JouleGuard: Energy Guarantees for Approximate Applications

Henry Hoffmann

Department of Computer Science, University of Chicago

hankhoffmann@cs.uchicago.edu

Abstract

Energy consumption has become a major constraint in computing systems as it limits battery life in mobile devices and increases costs for servers and data centers. Recently, researchers have proposed creating approximate applications that can trade accuracy for decreased energy consumption. These approaches can guarantee accuracy or performance and generally try to minimize energy; however, they provide limited guarantees of energy consumption. In this paper, we build on prior work in approximate computing to create JouleGuard: a runtime control system that coordinates application and system to provide control theoretic formal guarantees of energy consumption, while maximizing accuracy. These energy guarantees will aid any user who has an energy budget (e.g., battery lifetime or operating cost) and must achieve the most accurate result on that budget. We implement JouleGuard and test it on a Linux/x86 server with eight different approximate applications created from two different frameworks. We find that JouleGuard respects energy budgets, provides near optimal accuracy, and adapts to phases in application workload. JouleGuard is designed to be general with respect to the applications and systems it controls, making it a suitable runtime for a number of approximate computing frameworks and languages.

1. Introduction

Energy consumption is crucial to the full spectrum of computing systems from mobile, where it determines battery life, to supercomputing, where it increases costs. To address energy use, a number of approaches create *approximate* applications that reduce the accuracy of their results in exchange for other benefits, particularly reduced energy consumption (see Sec. 2.1). At the same time, system-level approaches create *energy-aware* systems, which trade delivered performance for reduced energy (see Sec. 2.2).

The combination of approximate applications and energy-aware systems creates a rich optimization space. Several *cross-layer* approaches coordinate application and system to take advantage of the opportunities that arise in the larger tradeoff space (see Sec. 2.3). Examples include GRACE-2 [43] and Agilos [25], both of which guarantee performance, while minimizing energy consumption.

Minimizing energy, however, usually coincides with running as fast as possible – thus, operating at the extreme end of the application's performance/accuracy tradeoff space. This situation creates a dilemma for users who have an *energy budget* – they do not want minimum energy consumption, but rather the maximum accuracy possible within the budget. For example, a mobile user on an important video conference needs the battery to last the duration of the call while maximizing quality.

This paper extends existing cross-layer management approaches to create JouleGuard, a runtime framework that coordinates approximate applications and energy-aware systems to provide energy guarantees while maximizing application accuracy. JouleGuard's runtime takes an energy goal and dynamically configures application and system to ensure that the goal is met and application accuracy is near optimal for that energy consumption. *The key insight in the JouleGuard approach is that this complicated multidimensional optimization problem can be split into two subproblems: maximizing system energy efficiency and dynamically managing application accuracy/performance tradeoffs.* These subproblems are dependent, but JouleGuard is provably robust despite the dependences. This robustness comes from a combination of machine learning and control theoretic techniques. JouleGuard is general with respect to both the approximate applications and energy-aware systems it coordinates, making it compatible with multiple frameworks supporting approximate programming.

We implement JouleGuard in C and test it on three hardware platforms (a mobile, desktop, and server system – each with different performance/power tradeoffs) running eight different approximate applications (four built with PowerDial [17] and four built with Loop Perforation [39]). Our results show:

- **Stability and Convergence:** JouleGuard quickly converges to a given energy goal with low error. On average, across all applications, all machines, and a number of energy goals JouleGuard achieves an average error of less than 3%.
- **Optimality:** JouleGuard converges to the energy goals with near optimal accuracy. On average, for all applications, systems and goals, JouleGuard is within 97% of true optimal accuracy.

2. Background and Related Work

2.1 Approximate Applications

Approximate applications trade accuracy for performance, power, energy, or other benefits. Approaches include both static, compile-time tradeoff analysis [37, 39, 1, 8, 7] and dynamic, runtime support for tradeoff management [19, 20, 9, 17, 4, 2, 40, 35]. 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 runtime behavior to specific inputs. For example, Green maintains accuracy goals while minimizing energy [4], and Eon extends battery life in exchange for accuracy [40]. Both Green and Eon use heuristic techniques for managing the tradeoff space. PowerDial [17], uses control theoretic techniques to provide performance guarantees while maximizing accuracy. Each supports a single constraint. For example, Green uses heuristics to ensure accuracy bounds are not violated, but it does not guarantee energy consumption. PowerDial formally guarantees performance (so it can meet real-time or quality-of-service goals), but does not manage energy. Eon uses heuristics to prevent embedded devices from running out of energy, but does not support maximizing accuracy on energy budgets. Furthermore, these approaches are designed to work at the application-level only.

2.2 Energy-aware Systems

Many system-level approaches coordinate the use of multiple resources to provide performance guarantees with reduced power or energy consumption. For example, Li et al. manage memory and processor speed [26], Dubach et al. coordinate several microarchitectural features [10], and Maggio et al. coordinate core allocation and clock speed [29]. Meisner et al. propose coordinating CPU power states, memories, and disks to meet performance goals while minimizing power consumption [30]. Bitirgen et al. coordi-

nate clockspeed, cache, and memory bandwidth in a multicore [5]. The METE system controls cache, processor speed, and memory bandwidth to meet performance requirements [38]. Still other approaches manage arbitrary sets of system-level components [41, 44, 18, 34]; however, none of these approaches can coordinate system resource usage with application-level adaptation or ensure energy budgets are met.

