Racing and Pacing to Idle: Minimizing Energy Under Performance Constraints

David H. K. Kim Henry Hoffmann
Deptartment of Computer Science, University of Chicago
Email: {hongk,hankhoffmann}@cs.uchicago.edu

Abstract—**The problem of minimizing energy for a real-time performance constraint has been widely studied, both in theory and in practice. Theoretical models have indicated large potential energy savings, but practical concerns have made these savings hard to realize. Instead, practitioners often rely on the *race-to-idle* heuristic, which makes all resources available to a task and then idles the system until the next task is released. While this heuristic has proven effective, recent results indicate that more sophisticated resource allocation schemes may now provide greater energy savings. This paper investigates resource allocation heuristics for real-time constraints using both analytical and empirical techniques. We formalize the problem as a linear program and develop a geometric interpretation, allowing derivation of the optimality conditions for various heuristics. We then demonstrate that the *pace-to-idle* heuristic is often better and never worse than race-to-idle. We confirm these analytical results by implementing a resource allocator based on the studied heuristics and measuring energy consumption for eight different applications on four different systems. The results confirm that pace-to-idle produces better energy savings than race-to-idle, by up to 20% on the newest platform in our study.**

I. Introduction

Many computing applications have performance constraints defined by their interaction with the outside world. For these applications, the goal is to maintain predictable performance; i.e., respect real-time or quality-of-service (QoS) constraints. As energy concerns have grown increasingly important, an additional goal has arisen: meeting the performance constraints while minimizing energy consumption.

To support energy minimization, computing systems come with configurable components, or resources, which expose tradeoffs between delivered performance and power consumption. For example, almost all processors now support dynamic voltage and frequency scaling (DVFS), which permits software to tune the processor's clock speed [24]. Additionally, many processors have aggressive power gating, allowing unused resources (e.g., cores or cache banks) to be placed in an *idle state*, decreasing power consumption [26].

The presence of multiple configurable resources creates a need for *energy-aware* resource allocation schemes that schedule component usage to minimize energy for a given performance requirement. This problem has been studied from theoretical [1–3, 6, 10, 13, 15, 32] and empirical [7, 12, 16, 17, 19, 21–23] perspectives.

While theoretical models have long demonstrated the potential energy savings of careful resource orchestration, the assumptions required to realize these savings often could not be implemented in practice. For example, in many systems empirical studies have found that the most energy efficient resource allocation strategy is simply to *race-to-idle* [7, 16, 17, 23]. The beauty of this heuristic is its simplicity: it does not require any runtime optimization calculations. Instead, it simply makes the highest-performance configuration of resources available when a task enters the system, and then idles the system when the task completes.

Recent technology changes, however, have altered the conditions that make race-to-idle efficient. Several studies have identified real platforms (including both embedded and server systems) where racing-to-idle produces suboptimal results [12, 17, 21, 22]. These studies suggest *there is a need to revisit earlier results and better understand when the race-to-idle strategy is appropriate in practice and how much energy can be saved using more sophisticated resource allocation schemes.*

We address this need by developing an analytical framework for studying real-time resource allocation strategies. We then evaluate that framework on real systems using eight applications and four different hardware platforms. Given an application's performance constraint and a configurable computing system, the problem of minimizing energy is cast as a linear optimization problem. In this formulation, the goal is to schedule time in different configurations (representing combinations of allocated resources) such that a required workload is completed by a deadline and total energy consumption is minimized. In this paper, we observe that this particular problem's structure has several interesting geometrical properties that make it easy to reason about different resource allocation strategies and heuristics. Specifically, it is possible to determine when race-to-idle is optimal, when other strategies should be employed, and the cost of using suboptimal strategies. After developing this framework, we demonstrate its use minimizing full-system energy while scheduling resource usage for real applications on real machines.

The combined analytical and empirical study presented in this paper makes the following contributions:

1) It proposes a geometric interpretation for allocating resources to meet a performance constraint while minimizing energy consumption. (See Section III-C.)
2) It derives conditions under which processor idling will result in optimal energy consumption. Importantly, this analysis also indicates when systems should not be idled. (See Section V, Observation 5.1.)
3) It proves that *pace-to-idle*, another heuristic strategy, is often better (and never worse) than race-to-idle. (See Section V, Corollary 5.6.)
4) It shows that the conditions where race-to-idle is opti-

mal are degenerate, and thus, may not persist in future machines. (See Section V, Observation 5.5.)

