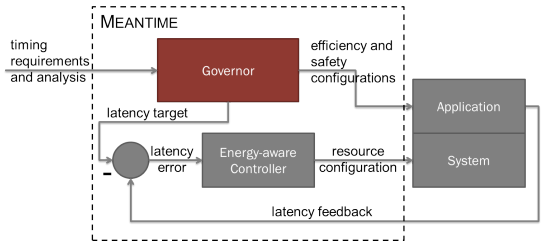

Figure 3: Overview of the MEANTIME approach.

nor with existing control techniques. The remainder of this section provides an overview of energy-aware control schemes, then describes the governor.

3.1 Approximate Computing

Many approximate computing frameworks exist (see Section 6) that expose application-level 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 a radar, accuracy is signal-to-noise ratio. In a search engine, it is the precision and recall of results returned. We discuss the accuracy metrics used for our test applications in Section 4. 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.

3.2 Control for Energy Efficiency

Control theory provides formal guarantees about how a controlled system responds to dynamic events [24, 60]. Control theoretic techniques have been used to meet soft real-time deadlines while minimizing resource use. Examples can be found controlling latency in multi-tier webservers [11, 30] and in media processing [57, 59].

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, 30, 33, 59].

The drawback is that these techniques, by themselves, cannot provide both hard real-time latency and energy efficiency. One might attempt to use control theoretic techniques to meet hard real-time constraints by reducing the target latency, but this does not guarantee hard real-time and may not provide optimal energy savings. 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 [33]. 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.

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

While timing information may vary, we only require information for the worst case, which is conservative and does not vary. This section shows how the governor derives the efficiency and safety regions from these inputs.

3.3.1 Notation

Table 3 summarizes the notation used in this section. We assume an application comprised of many jobs, where each 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. Application speedups

Table 3: 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}

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; *i.e.,* the minimum speedup that will be observed from approximation.

3.3.2 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:

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

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

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

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.

3.3.3 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.

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.

3.3.4 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 [29, 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 opti-

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

mization problem:

$$maximize \sum_i \frac{t_s^i}{t} \cdot a^i \quad (9)$$

$$s.t.$$

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

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

$$t_s^i,\ t_e \geq 0 \quad (12)$$

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 have 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.

3.3.5 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:

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

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

Table 4: System configurations.

Configuration	Settings	Max Speedup	Max Powerup
big cores	4	4.52	2.00
big core speeds	19	10.23	10.42
LITTLE cores	4	4.52	1.32
LITTLE core speeds	13	7.11	2.62
idle	-	0	.6

3.4 MEANTIME Summary

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.

4 Experimental Setup

4.1 Hardware Platform

Our hardware platform is an ODROID-XU3 from HardKernel. It runs Ubuntu Linux 14.04 using a modified kernel 3.10.58+. We use the `taskset` utility for managing processor core assignment and `cpufrequtils` for managing DVFS settings. This system supports five configurable resources that alter performance and power tradeoffs. Additionally, it has an extremely low-power *idle* state (about 0.1 Watt).

Table 4 summarizes system resources, showing the maximum increase in speed and power measured on the machine for any benchmark. Embedded INA-231 power sensors [34] provide power data for the big Cortex-A15 cluster, the LITTLE Cortex-A7 cluster, the DRAM and the GPU. The system sleeps at .1 Watts. The maximum measured power consumption is just under 6 Watts. *All energy and power numbers reported in this paper consider total usage as measured with the INA-321.*

4.2 Applications

We use six benchmark applications: x264, bodytrack, swaptions, ferret, and streamcluster (from the PARSEC

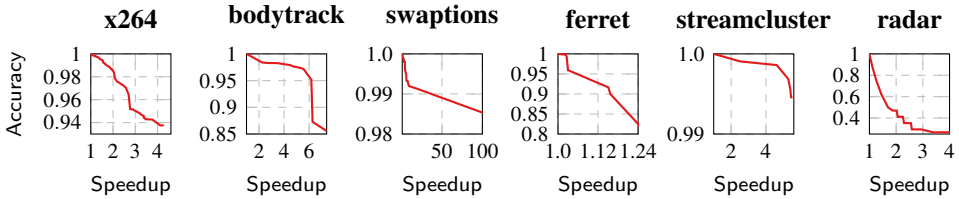

Figure 4: Speedup and accuracy tradeoffs for test applications.