2.3 Cross-layer Approaches

Cross-layer approaches work with both application and system. Static approaches coordinate by marking application regions as candidates for accuracy loss and then statically determining when the system can turn that loss into performance or energy savings. The Truffle architecture [11] supports applications which explicitly mark some computations and data as "approximate." Parrot replaces approximate regions of an application with a neural network implementation, which is then executed on a special hardware neural processing unit [12]. Flikker, allows applications to mark some data as "non-critical," and store it in a DRAM that trades accuracy for energy savings [27]. The co-design of application with *inexact* hardware has been proposed to reduce energy consumption dramatically in exchange for reduced application accuracy [32, 3, 33].

JouleGuard is inspired by prior work that *dynamically* coordinates across application and system layers. Flinn and Satyanarayanan coordinate operating systems and applications to meet user defined energy goals [13]. This system trades application quality for reduced energy consumption. GRACE-2 employs hierarchy to provide predictable performance for multimedia applications, making system-level adaptations first and then application-level adaptations [43]. Like GRACE-2, Agilos uses hierarchy, combined with fuzzy control, to coordinate multimedia applications and systems to meet a performance goal [25]. Maggio et al. propose a game-theoretic approach for a decentralized coordination of application and system adaptation which provides real-time guarantees [28]. xTune uses static analysis to build a model of application and system interaction and then refines that model with dynamic feedback [22]. The CoAdapt system uses a control theoretic approach to meet a performance, power, or accuracy constraint [15].

Two key features distinguish JouleGuard from prior cross-layer approaches. First, prior work does not provide energy guarantees, most instead guarantees performance while minimizing energy consumption. Second, prior work splits the system and application into two linear problems, which is possible because of the focus on performance [43, 25, 15]. Providing energy guarantees, however, is a non-linear problem (see Sec. 3). JouleGuard also splits the problem into two subproblems, but acknowledges that these problems are not independent. A key contribution of JouleGuard is to formulate a solution that is provably robust despite the dependence between the subproblems. JouleGuard's approach requires novel solutions for both subproblems rather than repurposing existing work to target energy.

3. Optimization Model

This section formalizes what it means to maximize an approximate application's accuracy for an energy budget. We first formalize the problem as a mathematical optimization. We then reason about 1) application-only, 2) system-only, and 3) cross-layer solutions.

We assume the application must perform some set amount of work W. The work cannot change, but the application can do this work faster or slower by changing its accuracy. For example, a video encoder must encode an entire video, but can use different algorithms that change the encoding time and the noise in the output. We assume an energy budget E representing the energy a user is willing to spend doing work W. Therefore, we must ensure the work is completed within the energy budget and accuracy is maximized.

We assume an approximate application with a set of configurations C_A, where each configuration $k \in C_A$ represents a unique performance r_k and accuracy a_k. We assume that a_c is a relative metric rather than absolute – many standard metrics for representing how far approximate applications are from a nominal behavior apply [35, 33]. An energy-aware system has configurations C_S, where each configuration $k \in C_S$ has performance r_k and power consumption p_k.

We do *not* assume that the application and system are independent. Instead we assume that changing either application or system may have unmodeled affects on the other. For example, changing application accuracy may change system power consumption. Similarly, changing system performance may have an unknown affect on the performance of different application configurations. The goal is to create a solution that is robust despite these unmodeled dependences.

3.1 The Model

The following maximizes accuracy given an energy budget:

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_c t_{\langle app,sys\rangle} \cdot a_{\langle app,sys\rangle} && (1)\\ \text{subject to} \quad & \\ & \textstyle\sum_c t_{\langle app,sys\rangle} \cdot p(app,sys) \leq E && (2)\\ & \textstyle\sum_c t_{\langle app,sys\rangle} \cdot r(app,sys) = W && (3)\\ & t_{\langle app,sys\rangle} \geq 0 && (4) \end{aligned}$$

Here $a(app, sys)$, $p(app, sys)$, and $r(app, sys)$ are (possibly) non-linear functions representing the accuracy, power, and performance of the combination of application configuration $app \in C_A$ and $sys \in C_S$. Eqns. 1–4 schedule times $t_{\langle app,sys\rangle}$ to spend in the configuration $\langle app, sys\rangle$. Eqn. 1 maximizes accuracy, the remaining equations ensure that the energy budget is not exceeded (Eqn. 2), that the work is completed (Eqn. 3), and that the times are non-negative (Eqn. 4).

Eqns. 1–4 represent a non-linear optimization problem. This non-linearity distinguishes the energy guarantee problem from prior work that provides performance guarantees, formulated as a linear program [43, 17, 29]. While JouleGuard adopts a similar approach to prior work – namely, splitting the optimization problem into application and system components – it adopts novel solutions to these subproblems to provide energy guarantees.

This section, presents several possible solutions to Eqns. 1–4. The first considers only application-level configurations, the second considers only system-level configurations, and the third considers cross-layer approaches coordinating application and system.