5) In a study of four different hardware platforms, it shows that race-to-idle is close to optimal on older machines, but newer machines can achieve significant energy savings by adopting better resource allocation strategies. (See Section VI-B).
6) On the newest platform in the study, it shows pace-to-idle achieves a 20% energy savings compared to race-to-idle, while the true optimal resource allocation reduces energy by almost 30% compared to race-to-idle. (See Section VI-C).

The rest of this paper is organized as follows. Section II discusses related work (both theoretical and empirical) in energy-aware resource allocation under performance constraints. Section III formalizes resource allocation as a linear program and presents a geometric interpretation of this problem. Section IV describes how this geometric interpretation can be used to construct a convex function describing available performance and power tradeoffs. Section V observes several practical implications of the proposed problem formulation. Section VI presents our empirical evaluation. Section VII concludes the paper.

II. Related Work

This section discusses related work minimizing energy for a performance constraint. We begin by reviewing theoretical and algorithmic contributions. We then discuss some practical concerns and empirical studies.

A. Scheduling Algorithms and Theoretical Results

There has been considerable research concerning algorithms that optimize energy consumption under performance constraints for variable-speed processors. Such processors are equipped with a mechanism called *dynamic voltage and frequency scaling* (DVFS), which refers to their ability to dynamically set the speed/frequency depending on the current workload [20, 24]. A processor may execute tasks in higher performance states for higher energy consumption or trade performance for energy savings[1]. Some systems are further equipped with a *sleep state*, in which a system can be placed in a low-power state at a constant cost of waking up later [20].

In this framework, theoretical models formulate the optimization problem as a deadline-based job scheduling problem incorporating DVFS [1, 3, 4, 6, 10, 15, 20, 25, 32]. A single variable-speed processor is given a set of jobs, each specified by its release time, deadline, and workload. In the offline setting, all jobs are known in advance, while in the online setting, a job is known to the processor only when it arrives at its release time. In both cases, a job may be executed at any point during the time interval defined by its release time and deadline and requires the given amount of workload to be processed for completion. At any point in time, the processor must choose both the job to execute and the processing speed. The goal is to create a feasible schedule such that all jobs are processed while minimizing total energy consumption.

While various online/offline algorithms have been designed for different versions of the problem, their efficacy is largely determined by the *power-performance tradeoff space*, or the *power function* of a given system implementation. Simply put, it is the space of all feasible speed/power consumption states that the computing device may utilize. On modern multicore systems with power-gating (e.g., Intel's SandyBridge and later [26]), these computation states include not only processor speed but also the active cores. This state space is usually modeled by a convex power function, $P(\cdot)$, where a computing device processing at a work rate of s has power consumption of $P(s)$. For example, the *cube-root* rule for CMOS devices state that the power of a device running at the speed s is proportional to s^3 [1]. Many algorithms naturally generalize this function to $P(s) = s^\alpha$ where $\alpha \geq 1$ is a constant, or to even general convex functions [1, 2, 6, 13].

The cube-root rule is based on the fact that power P in a CMOS combinational logic circuit is proportional to the square of voltage times the frequency; i.e., $P \propto v^2 f$ [24]. Reducing the frequency, or speed, allows voltage to be reduced as well. In contrast, increasing speed requires increasing voltage to avoid timing errors.

The YDS algorithm addresses offline scheduling with a single-core, variable-speed processor without a sleep state and power function $P(s) = s^\alpha$. This algorithm computes a feasible schedule minimizing total energy consumption [32]. These authors also propose two algorithms known as *Average Rate* and *Optimal Available* for the online problems, and their competitiveness has been analyzed [5]. Some models assume discrete speed settings [10]. There are models which assume a bounded speed processor. However, in this case feasibility is not guaranteed even in the offline setting, and the objective is to design algorithms achieving good *throughput*, or the total workload of jobs processed by their deadline, while minimizing energy consumption. The *Slow-D* algorithm is known to achieve optimal competitiveness in throughput for this setting [5].

When the system is equipped with a sleep state, even the offline problem with a processor of unbounded-speed becomes intractable, and algorithms have been developed achieving constant approximation factors for general power functions. Albers and Antoniadis also develop approximation factor lower bounds for a class of algorithms for general convex power functions [2].

B. Practical Concerns and Empirical Studies

However, such algorithms based on theoretically-founded models are often not applicable in practice. In fact, researchers have found that, counter-intuitively, on real machines energy can often be reduced by *racing-to-idle*; i.e., completing any set of jobs as fast as possible and then idling the system[7, 23]. Part of the reason is that CMOS voltage and frequency scaling for combinational logic (that gives $P \propto v^2 s$) does not apply to tradition SRAM circuits, as reducing the voltage in these circuits leads to unacceptable error rates [8]. As SRAM circuits are used to implement caches, this means significant portions of a processor do not obey the cube-root law. In addition, other essential system components (main memory, disks, network cards, and fans) consume significant energy making the energy savings due to DVFS in the processor small in comparison to total energy [7, 16] and accounting for these other components has a dramatic effect on real-time embedded systems [14, 31, 33]. Finally, most of the theoretical approaches assume that idle power is negligible, but in practice commercial systems have not brought idle power that low (see Section VI).