Table 5: 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	30.24
streamcluster	16	3.82	54.8
radar	512	3.95	73.4

Table 6: 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
radar	radar pulses	100 pulses

benchmark suite [12]); as well as a radar processing application [28]. We use the PowerDial framework to modify all seven benchmarks so they support dynamic approximation [29]. PowerDial turns static configuration parameters into a data structure controlling the application's runtime behavior. PowerDial also instruments the applications, providing latency feedback for every outer loop iteration [29].

We measure application accuracy as a normalized distance from full accuracy. This is a standard metric allowing comparison across applications [49]. 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 5 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. *None of these applications were originally intended to be run with hard-timing constraints making them a real test of the proposed technique.*

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 trades the work performed to find redundancy for the amount of redundancy identified. The total number of distinct application-level 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 space. 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. In total, the application supports 200 different configurations.

ferret: This application performs 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 support 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 parameters which trade clustering accuracy (measured using the B^3 score [2]) and clustering latency. The first changes the number of iterations used in the approximation, the second changes the distance metric used to assign a sample to a cluster. Together, they 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 parameter, with 100 settings, controlling the number of simulations per swaption.

radar: This application is the front-end of a radar signal processor and it turns raw antenna data into a target list. The application supports four different parameters that tradeoff signal-to-noise ratio (SNR) [4]. The first

[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 tar-

Table 7: Application Timing Statistics.

	Latency Statistics (s)			
Application	Mean	Min	Max	STDEV/Mean
x264	1.33	0.14	2.97	0.59
bodytrack	0.75	0.64	0.92	0.11
swaptions	0.26	0.01	4.32	1.96
ferret	0.44	0.19	1.09	0.30
streamcluster	0.06	0.03	0.09	0.23
radar	0.03	0.03	0.05	0.03

two change the strength of the low-pass filter. The third changes the number of distinct directions the phased array antenna can "look." The fourth changes the range resolution. The radar can enter 512 separate configurations by changing these parameters.

All applications expose timing/accuracy tradeoffs. We illustrate the tradeoff spaces in Figure 4. Each chart's x-axis shows speedup and y-axis shows the resulting accuracy. Speedup and accuracy are both reported normalized to the default configuration. For clarity, we show only Pareto-optimal tradeoffs. Note that the y-axes show the range of accuracy tradeoffs for each application, and the x-axes show the range of possible speedups. Each application has a different range of both speedup and accuracy, so the axes also have different ranges.

4.3 Statistical Timing Characteristics

Table 6 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, and standard deviation over mean for all jobs in a benchmark. This data – summarized in Table 7 – 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.

5 Experimental Evaluation

This section evaluates MEANTIME's ability to provide hard timing guarantees 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 the following approaches:

- **wcet**: meets hard real-time constraints by reserving resources sufficient for worst case latency and intelligently sleeping if a job finishes early [31]. We do not have a tool that can accurately estimate worst case latency on our test platform, so we use worst observed latency as a proxy.
- **PowerDial**: uses control theory to adjust application-level parameters and meet performance constraints with maximum accuracy [29]. PowerDial, however, cannot adjust system-level resource usage, so it cannot proactively lower system energy consumption. In fact, the only way to proactively achieve energy savings with PowerDial is to trade increased performance for reduced accuracy. The increased performance allows each job to finish earlier and then the system can spend more time in the idle state.
- **cross**: Extends existing cross-layer techniques (*e.g.,* [21, 59]) which combine application accuracy and system resource management. These prior works do not provide hard guarantees, but we extend them in a simple way by using worst case timing for both the application accuracy and system resource usage. This approach is much simpler than MEANTIME.
- **energy-aware**: uses state-of-the-art control and optimization techniques to allocate minimal energy resources for the current job [33, 42].
- **optimal**: is the true optimal energy-aware approach if we knew the future and the could instantly configure the system for each task. We compute the optimal by running each application in each system configuration and logging the latency of each job within the application. We then post-process these logs to determine the minimal energy configuration for each job that would have met the latency goal. This approach is obviously not practical, but we believe it represents the true energy savings available.

To test MEANTIME, we deploy it on our ODROID platform and run each application with a target latency equal to its maximum latency (as reported in Table 7). We measure missed deadlines and the average latency, energy, and accuracy for each job.

gets harder to detect. All configurations used in this paper still detect all targets with more than 10dB of margin.