3.2 Application-Only Optimization

Considering only application-level optimization, we have a single system configuration with a single power consumption p_s. Thus, it is trivial to solve Eqns. 1–4. Since power cannot change, energy will be reduced by completing the work in the shortest amount of time possible. Therefore, the optimal accuracy solution will be the one that lets $t_{\langle app,s\rangle} \cdot p_s = E$. Therefore, $r_{app} = p_s \cdot W/E$. So, the solution is to set the application in the highest accuracy configuration that achieves a computation rate of at least $p_s \cdot W/E$.

3.3 System-Only Optimization

In contrast, system-only solutions cannot alter application behavior. Instead, they work on performance and power tradeoffs. In this case, accuracy is not a variable, so every feasible solution to the constraints (Eqns. 2–4) is equivalent. Thus, a system-only approach can solve this problem as long as there is a system configuration s such that $r_s/p_s \geq W/E$. In other words, a system-only approach

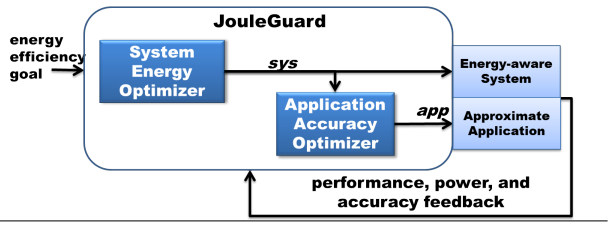

Figure 1. The JouleGuard runtime system.

works if there is a configuration whose energy efficiency is greater than or equal to W/E. Without such a configuration, then the problem has no feasible solution.

3.4 Coordinated Cross-layer Optimization

A coordinated, cross-layer approach selects both application and system configuration. The true optimal solution requires knowledge of all combinations of application and system performance, power, and accuracy as well as a solution to the non-linear optimization problem. In this section we imagine what a solution would look like if application and system were independent.

As shown above, the system does the most work for an energy budget when running in its single most energy efficient state:

$$sys = \underset{c}{\text{argmax}}\{r_c/p_c | c \in S\}. \tag{5}$$

By the theory of mathematical optimization, an optimal solution to Eqns. 1–4 lies on the polytope of feasible solutions [6]. For Eqns. 1–4, this property implies that the optimal solution occurs when there is strict equality in Eqn. 2; *i.e.,* when:

$$\sum_c t_c \cdot p_c = E \tag{6}$$

Combining Eqns. 5 and 6, $t_{sys} = 1.0$, $t_c = 0 \forall c \neq sys$ and

$$t_{sys} = E/p_{sys} \tag{7}$$

To satisfy Eqn. 3, the application and system must work at a combined performance of r. If we denote the approximate application's speedup as s_{app}, then we can write $r = s_{app} \cdot r_{sys}$; *i.e.,* the work rate is the product of the system's computation rate and the speedup provided by adapting the application. We now solve for an s_{app} that both ensures the work gets done and that accuracy is maximized. We find this by substituting r and t_{sys} into Eqn. 3:

$$\begin{aligned} t_{sys} \cdot r &= W \\ E/p_{sys} \cdot s_{app} \cdot r_{sys} &= W \\ s_{app} &= \frac{W \cdot p_{sys}}{E \cdot r_{sys}} \end{aligned} \tag{8}$$

Therefore, the application must be configured to respect Eqn. 8 and maximize accuracy:

$$app = \underset{c}{\text{argmax}}\{a_c | s_c \geq \frac{W \cdot p_{sys}}{E \cdot r_{sys}} \wedge c \in A\} \tag{9}$$

Thus, we solve the problem of coordinating application and system to maximize accuracy on an energy budget by putting the system in its most energy efficient configuration (determined by Eqn. 5) and then configuring the application to achieve the necessary additional speedup (as in Eqn. 9).

4. JouleGuard

This section presents the JouleGuard runtime system, illustrated in Fig. 1. Following the analysis of the previous section, we split the optimization problem into two components. The first, labeled *System Energy Optimizer* (SEO) in the figure, is responsible for putting the system into sys, the most energy efficient system configuration (found according to Eqn. 5). The expected speed and power of this configuration are passed to the *Application Accuracy Optimizer* (AAO), which determines the highest accuracy application configuration that will meet the energy budget (according to Eqn. 9).

If the performance, power, and accuracy of all combinations of application and system configuration are known ahead of time and do not change, then the application and system configuration need only be configured once. In general, however, we expect unpredictable dynamic fluctuations making it impossible to predict the highest energy efficiency system configuration ahead of time. Furthermore, this configuration may be both application and input dependent [31, 23]. Therefore, we solve the optimization at runtime using dynamic feedback. Both the SEO and AAO adapt to changes in the other, yet still converge to a reliable steady-state behavior. This section first describes SEO and AAO. It then formally analyzes JouleGuard's stability and robustness.