In addition, the aforementioned YDS algorithm and online algorithms make crucial assumptions, that the power function $P(s)$ is continuous and unbounded. While such models guarantee optimality or competitiveness theoretically, the simplifying assumptions disallow analysis of heuristics on real systems optimizing over a bounded, finite power-performance tradeoff space. Furthermore, even with assumptions of discreteness and boundedness, such algorithms have polynomial running times which may not be useful in practice.

Several empirical studies have investigated the practical constraints of minimizing energy for a performance bound. Empirical studies done in the early and middle part of the 2000's reinforced the notion that theoretical results are hard to realize in practice and race-to-idle is often near-optimal [7, 22, 23, 27]. More recent studies, however, suggest that this trend is starting to reverse and they call into question the efficacy of the race-to-idle heuristic [12, 17, 21]. These contradictory results suggest that it is time to re-evaluate resource allocation approaches and re-examine the potential energy savings of more sophisticated algorithms, while keeping in mind practical implications.

This paper focuses on the energy optimization of a variable-speed multicore system with a focus on idling heuristics. We give an in-depth analysis of the comparison of idling heuristics and the optimal solution based on our geometric interpretation of the problem. We then verify the practical usefulness of this interpretation by implementing these heuristics on real machines and allocating resources to real applications and measuring their effect on full system energy consumption.

III. The Optimization Problem

This section formalizes the constrained optimization problem of completing a task by a deadline while minimizing energy consumption. We consider a single task where the workload is W, and the deadline t. This formulation is broadly applicable to many such tasks on systems with differing configurations. We first formulate the problem as a linear program (LP). We then discuss heuristic solutions to the LP. We next formulate the dual LP, allowing a geometric interpretation of the problem, allowing us to construct the optimal solution simply.

A. LP Formulation

Assume the task starts at time 0 and must complete W units of work by time t. The system supports C configurations in $c \in \{0, \ldots, C-1\}$, each of which has a computation rate s_c and a power consumption of p_c. These configurations represent settings for all configurable components in the system (e.g., DRAM speed, processor speed, and number of active cores). The system has a unique *idle* configuration $c = 0$ with $s_0 = 0$ and $p_0 = p_{idle}$, where p_{idle} is a system dependent (and application independent) value representing the system's power consumption when not executing a computation. We assume configuration $C-1$ represents making all resources available to a task. The goal is to assign a time $0 \le t_c \le t$ to each machine configuration. Note that configuration c will contribute $t_c \cdot s_c$ work at a cost of $p_c \cdot t_c$ energy. This formulation is similar to others that appear in the literature, indicating this is a broadly applicable problem [3, 18, 30]. Thus, the problem of minimizing energy while completing the work can be expressed as:

$$minimize \sum_c t_c \cdot p_c \tag{1}$$

$$subject\ to$$

$$\sum_c t_c \cdot s_c = W \tag{2}$$

$$\sum_c t_c = t \tag{3}$$

$$0 \le t_c \le t,\ for\ c = 0, \ldots, C-1 \tag{4}$$

Equations 1–4 assign values for all t_c to minimize total energy consumption (Equation 1) subject to the constraints that the total work accomplished is equal to the required work W (Equation 2) and the deadline is met (Equation 3). The final constraint (Equation 4) ensures that the time spent in any configuration is non-negative.

B. Heuristic Solutions to the Linear Program

We describe some well-known heuristics commonly used in practice [17]. These solutions all meet the performance constraint (making them appropriate for real-time), but may not deliver minimal energy.

1) **naive race:** provide all resources (i.e., use $C-1$) until the task completes and idle until the next task is ready. This solution is easy to implement in practice since it does not require any analysis of the characteristics of different configurations.
2) **race:** run in the fastest configuration, $race$, until the task is complete then idle. The fastest configuration is the one that produces the highest computation rate, s_c; i.e. $race = \arg\max_c s_c$.
3) **pace:** run in the most energy efficient configuration, $pace$, until the task is complete and then idle. The energy efficiency of a configuration c is $e_c = s_c/p_c$, thus $pace = \arg\max_c\{s_c/p_c\}$. In the case that the most energy efficient configuration's work rate is too small, set $pace = \arg\max_c\{s_c/p_c : s_c \ge W/t\}$. Applying this heuristic requires finding the pace configuration.
4) **no-idle:** Never idle the system. Instead, alternate between the lowest power state that is just faster than necessary, hi, and the most efficient state that is slower than necessary, lo. Here, $hi = \arg\min_c\{p_c : s_c \ge W/t\}$ and $lo = \arg\max_c\{s_c/p_c : s_c \le W/t\}$, where t =total time given by the deadline.