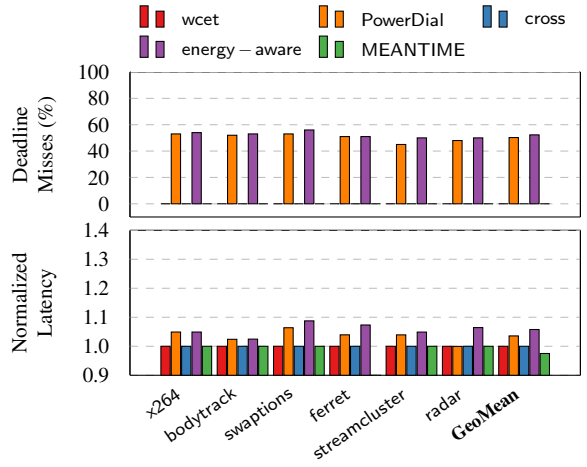

Figure 5: Deadline misses (top) and normalized latency (bottom) for different resource techniques.

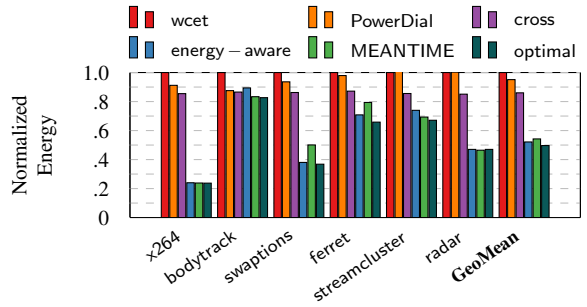

Figure 6: Energy consumption normalized to wcet. Lower numbers represent reduced energy consumption.

5.1 Timing Properties

Figure 5 shows the timing results with deadline misses on the top and latency (normalized to the target) on the bottom. These results confirm that none of wcet, cross, nor MEANTIME miss deadlines, thus they provide hard real-time guarantees. This agrees with the mathematical basis for MEANTIME established in Section 3, which should never miss deadlines as long as it is provided with accurate (or conservative) worst case timing information. PowerDial and energy-aware approach do not provide hard guarantees, so it is not surprising that they miss deadlines. Instead, these approaches support soft real-time goals, reflected by the average latency results in the bottom of Figure 5. Overall, these approaches provides good average timing; they are just not as predictable as wcet and MEANTIME.

5.2 Energy

Figure 6 shows each benchmark's energy consumption 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 wcet (confirming prior results [17, 32, 38]). By geometric mean, MEANTIME consumes only 54% of the energy of the wcet approach. MEANTIME's energy savings is comparable to the energy-aware approach, which consumes only 52% of wcet's energy, while the optimal approach consumes just 49% of wcet's energy.

The cross approach presented here consumes 86% of wcet's energy, which is a significant savings, but not close to MEANTIME. This cross layer approach is conservative in both system resource usage and application configuration, while MEANTIME is aggressive in resource allocation and conservative (for timing) in application configuration. The result show MEANTIME's combination achieves much better energy reductions. Similarly, PowerDial's energy savings is small – only about a 5% reduction – because PowerDial cannot proactivel alter system energy consumption. This result for PowerDial confirms previous work demonstrating that significant additional energy savings arises by coordinating application behavior with system resource usage [27].

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 7) and the speedups from approximation (Table 5) to the energy reduction in Figure 6 validates these observations. For instance, swaptions has much higher timing variance and speedups from approximation than bodytrack, and consequently, it exhibits better energy savings.

MEANTIME achieves large energy savings because allocating for worst case and idling after the job completes is one of the worst things to do on these processors. One study demonstrates that it is best to keep low-power multicores busy as much as possible [17]. Another shows that the Exynos Octa processor is more energy efficient at lower clockspeeds (and using the LITTLE cores) [32]. MEANTIME uses a control system in an attempt to keep the cores as busy as possible (and use the LITTLE cores as much as possible). On this architecture, the policy results in tremendous savings. On a another system where racing-to-idle is energy efficient, the MEANTIME approach would provide no benefit. We believe, however, that future embedded architectures will tend to look more like the processor used in this study and present a range of performance and energy tradeoffs.

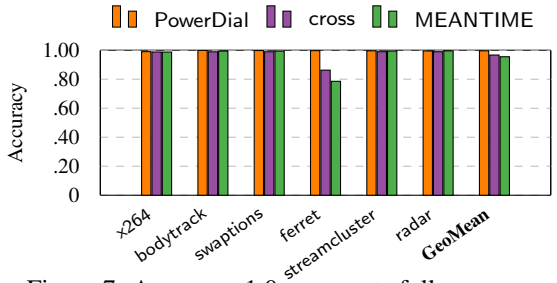

Figure 7: Accuracy, 1.0 represents full accuracy.