4.1 System Energy Optimization

JouleGuard uses reinforcement learning to identify the most energy efficient system configuration, employing a *bandit-based* approach [21]. We model system configurations as arms in a multi-armed bandit (essentially levers in different slot machines). The reward for pulling an arm is the energy efficiency of that configuration. Our goal is to quickly determine which arm (configuration) has the highest energy efficiency. Specifically, JouleGuard estimates configuration c's energy efficiency by estimating performance and power $\hat{r}_c$ and $\hat{p}_c$ using exponentially weighted moving averages:

$$\begin{aligned} \hat{p}_c &= (1-\alpha) \cdot \hat{p}_c + \alpha \cdot p_m \\ \hat{r}_c &= (1-\alpha) \cdot \hat{r}_c + \alpha \cdot r_m \end{aligned} \tag{10}$$

Where p_m and r_m represent the *measured* power consumption and performance respectively and α is a parameter between 0 and 1 that represents the sensitivity to noise. For our experiments, we use $\alpha = .85$, which provides the best outcomes on average across all applications and systems.

In a typical bandit problem, the initial estimates might be random values. This is not a good choice for estimating performance and power as we know a general trend: power and performance will increase with increasing resources. Therefore, JouleGuard initializes its performance and power estimates so that the performance and power increase linearly with increasing resources. This is an overestimate for all applications, but it is not a gross overestimate. Such an initialization performs exceedingly well in practice.

The final component of a bandit solution is balancing exploration (*i.e.,* trying different configurations) and exploitation (*i.e.,* making use of the best configuration found so far). In addition, JouleGuard must be reactive to changes caused by application-level adaptation. Therefore, JouleGuard explores the system configuration space using Value-Difference Based Exploration (VDBE) [42]. VDBE balances exploration and exploitation by dynamically computing a threshold, ϵ ($0 \leq \epsilon \leq 1$). When selecting a system configuration, JouleGuard generates a random number r ($0 \leq r < 1$). If $r < \epsilon$, JouleGuard selects a random system configuration. Otherwise, JouleGuard selects the most energy efficient configuration found so far. ϵ is initialized to 1 and updated every time the runtime is invoked. A large difference between the measured efficiency r_m/p_m and the estimate $\hat{r}/\hat{p}$ results in a large ϵ, while a small difference makes ϵ small. ϵ is initially 1. At each iteration of the runtime ϵ is updated as:

$$\begin{aligned} x &= e^{\frac{-|\alpha(r_m/p_m - \hat{r}/\hat{p}|}{5}} \\ f &= \frac{1-x}{1+x} \\ \epsilon(t) &= \frac{1}{|C_S|} \cdot f + (1 - \frac{1}{|C_S|}) \cdot \epsilon(t-1) \end{aligned} \tag{11}$$

If the random number is below $\epsilon(t)$, JouleGuard selects a random system configuration. Otherwise, JouleGuard uses Eqn. 5 but sub-

stituting the estimates:

$$sys = \underset{c}{\text{argmax}}\{\hat{r}_c/\hat{p}_c | c \in S\}. \quad (12)$$

JouleGuard then puts the system into this configuration and uses the expected performance and power consumption to perform application accuracy optimization.

This bandit-based approach has the nice property that when the system models are correct, JouleGuard does not change configurations. If the system is disturbed in anyway, or the application has an unexpected impact on system performance and power, JouleGuard will explore new states to find more efficient configurations.

4.2 Application Accuracy Optimization

Given the system configuration found above, JouleGuard determines the application configuration that satisfies Eqn. 9 while maximizing accuracy. Recall from Sec. 3.4, that this configuration must have a particular performance given by Eqn. 8. Substituting $\hat{r}_c$ and $\hat{p}_c$ into Eqn. 8, JouleGuard must find the application configuration that provides speedup:

$$s_{app} = \frac{W \cdot \hat{p}_{sys}}{E \cdot \hat{r}_{sys}} \quad (13)$$

The difficulty is that ensuring energy requires that JouleGuard maintains this performance despite unpredictable events (*e.g.,* application workload fluctuations), temporary disturbances (*e.g.,* page faults), or unmodeled dependences between application configuration and system power consumption. Therefore, JouleGuard models the problem of meeting speedup s_{app} as a control problem and minimizes the error $error(t)$ between $r = s_{app} \cdot r_{sys}$ and the measured performance $r_m(t)$ at time t; *i.e.,* $error(t) = r - r_m(t)$.

Maintaining performance despite dynamic environmental changes is a classical control problem and many cross-layer approaches incorporate control for this reason [14, 41, 44, 43, 25, 15]. JouleGuard builds on these examples by formulating a proportional integral (PI) controller for eliminating $error(t)$:

$$s(t) = s(t-1) + \frac{(1-p) \cdot error(t)}{\hat{r}_{sys}} \quad (14)$$

Where $s(t)$ is the speedup required beyond $\hat{r}_{sys}$ (*i.e.,* $r = s(t) \cdot \hat{r}_{sys}$), and p is the *pole* of the control system, a constant that determines the largest possible error that JouleGuard can tolerate while maintaining stability and ensuring that the energy goal is met. While many systems use control, JouleGuard's approach is unique in that the controller constantly adjusts to the changing system configuration by setting its pole to maintain robust behavior (see Sec. 4.3).