C. Dual LP

By standard techniques in LP optimizations, we form the dual of the linear program. Simply put, we assign a variable for each constraint in (2) and (3), and a constraint for each variable in the original LP and form a new minimization problem which upper bounds the objective of the primal LP. By the duality theorems of LP, any feasible solution to the dual linear program (minimization) is an upper-bound on the primal optimal (maximization). Furthermore, the optimal value

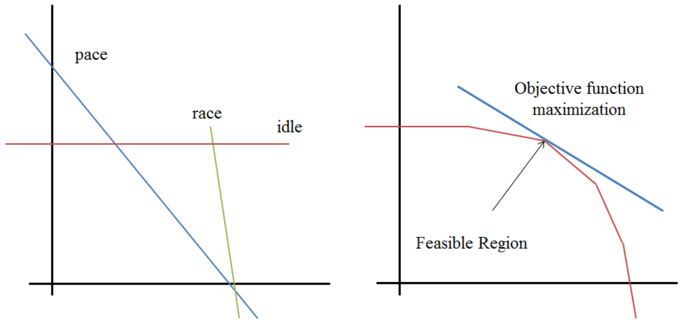

Fig. 1. (1) race, pace, and idle configurations in the dual space and (2) optimization in the convex hull formed by all configurations

for the dual program is exactly the optimal value of the primal, given that the primal has an optimal.

Since our primal LP had 2 constraints and C variables forming a convex polytope in a C-dimensional space, the dual program gives us C constraints in a 2-dimensional space, where the value of the optimal of the dual is exactly the value of the primal optimal. The significance of the dual program is in its low dimension - the geometric abstraction is meant only to find and describe the optimal solution and characterize it - which is a lot easier to see in 2D.

The dual LP of the primal (given in Equations 1–4) is:

$$maximize\ W \cdot x + t \cdot y \tag{5}$$

$$subject\ to$$

$$s_c \cdot x + y \leq p_c,\ \forall c = 0, \ldots, C-1 \tag{6}$$

$$x, y\ unconstrained \tag{7}$$

In the above LP, x is the new variable assigned for constraint (2) in the primal, and y is the new variable for constraint (3). There are C constraints in (6), each corresponding to a variable c in the primal LP. (5) is the new objective function formed with x and y.

Note that each inequality of (6) defines an under a line in a 2D plane. The line corresponding to a configuration c has slope $-s_c$, y-intercept p_c and x-intercept $p_c/s_c = 1/e_c$.

The intersection of the areas below the C lines in (6) define a convex hull, consisting of the feasible solutions. Note that a given constraint may be discarded, in the sense that some other constraint is strictly stronger, defining an area entirely contained in the area of the given constraint. Note that as shown in Figure 1, the race, pace, and idle configurations may also form edges of the convex hull. In fact, it is easy to show that the constraints corresponding to the race (steepest line), pace (smallest x-intercept), and idle (horizontal line) are always edges in the convex hull and not discarded by any other tighter constraints.

This convex hull is precisely the search space for the dual LP, where we find the point (x, y) which maximizes the objective function, $W \cdot x + t \cdot y$. Given the convex hull formed by the C constraints, finding the optimal point is simple. Take the line $-W/t \cdot x$ passing through the origin, and increase its y-intercept as much as possible, such that the line contains a feasible point, consisting of a single vertex of the convex hull (or an edge, in which case we can choose one vertex).

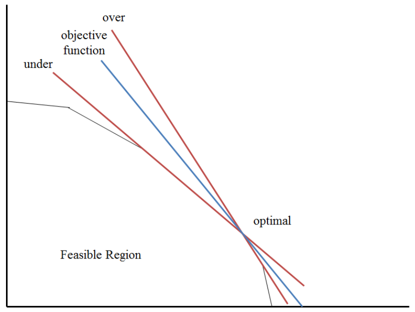

Fig. 2. Maximizing the objective function in the dual space

D. Properties of the Optimal

The objective function will eventually meet a single vertex or an edge of the convex hull when its y-intercept is maximized. In the case of an edge, all points on it are optimal solutions. In any case, there is a vertex which achieves the optimal value, which is defined by two constraints (edges) that meet to form the vertex.