5.3 Accuracy

Figure 7 shows accuracy for PowerDial, cross, and MEANTIME. These results are normalized to nominal application behavior, so unity represents full accuracy. The results show generally high accuracy. Five of the six benchmarks have accuracies above 0.98 representing accuracy loss of less than 2%. The one exception is ferret, whose accuracy is 0.78. This is also the application for which the cross layer approach provides clear benefit.

While it may be hard to see, the accuracy for PowerDial is always higher than for MEANTIME. PowerDial really represents a technique for exposing and controlling performance/accuracy tradeoff spaces. In this case, we have configured PowerDial to produce the highest possible accuracy. An alternative could run in the lowest accuracy configuration to produce the highest possible performance, finish each task as early as possible, and spend the maximum time in the idle state. Doing so would produce the lowest accuracy output, but have significant energy savings. Prior work shows that the energy savings available in this case is always worse than coordinating between application and system [27].

Ferret's low accuracy is due to several factors. The first is the application's high variance (see Table 7). While this may at first appear beneficial, as it provides more room for high energy savings, adding the second factor creates problems. The second factor is the low speedup relative to the difference between minimum and maximum measured latency (see Table 5). The combination of high-variance (which is generally good for MEANTIME) and low speedup available at the application-level lead the application to spend much of its time in a reduced accuracy state. In contrast, because it is conservative in resource usage, cross spends less time in the reduced resource states that put more pressure on the application. MEANTIME's 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 transitioning from reduced accuracy to perfect accuracy to support changing user goals.

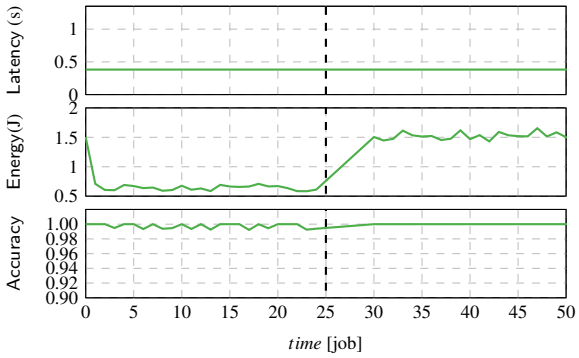

Figure 8: MEANTIME reacting to changing radar accuracy goals.

In general, for all approaches the accuracies are so high because very little speedup is needed most of the time; *i.e.,* most applications worst case behavior is far from the normal case so many jobs terminate in the safety zone. From our calculations of the true optimal energy savings available, we know the speedup required to meet the latency target for each job in each application. Table 8 shows the geometric mean of required speedup for each application across all its jobs. For most applications, the mean speedups required are quite small compared to the available speedups at the largest accuracy loss (compare Tables 5 and 8), so the accuracy losses are correspondingly small. The one exception is ferret, which has an achievable speedup of about 1.24×, but the mean required speedup is close to 1.14. This comparison helps explain ferret's lower accuracy.

Table 8: Required Speedup.

Application	Speedup
x264	1.18
bodytrack	1.06
swaptions	1.19
ferret	1.14
streamcluster	1.09
radar	1.03

5.4 Adapting to Changing Requirements

We transition a running application from energy minimization to accuracy maximization. To demonstrate this capability, we extend the radar example from Section 2. We now launch the radar application using MEANTIME to meet real-time goals and minimize energy consumption. After 25 jobs we change the radar's goal to real-time with maximum accuracy.

Figure 8 shows the results. The figure has three charts which show latency, energy, and accuracy as functions of time (measured in jobs). The vertical dashed lines represent the time when the acceptable approximation changes. The first half of the input is processed exactly as in the example section. The second half transitions

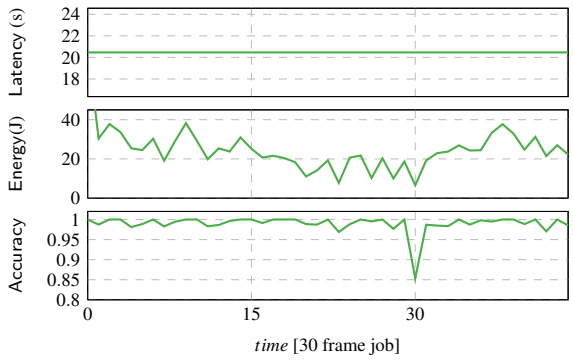

Figure 9: MEANTIME reacting to phases in x264.