JouleGuard determines the application configuration by 1) measuring the performance at time t, computing the error between the required performance and the measured performance, then computing a speedup $s(t)$. JouleGuard then searches application configurations on the Pareto-optimal frontier of performance and accuracy tradeoffs to select the highest accuracy configuration delivering that speedup:

$$app = \underset{c}{\text{argmax}}\, a_c | s_c > s(t) \wedge c \in A \quad (15)$$

4.3 Control Theoretic Formal Guarantees

JouleGuard's control system provides formal guarantees of energy consumption. We first show that the the control system converges to the desired speedup. This can be done through standard analysis in the Z-domain [24].

4.3.1 Stable and Convergent

We want to analyze the behavior of the closed loop system that maps the performance goal r into measured performance $r_m(t)$. The Z-transform of the application is simply $A(z) = \frac{\hat{r}_{sys}}{z}$. While the transfer function of the control system is $C(z) = \frac{(1-p)z}{(z-1)}$. Therefore, the transfer function of the closed loop system is:

$$F(z) = \frac{C(z) \cdot A(z)}{1 + C(z) \cdot A(z)} = \frac{\frac{(1-p)z}{(z-1)} \cdot \frac{\hat{r}_{sys}}{z}}{1 + \frac{(1-p)z}{(z-1)} \cdot \frac{\hat{r}_{sys}}{z}} = \frac{1-p}{z-p} \quad (16)$$

Thus, JouleGuard is stable and converges to the desired performance if $0 \leq p < 1$.

4.3.2 Robust to Estimation Error

Recall that $\hat{r}_{sys}$ used in the controller is an estimate of the application's true performance in that system configuration. We can determine the controller's robustness in the face of errors in this estimate by analyzing Eqn. 16. Suppose the estimate is incorrect and the true value is $\bar{r}_{sys} = \delta \hat{r}_{sys}$ where δ is a multiplicative error in the estimation. For example, $\delta = 5$ indicates that model is off by a factor of 5. We determine JouleGuard's robustness to errors by substituting $\delta \hat{r}_{sys}$ into $F(z)$:

$$F(z) = \frac{C(z) \cdot A(z)}{1 + C(z) \cdot A(z)} = \frac{\frac{(1-p)z}{(z-1)} \cdot \frac{\delta \hat{r}_{sys}}{z}}{1 + \frac{(1-p)z}{(z-1)} \cdot \frac{\delta \hat{r}_{sys}}{z}} = \frac{(1-p)\delta}{z + (1-p)\delta - 1} \quad (17)$$

The controller represented by Eqn. 17 is stable if and only if the poles are between 0 and 1. Thus, for a stable system

$$0 < \delta < \frac{2}{1-p}. \quad (18)$$

So, the value of p determines how robust JouleGuard is to errors in performance estimates. For example, $p = 0.1$ implies that $\hat{r}_{sys}$ can be off by a factor of 2.2 and JouleGuard will still converge.

To provide a guarantee of convergence, JouleGuard must set the pole to provide stability. JouleGuard has a bound on error as it is constantly updating its estimates of system performance using Eqn. 10. Thus, the runtime computes the error bound as:

$$\delta(t) = r_m(t)/\hat{r}_{sys}(t-1) - 1 \quad (19)$$

and computes the pole as:

$$p(t) = \begin{cases} \delta(t) > 2: & 1 - 2/\delta(t) \\ \delta(t) \leq 2: & 0 \end{cases} \quad (20)$$

JouleGuard automatically adapts the pole so that controller is robust to error in the system performance estimates. In practice, the pole is large when the learner is unsure and likely to randomly explore the space. This means that the controller will be slow to change configurations when the learner is aggressive. In contrast, when the learner converges the error is low (by definition) and the controller can be more aggressive. *This adaptive pole combined with machine learning are unique features of JouleGuard which distinguish it from prior approaches and allow JouleGuard to split the energy guarantee problem into two subproblems yet still provide robust energy guarantees.*

4.3.3 Impossible Goals

A user may request an energy goal that is impossible to meet given the application and the system. In this case, JouleGuard reports that the goal is infeasible and then configures application and system to provide the smallest possible energy consumption.

4.4 Implementation

The JouleGuard runtime is summarized in Algorithm 1. We implement this algorithm through a straightforward coding of the math described above and summarized in the algorithm listing. The two key challenges are measuring feedback and configuring the application and system. These are really interface issues rather than technical issues. JouleGuard needs to be supplied a function that reads

Algorithm 1 The JouleGuard Runtime.