Let (x^*, y^*) be the optimal vertex. By complementary slackness conditions of LP theory, if $t^* = (t_1^*, \ldots, t_{C-1}^*)$ is an optimal solution for the primal, the binding constraints (lines forming the vertex) where the objective function line meets the convex hull correspond to the non-negative variables in the primal, and the non-binding constraints to variables that are zero [11].

This basically says that a primal optimal solution will only have $t_c^* > 0$ for a corresponding binding constraint in the dual LP. Since there are only two binding constraints for a vertex at the optimal which we found, only two configurations will be used by the primal LP to obtain an optimal solution. In fact, we can exactly characterize the two configurations as the two corresponding edges on the convex hull which have the closest slopes to $-W/t$ of the objective function from above and below, as in Figure 2.

If $\mathcal{C}'$ is the set of configurations that comprise the convex hull in the dual space, then we have

Optimal Configurations:

$$over = \arg\min_{c \in \mathcal{C}'} \{s_c : s_c \geq W/t\} \tag{8}$$

$$under = \arg\max_{c \in \mathcal{C}'} \{s_c : s_c \leq W/t\} \tag{9}$$

This tells us that the two configurations to use will be the closest two which have work rates close to the average workload given by the task.

Optimal Algorithm:

The optimal algorithm computes the set of configurations forming the convex hull, then finds the two configurations whose slopes are just larger and smaller than that of the objective function. Once the two configurations have been found, it allocates time to the two configurations to get the

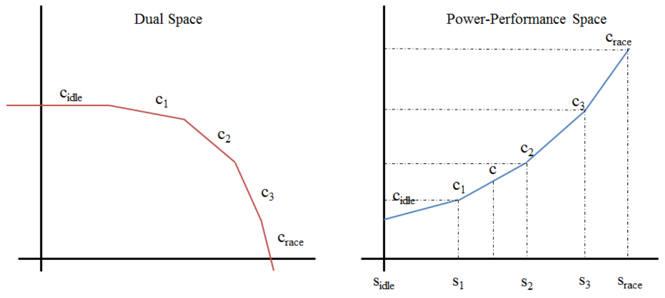

Fig. 3. Equivalent representations. Configuration c can be obtained by averaging c_1 and c_2.

work done exactly at the deadline. The times allocated to each configuration are:

$$t_{over} \cdot s_{over} + t_{under} \cdot s_{under} = W \tag{10}$$
$$t_{over} + t_{under} = t \tag{11}$$

which gives us

$$t_{over} = \frac{W - s_{under} \cdot t}{s_{over} - s_{under}} \tag{12}$$
$$t_{under} = \frac{s_{over} \cdot t - W}{s_{over} - s_{under}} \tag{13}$$

IV. Construction of a Power Function

In the optimization problem, the convex hull formed in the above section is entirely defined by the configuration space. Given any task to execute within a given deadline, we can always optimize for energy by considering its allotted time and finding the two configurations which achieve optimal energy consumption. Hence, every system with a finite set of configurations defining the power-performance tradeoff space has a fixed set of 'useful' configurations, and all other configurations will never be used if optimizing for energy. *The useful configurations are exactly the configurations defining the edges of the convex hull in the dual space.*

The two configurations from Equations 8 and 9 give equivalent average work rate and average power consumption to a single configuration with $s_{avg} = \frac{W}{t}$ and $p_{avg} = \frac{t_{over}}{t} \cdot p_{over} + \frac{t_{under}}{t} \cdot p_{under}$, assuming no transition costs between configurations. This means that the two configurations act as if the system were running at a single configuration on the edge of the convex hull of the two points, implying the following:

Observation 4.1: Assuming there are no switching costs for the processor to change configurations, we can assume that our power-performance space is described by bounded, finitely piecewise-linear power function.

This observation is illustrated in Figure 3. Given the constraints which form the convex hull in the dual space, we may consider the corresponding configurations in their original power-performance space. It can be easily shown that the points in the power-performance space also form a convex hull, and there are no other points below or on the function lines (see Appendix A for details). Intuitively, these points are the most energy efficient and 'useful' configurations.

We may assume a piecewise-linear function, because two adjacent configurations can be averaged to achieve the net effect of any power/performance of a point in the line segment connecting the two. As shown in Figure 3 the two closest configurations c_1 and c_2 can be combined to produce the same effect as c, even if the machine does not have c in its finite set of its configurations.

In the analysis of idling heuristics in the next section, we assume this system-dependent power function.