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.

5.5 Adapting to Phases

We run the x264 video encoder on an input consisting of three distinct video scenes (each of 450 frames). The difficulty (computational resources required to meet the target latency) 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.

Figure 9 shows the results with latency, energy, and accuracy as a function of time (measured in 30 frame jobs). The latency target here is high, as it is hard to encode HD frames at high-quality on the test platform. MEANTIME achieves perfect timeliness, equivalent to wcet. These results 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 [33]. Finally, these results show that the accuracy and energy are tailored to the application workload. During the middle scene (between jobs 15 and 29) the resource demand is lessened and the average accuracy improves slightly and the energy decreases. These results demonstrate that MEANTIME does not reduce accuracy needlessly, but tailors it to the workload requirements.

5.6 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. To measure MEANTIME's latency, 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 as approximately 100 μs. In practice, we account for this as part of the switching overhead as discussed in Section 3.

6 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, 20, 31, 37, 39, 47, 50, 52]) or soft (*e.g.,* [10, 11, 23, 25, 30, 33, 46]) 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, 20, 25]); others manipulate processor sleep states (*e.g.,* [9, 31]). Still others work on general sets of resources specified at runtime (*e.g.,* [33, 44, 48, 54, 60]). 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 conservative allocation and never remove so much slack that timing might be violated [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 [58].

MEANTIME does not overturn these prior results. Rather, MEANTIME is based on the observation that we can 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, 18, 29, 49, 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 systems tailor behavior online. 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 [29], uses control theoretic techniques to provide performance guarantees while maximizing accu-

racy. 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 reduce energy consumption by tailoring resource usage. To reduce total system power, most of these approaches coordinate multiple resources [19]. 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 [33, 44, 48, 54, 60]. 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 [59] and GRACE-2 [57] 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 [26].

Prior cross-layer approaches coordinate application accuracy and system energy, but none provide both hard real-time and energy minimization. Our empirical results show that a straightforward extension of existing cross layer approaches can meet hard real-time deadlines and saves energy compared to allocating for worst case and maximum accuracy, but it still consumes much more energy than MEANTIME (see Figure 6). *The combination of hard timing guarantees with the energy efficiency of an optimistic soft-timing system is the unique contribution of MEANTIME.*

A number of the approaches listed above use feedback control to manage timing constraints [11, 22, 25–27, 33, 40–42, 56, 57, 60]. Control theory provides a formal framework for reasoning about the dynamic behavior of the system. Prior work 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.

7 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.

Acknowledgments The effort on this project is funded by the U.S. Government under the DARPA PERFECT program, the DARPA BRASS program, by the Dept. of Energy under DOE DE-AC02-06CH11357, by the NSF under CCF 1439156, and by a DOE Early Career Award. Additional funding for Anne Farrell comes from a GAAN fellowship.

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. ISSN: 1386-4564. DOI: 10.1007/s10791-008-9066-8. URL: http://dx.doi.org/10.1007/s10791-008-9066-8.

[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] A. Carroll and G. Heiser. "Mobile multicores: use them or waste them". In: *Operating Systems Review* 48.1 (2014), pp. 44–48. DOI: 10.1145/2626401.2626411. URL: http://doi.acm.org/10.1145/2626401.2626411.

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

[19] 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).

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

[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. ISBN: 047126637X.

[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. “CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems”. In: *ECRTS*. 2014.

[27] H. Hoffmann. “JouleGuard: Energy Guarantees for Approximate Applications”. In: *SOSP*. 2015.

[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. DOI: 10.1109/TPDS.2011.298. URL: http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.298.

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

[30] 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.

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

[32] 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. DOI: 10.1145/2724942.2724950. URL: http://doi.acm.org/10.1145/2724942.2724950.

[33] 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.

[34] T. Instruments. http://www.ti.com/product/ina231.

[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. URL: http://dl.acm.org/citation.cfm?id=2002181.2002197.

[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] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann. “A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints”. In: *ASPLOS*. 2015.

[45] 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. ISBN: 1-58113-483-5. DOI: 10.1145/514191.514200. URL: http://doi.acm.org/10.1145/514191.514200.

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

[47] 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.

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

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

[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] 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).

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

[59] 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.

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