Require: W ▷ Workload provided by user
Require: E ▷ Energy budget provided by user
loop
  Measure work done $W(t)$ and energy consumed $E(t)$.
  Measure performance $r_m(t)$ and power $p_m(t)$ in configuration c.
  Update performance and power estimates $\hat{r}_c$ and $\hat{p}_c$ (Eqn. 10).
  Update $\epsilon(t)$ (Eqn. 11).
  Select energy optimal system configuration sys (Eqn. 12).
  Put the system in configuration sys.
  Compute the controller's pole (Eqns. 19 and 20)
  Compute remaining energy and work.
  Use those values to compute speedup target (Eqn. 13).
  Compute speedup control signal (Eqn. 14).
  Select the application configuration to deliver speedup (Eqn. 15).
end loop

App	Configs	Speedup	Acc. Loss (%)	Acc. Metric
x264	560	4.26	6.2	encoding quality [17]
swaptions	100	100.35	1.5	swaption price [17]
bodytrack	200	7.38	14.4	track quality [17]
swish++	6	1.52	83.4	precision & recall [17]
jacobi	18	9.99	0.19	residual error [39]
canneal	3	1.93	7.1	wire length [39]
ferret	8	1.24	18.2	similarity [39]
streamcluster	7	5.52	0.55	cluster quality [39]

Table 1. Approximate Application configurations.

performance and power. Any performance metric can be used as long as it increases with increasing performance. Similarly, power can be read from an external power monitor or from modern hardware devices that support on-board power measurement. Prior work defined a general interface for specifying system-level configurations [18]. A straightforward extension of this interface allows it to support application configuration changes as well. Given these interfaces, we implement JouleGuard as a C runtime that can be compiled directly into an application. It can replace existing runtime systems for approximate applications, or it can convert a statically configured approximate application into one dynamically configurable to meet energy goals.

JouleGuard does not require quantification of application accuracy. Many frameworks provide this (*e.g.,* [4, 17, 16, 35]), but others do not (*e.g.,* [40, 9]). Approaches that do not quantify accuracy still order configurations, but the order represents preferences rather than numerical differences. JouleGuard never needs a numerical value for accuracy. The only place it reasons about accuracy at all is in Eqn. 15 when selecting an application configuration. This equation does not actually require a numerical comparison, it simply requires a total order on available configurations. Thus, JouleGuard is compatible with a wide range of approximate approaches including those that do not specifically quantify accuracy.

5. Experimental Setup

To demonstrate generality, we test JouleGuard with eight different applications on three different hardware platforms.

We draw on existing approximate applications from two sources. The first is PowerDial, which automatically turns static command line parameters into a data structure which alters runtime performance and accuracy tradeoffs [17]. The second is Loop Perforation, which eliminates some loop iterations to trade accuracy for performance [39]. We build x264, swaptions, bodytrack, and swish++ with PowerDial. We build jacobi, canneal, ferret, and streamcluster with Loop Perforation. Table 1 summarizes the available application configurations, showing the total available configurations, the maximum speedup, maximum accuracy loss (as a percentage of the default value), and the accuracy metric.

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
idle	n/a	1.00	1.00
mem controllers	2	1.84	1.11

Table 2. System configurations.

These applications represent several different workloads. x264 is a video encoder, which can trade increased noise in the output video for increased frame rate. swaptions is a financial analysis benchmark that trades accuracy of price for speed of pricing. bodytrack does image analysis to follow a person moving through a scene. It trades the precision of the track for increased throughput. The swish++ search engine is a webserver which supports document search and can trade the precision and recall of the search results for decreased search time. jacobi is an iterative application that solves partial differential equations and can trade the floating point precision of its computation for decreased runtime. canneal is an engineering application that performs place-and-route on a netlist; it can trade increased wire length for decreased routing time. ferret is a image similarity search that can decrease the similarity of the results it returns in exchange for decreased search time. Finally, streamcluster is a clustering algorithm that can decrease the quality of its clustering for increased performance. Each application supports a different accuracy metric. To standardize the presentation, we report accuracy as a proportion of the accuracy achieved when running in the application's default configuration; *i.e.,* without PowerDial or Loop Perforation.

We evaluate JouleGuard on a dual socket system with two eight-core Intel Xeon E5-2690 processors. It supports hyperthreading and TurboBoost. It allows power and energy consumption to be read directly from registers at runtime in millisecond intervals [36]. The processor supports 16 different DVFS settings and a low power idle state where it consumes approximately 12 Watts. The maximum power consumption is 270 Watts. We run Linux 3.2.0 with the `cpufrequtils` package to change clock speeds. We use process affinity masks to assign an application to cores and hyperthreads. We use the `numalib` package to assign memory controllers to an application on the server. The available system configurations are summarized in Table 2, which shows the total number of available configurations and the maximum increase in speed and power measured. There are effectively an unlimited number of idle settings, as any application could be stalled arbitrarily.

6. Evaluation

For each application and platform, we first measure accuracy and energy consumption in the default configuration; *i.e.,* we run out-of-the-box on our platforms with no changes. Having established a baseline energy consumption, we then deploy each application with JouleGuard for a several energy goals which decrease energy by some factor f where $f \in \{1.1, 1.2, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0\}$. For example, $f = 2.0$ reduces energy consumption by $2\times$ compared to the default. For each goal we measure both the achieved energy and accuracy.

To quantify JouleGuard's ability to meet energy goals, we compute *relative error*:

$$Relative\ Error = \frac{e_m(t) - e_{goal}}{e_goal(t)} \cdot 100\% \qquad (21)$$

between the measured energy and the goal. Where n is the number of measured samples, $e_m(t)$ is the measured energy at time t and e_{goal} is the target. Low relative error indicates the energy consumption is close to the target, while high error represents a lack of convergence. Relative error is a percentage, allowing comparisons across different targets.