V. Practical Implications: Idling Heuristics

This section gives an in-depth analysis of idling heuristics for the single task optimization problem. By an *idling heuristic*, we mean a heuristic which always employs a c-to-idle mechanism for a fixed or well-defined configuration c; e.g., race-to-idle and pace-to-idle. We first investigate the conditions under which an idling heuristic is optimal. Then we give a simple condition to compare idling heuristics and show that pace-to-idle will consume less energy than race-to-idle.

A. Optimal vs an idling heuristic

The previous section shows that we may simply work with a convex, bounded piecewise-linear function in the power-performance space. Let $P(s)$ be this power function, and suppose we are given a total workload of W with time t as in the LP formulation. Let OPT denote both the optimal solution and the value of its total energy consumption.

Then by the convexity of $P(s)$, the optimal would consist of a single point on $P(s_{OPT})$, where $s_{OPT} = s_{avg} = W/t$, processing at s_{avg} for the entire time t. We know that even if we do not have this in our configuration space, we may get the same net effect using two configurations *over* and *under* as in the LP solution.

On the other hand, a c-to-idle heuristic would complete W in $t_c = (s_{OPT}/s_c) \cdot t$ and idle for the remaining time.

Figure 4 depicts the energy consumed by both OPT and c-to-idle over t. The horizontal axis represents time flow and the vertical axis represents the power consumption of a heuristic. The blue horizontal line represents the optimal (or combination of *over* and *under*) processing at s_{avg} for the entire time t with power consumption $P_{OPT} = P(s_{avg})$, given by the point in the piecewise-linear function $P(s_{avg})$. The idling heuristic, represented by the red line, works at a higher work rate and then transitions to the idle state.

The total energy consumption of each heuristic is the area under the corresponding lines representing the power consumption at each point in time. In Figure 4, the energy consumption of OPT is given by $OPT = Area2 + Area3 + Area4$, and the total energy consumption of c-to-idle is $ALG = Area1 + Area3 + Area4$. This means that $ALG - OPT = Area2 - Area1$.

So an idling heuristic is optimal iff $Area1 = Area2$, which gives us the following condition:

Observation 5.1 (Optimality of an idling heuristic): $OPT = ALG$ for a c-to-idle heuristic iff the following

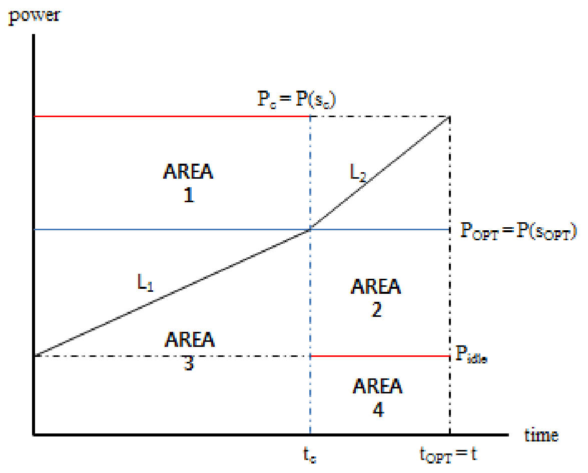

Fig. 4. Power consumption of OPT and c-to-idle over time.

condition holds:

$$\frac{s_c - s_{avg}}{P_c - P_{avg}} = \frac{s_{avg}}{P_{avg} - P_{idle}} \tag{14}$$

i.e., the slope of L_1 is equal to slope of L_2 in Figure 4.

This observation is based on the fact that OPT processes with two configurations that act as a single configuration on average on $P(s)$. Because the time that the c-to-idle heuristic performs work is exactly shortened by the ratio of s_c to s_{avg}, the above condition holds when we consider the slopes of the corresponding configurations in the power function, with performance as the x-axis.

Using the areas in Figure 4, we can calculate the optimality of an idling heuristic as follows:

$$\frac{ALG}{OPT} = \frac{\frac{p_c}{s_c}}{\frac{p_{avg}}{s_{avg}}} + \frac{p_{idle}}{p_{avg}}\left(1 - \frac{s_{avg}}{s_c}\right) \tag{15}$$

The left hand side of (Equation 15) can be interpreted as the penalty of ALG for its worse energy efficiency compared to OPT. The right hand side represents the penalty of using the idle state if $p_{idle} > 0$. It is easy to see that higher s_c simply results in a bigger ratio between ALG and OPT, due to convexity of $P(s)$.

The following immediately follows:

Corollary 5.2: Given any configuration space, the closer p_{idle} is to 0, the closer the energy consumption of any idling heuristic is to OPT.

It can also be seen that the ratio of the slopes between L_1 and L_2 in Figure 4 will decrease as p_{idle} goes down, making an idle heuristic better in terms of energy efficiency.