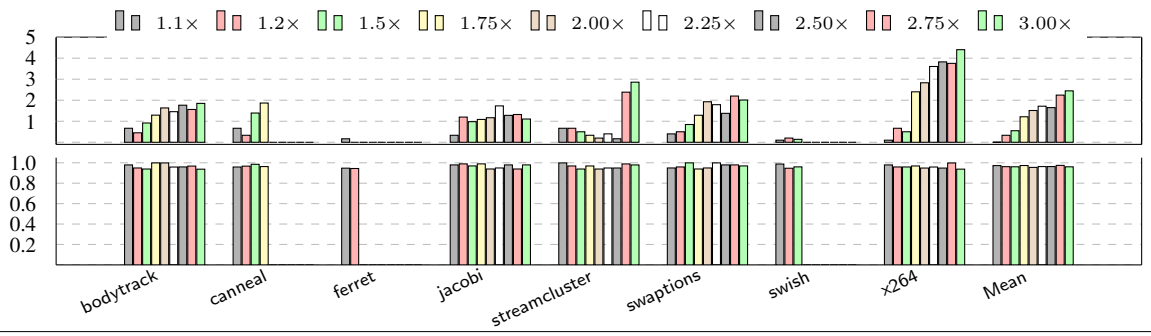

Figure 2. JouleGuard's low relative error (top) shows it is within a few % of the desired energy, with near near optimal accuracy (bottom).

To quantify optimality, we construct an *oracle* representing the best possible accuracy for an application, system, and energy target. We exhaustively profile the application and system in every possible configuration to determine the optimal accuracy for different energy targets. We then calculate the best system and application state for every application iteration. The oracle thus represents the best accuracy that could be accomplished by dynamically managing application and system with perfect knowledge of the future and no overhead. We quantify optimality as *effective accuracy:*

$$effective\ acc = acc_m / acc_{oracle}(goal) \tag{22}$$

Where acc_m is the measured accuracy for the application running with JouleGuard and $acc_{oracle}(goal)$ represents the accuracy our oracle returns for the given energy goal.

Stability: One of JouleGuard's essential properties is convergence to desired energy targets. To demonstrate the stability and convergence in general we compute relative error for all applications and energy targets. These results are shown in the top of Fig. 2 with benchmark name on the x-axis and relative error (as a percentage) on the y-axis. There is a bar for each energy target if the target is feasible (*e.g.,* given the available application configurations ferret can only achieve reductions up to $1.2\times$). In general, relative error is low – under 3% – demonstrating that JouleGuard meets energy guarantees on a number of platforms for a number of applications.

Optimality: We now quantify accuracy for all benchmarks and energy targets by presenting the effective accuracy calculated according to Eqn. 22. The bottom chart of Fig. 2 demonstrates JouleGuard's optimality. This chart shows the benchmark name on the x-axis and the effective accuracy on the y-axis. An effective accuracy of 1 represents the optimal achievable accuracy for that energy target as determined by our oracle. The effective accuracy for these benchmarks is close to unity, representing accuracy very close to the oracle. This accuracy is achieved despite JouleGuard's overhead and the inherent noise in the benchmarks.

7. Conclusion

This paper has proposed JouleGuard, an optimizing runtime system that coordinates approximate applications and energy-aware systems to meet energy goals while maximizing accuracy. JouleGuard is based on the key insight that we can solve this particular optimization by dividing the problem into two sub-problems, each of which can be solved efficiently. The first sub-problem moves the system to the most energy efficient configuration, while the second dynamically manages performance. JouleGuard proposes a machine learning approach to the first problem and a control theoretic solution to the second. We have implemented JouleGuard and tested it with a number of applications and systems. We find that – both empirically and analytically – JouleGuard meets energy budgets with near optimal accuracy, while adapting to application phases and consistently outperforming approaches that consider only application or system configurations.