B. Race-to-Idle and Pace-to-idle

Recall the definition of pace-to-idle: it processes the workload at the most energy efficient configuration that completes the given workload in the deadline, then idles. Let $pace$ denote the unique, most energy efficient configuration at which the line from the origin meets the power function tangentially. If the given average workload $s_{avg} = W/t$ is less than s_{pace}, pace-to-idle will execute at the $pace$ configuration and then idle. Otherwise, by the convexity of $P(s)$, higher performance states have worse power efficiency, and therefore pace-to-idle will execute at the lowest work rate possible to complete the given task by the deadline.

Observation 5.3: If $p_{idle} = 0$ and the most energy efficient configuration, $pace$, has a high enough work rate to complete W within t, pace-to-idle is optimal.

Clearly, if $p_{idle} = 0$, then the pace configuration will form a linear line segment from the origin in $P(s)$, and the average work rate at which the optimal will work at will be on the same line. Note that it forms a linear line segment from the origin, because by definition no other point can be below the line segment formed by the origin and the pace configuration.

Observation 5.4: If the given workload and deadline requires a larger work rate than s_{pace}, pace-to-idle may not be optimal, but will be the lowest energy-consuming idling heuristic.

By the convexity of $P(s)$ and (Equation 15), an idling heuristic which processes the given task as close to the average work rate is more energy efficient and has the smallest slope of L_2 in Figure 4. The configuration c which achieves the smallest slope is the configuration with the minimum performance that can complete the workload in time, which is exactly the most energy efficient state in the power function that can complete the given workload in time. Since pace-to-idle is exactly this idling heuristic, the above observation holds.

Observation 5.5: race-to-idle is close to optimal only if the system achieves good energy efficiency in high work rates, i.e., the power function is close to linear.

The higher the convexity of the power function, the lower the energy efficiency of the race state. The difference between OPT and race-to-idle is maximized at the point in the power function which has the largest difference in slopes from the idle state and to the race state. If the power function is linear, there will be no difference between the two and race-to-idle will be optimal (see (Equation 15)). The following corollary is immediate from these observations, along with (Equation 15).

Corollary 5.6: pace-to-idle is always more energy efficient than race-to-idle. Their difference will be large if the power function has large convexity and the given workload and deadline has an average work rate on the power function at which the slope of the line from the $idle$ to $race$ has a large difference.

The two heuristics will perform similarly if the energy efficiency of $race$ is very good, (i.e., close to $pace$), or if the task requires very high performance. Note that this does not mean that pace-to-idle is the optimal among all heuristics. We present instances in which race-to-idle performs poorly in our empirical studies.

VI. Empirical Study

This section evaluates our framework on several different hardware platforms and eight different applications. We first discuss the specific machines and benchmarks used in this study. We then illustrate an example of the power functions (see Section IV) characteristic of these machines. Finally, we evaluate several heuristics and compare them to optimal by modifying a runtime control system to use these heuristics to maintain performance for each benchmark on each machine.

A. Machines and Applications

We use four hardware platforms. Their characteristics are summarized in Table I. $\mathsf{Machine}_1$ is a dual socket Intel Xeon X5460 with 8 cores, 1 memory controller, and 4 DVFS settings from 2.0 — 3.16 GHz. $\mathsf{Machine}_2$ is a dual socket Intel Xeon E5520 with 8 cores, 2 memory controllers, and 8 DVFS settings from 1.6 — 2.4 GHz. $\mathsf{Machine}_3$ is a single socket Intel E5-1650 with 6 cores, 1 memory controller and 12 DVFS settings from 1.2 — 3.2 GHz. $\mathsf{Machine}_4$ is a dual socket Intel Xeon E5-2690 with 16 cores, 2 memory controllers and 16 DVFS settings from 1.2 — 2.9 GHz. All machines support hyper-threading, $2-4$ support TurboBoost. All allow suspension of the current application to enter low-power idle states. On $\mathsf{Machine}_2$, idling consumes 69% of the lowest measured active power (90 W idle, 131 W lowest active power). On $\mathsf{Machine}_1$, idling consumes 94% of the lowest measured active power (200 W to 213 W). On $\mathsf{Machine}_3$, idling consumes 85% of the lowest measured active power (85W to 100W). On $\mathsf{Machine}_4$, idling consumes 75% of the lowest measured power (75W to 100W). The highest measured power consumption for these machines are 320W, 225W, 235W, and 430W respectively. To measure power consumption, all machines are connected to power meters which report total system power consumption. All results in this section concern total system power and total system energy.