References

[1] J. Ansel et al. "Language and compiler support for auto-tuning variable-accuracy algorithms". In: *CGO*. 2011.
[2] J. Ansel et al. "Siblingrivalry: online autotuning through local competitions". In: *CASES*. 2012.
[3] L. Avinash et al. "Designing Energy-Efficient Arithmetic Operators Using Inexact Computing". In: *J. Low Power Electronics* 9.1 (2013).
[4] W. Baek and T. Chilimbi. "Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation". In: *PLDI*. June 2010.
[5] R. Bitirgen et al. "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach". In: *MICRO*. 2008.
[6] S. Bradley et al. *Applied mathematical programming*. Addison-Wesley Pub. Co., 1977.
[7] M. Carbin et al. "Proving acceptability properties of relaxed nondeterministic approximate programs". In: *PLDI*. 2012.
[8] M. Carbin et al. "Verifying quantitative reliability for programs that execute on unreliable hardware". In: *OOPSLA*. 2013.
[9] F. Chang and V. Karamcheti. "Automatic Configuration and Run-time Adaptation of Distributed Applications". In: *HPDC*. 2000.
[10] C. Dubach et al. "A Predictive Model for Dynamic Microarchitectural Adaptivity Control". In: *MICRO*. 2010.
[11] H. Esmaeilzadeh et al. "Architecture support for disciplined approximate programming". In: *ASPLOS*. 2012.
[12] H. Esmaeilzadeh et al. "Neural Acceleration for General-Purpose Approximate Programs". In: *MICRO*. 2012.
[13] J. Flinn and M. Satyanarayanan. "Managing battery lifetime with energy-aware adaptation". In: *ACM Trans. Comp. Syst.* 22.2 (May 2004).
[14] A. Goel et al. "SWiFT: A Feedback Control and Dynamic Reconfiguration Toolkit". In: *2nd USENIX Windows NT Symposium*. 1998.
[15] H. Hoffmann. *CoAdapt: Predictable Behavior for Accuracy-Aware Applications Running on Power-Aware Systems*. Tech. rep. 2014.
[16] H. Hoffmann et al. *Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures*. Tech. rep. MIT-CSAIL-TR-2009-042. MIT, 2009.
[17] H. Hoffmann et al. "Dynamic Knobs for Responsive Power-Aware Computing". In: *ASPLOS*. 2011.
[18] H. Hoffmann et al. "A Generalized Software Framework for Accurate and Efficient Managment of Performance Goals". In: *EMSOFT*. 2013.
[19] M. Kambadur and M. Kim. *Energy Exchanges: Internal Power Oversight for Applications*. Tech. rep. CUCS-009-14. Columbia, 2014.
[20] M. Kambadur and M. Kim. "Trading Functionality for Power within Applications". In: *APPROX*. 2014.
[21] M. N. Katehakis and A. F. Veinott. "The Multi-Armed Bandit Problem: Decomposition and Computation". In: *Mathematics of Operations Research* 12.2 (1987), pp. 262–268. DOI: 10.1287/moor.12.2.262.
[22] M. Kim et al. "xTune: A Formal Methodology for Cross-layer Tuning of Mobile Embedded Systems". In: *ACM Trans. Embed. Comput. Syst.* 11.4 (Jan. 2013).
[23] E. Le Sueur and G. Heiser. "Slow Down or Sleep, That is the Question". In: *Proceedings of the 2011 USENIX Annual Technical Conference*. Portland, OR, USA, 2011.
[24] W. Levine. *The control handbook*. Ed. by W. Levine. CRC Press, 2005.
[25] B. Li and K. Nahrstedt. "A control-based middleware framework for quality-of-service adaptations". In: *IEEE Journal on Selected Areas in Communications* 17.9 (Sept. 1999).
[26] X. Li et al. "Cross-component energy management: Joint adaptation of processor and memory". In: *ACM Trans. Archit. Code Optim.* 4.3 (Sept. 2007).
[27] S. Liu et al. "Flikker: saving DRAM refresh-power through critical data partitioning". In: *ASPLOS*. 2011.
[28] M. Maggio et al. "A Game-Theoretic Resource Manager for RT Applications". In: *ECRTS*. 2013.
[29] M. Maggio et al. "Power Optimization in Embedded Systems via Feedback Control of Resource Allocation". In: *IEEE Trans. on Control Systems Technology* 21.1 (2013).
[30] D. Meisner et al. "Power management of online data-intensive services". In: *ISCA* (2011).
[31] A. Miyoshi et al. "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling". In: *ICS*. 2002.
[32] K. V. Palem. "Energy aware algorithm design via probabilistic computing: from algorithms and models to Moore's law and novel (semiconductor) devices". In: *CASES*. 2003.
[33] K. V. Palem and L. Avinash. "Ten Years of Building Broken Chips: The Physics and Engineering of Inexact Computing". In: *ACM Trans. Embedded Comput. Syst.* 12.2s (2013), p. 87.
[34] R. Rajkumar et al. "A resource allocation model for QoS management". In: *RTSS*. 1997.
[35] M. Rinard. "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks". In: *ICS*. 2006.
[36] E. Rotem et al. "Power management architecture of the 2nd generation Intel Core microarchitecture, formerly codenamed Sandy Bridge". In: *Hot Chips*. Aug. 2011.
[37] A. Sampson et al. "EnerJ: approximate data types for safe and general low-power computation". In: *PLDI*. 2011.
[38] A. Sharifi et al. "METE: meeting end-to-end QoS in multicores through system-wide resource management". In: *SIGMETRICS*. 2011.
[39] S. Sidiroglou-Douskos et al. "Managing performance vs. accuracy trade-offs with loop perforation". In: *ESEC/FSE*. 2011.
[40] J. Sorber et al. "Eon: a language and runtime system for perpetual systems". In: *SenSys*. 2007.
[41] D. C. Steere et al. "A Feedback-driven Proportion Allocator for Real-rate Scheduling". In: *OSDI*. 1999.
[42] M. Tokic. "Adaptive ϵ-Greedy Exploration in Reinforcement Learning Based on Value Differences". In: *KI*. 2010.
[43] V. Vardhan et al. "GRACE-2: integrating fine-grained application adaptation with global adaptation for saving energy". In: *IJES* 4.2 (2009).
[44] R. Zhang et al. "ControlWare: A middleware architecture for Feedback Control of Software Performance". In: *ICDCS*. 2002.