We evaluate eight applications: blacksholes, bodytrack, fluidanimate, freqmine, swaptions, vips, and x264 from PARSEC [9], plus the swish++ search engine [29]. These benchmarks are broadly representative of applications with performance constraints. blackscholes and swaptions price financial instruments and represent computations with latency constraints. bodytrack and x264 process video data and must meet throughput and latency constraints to keep pace with a camera. fluidanimate must render graphics at a predictable frame rate. vips is an image processing application which has a latency requirement. freqmine is a data-mining application which may have a latency requirement for online data analysis. Finally, swish++'s search requests must be processed within a latency limit.

B. Power Functions

The geometric representation of Figure 3 relates to empirical measurements taken on these platforms. This section uses the x264 application as an example, but results follow similar trends for other applications (they are omitted to save space).

For x264, we measure the performance and power consumption in all possible configurations on each machine. These results are illustrated in Figure 5. The figure consists of a plot for each machine showing performance (measured in frames/s) on the x-axis and power consumption (in Watts) on the y-axis. The points represent each possible configuration. The solid lines represent the convex hull of optimal solutions to resource allocation problems. The structure of these plots (taken from measured data) mirrors that of Figure 3.

The geometric structure of these results immediately suggests which heuristics will be near-optimal. The power functions (represented by the convex hulls) for machines 1,3, and 4 have upward curves suggesting that race-to-idle will not be optimal (see Observation 5.5). In contrast, $\mathsf{Machine}_2$ has a nearly linear convex hull indicating that race-to-idle will be close to optimal.

In addition to visually inspecting the power functions, we can evaluate them analytically using Equations 10–14. Table II shows the points that make up the convex hull for each machine[1]. Assuming we have a performance requirement equal to 50% of the maximum performance on each machine, then we can easily solve Equations 10–14. Indeed, the analytical results confirm our visual interpretation. We see that 1) not idling will be optimal (with a small energy savings) on $\mathsf{Machine}_1$, 2) race-to-idle will be nearly optimal on $\mathsf{Machine}_2$, and 3) not idling will be optimal (with significant savings) on $\mathsf{Machine}_3$ and $\mathsf{Machine}_4$.

We note that energy is saved here because the required work rate is lower than the maximum available work rate. Obviously, as the required work rate approaches the maximum, the energy savings of all schedulers will converge. However, there are many cases where systems are over-provisioned and do not need to or cannot run at their maximum work rate. For example, Google data centers have been shown to spend most of their time at around 30-40% utilization, but they must be on to provide responsiveness in rare times of heavy load [7]. Additionally, the Exynos 5 processor (in the Galaxy S4) has a 5.5 Watt peak power consumption, but that is nearly twice the sustainable power [28]. While one system is a datacenter and the other an embedded device, both are over-provisioned and will spend considerable time operating below their maximum work rate. In both systems it is essential to determine how to run below the maximum work rate and maintain predictable performance (leading to user satisfaction), while minimizing energy to reduce costs (data center) or extend battery life (phone).

C. Energy Savings Comparison

We test the energy savings of different heuristics by integrating them into the PTRADE runtime [18]. PTRADE uses feedback control to allocate resources such that applications meet real-time constraints and minimize energy. PTRADE's runtime system uses heuristics to continuously solve an optimization problem equivalent to that of Equations 1–4.

To perform an empirical evaluation of our optimization framework, we modify the PTRADE runtime to use different heuristic solutions to Equations 1–4. Specifically, we evaluate the heuristics from Section III-B: $\mathsf{NaiveRace}$, race, pace, and $\mathsf{no-idle}$. In addition, we evaluate one version of PTRADE modified to solve Equation 14 using the true optimal solution.

We deploy PTRADE on each machine. Then, for each benchmark, we assign a performance constraint (equal to twice the time it would take to complete in the fastest configuration). We then record the energy consumption for each combination

[1]As an aside, it is somewhat surprising how few configurations appear on the convex hull for each machine. This data indicates that most configurations are not Pareto-optimal for a particular application.

TABLE I. HARDWARE PLATFORMS USED IN EVALUATION.

Name	Processor	Cores	Memory Controllers	Speeds (GHz)	TurboBoost	HyperThreads	Idle Power (W)	Total Configurations
Machine$_1$	Xeon X5460	8	1	2.000–3.160	no	yes	200	33
Machine$_2$	Xeon E5520	8	2	1.596–2.395	yes	yes	90	57
Machine$_3$	Xeon E5-1650	6	1	1.2–3.2	yes	yes	85	145
Machine$_4$	Xeon E5-2690	16	2	1.2–2.9	yes	yes	75	1025

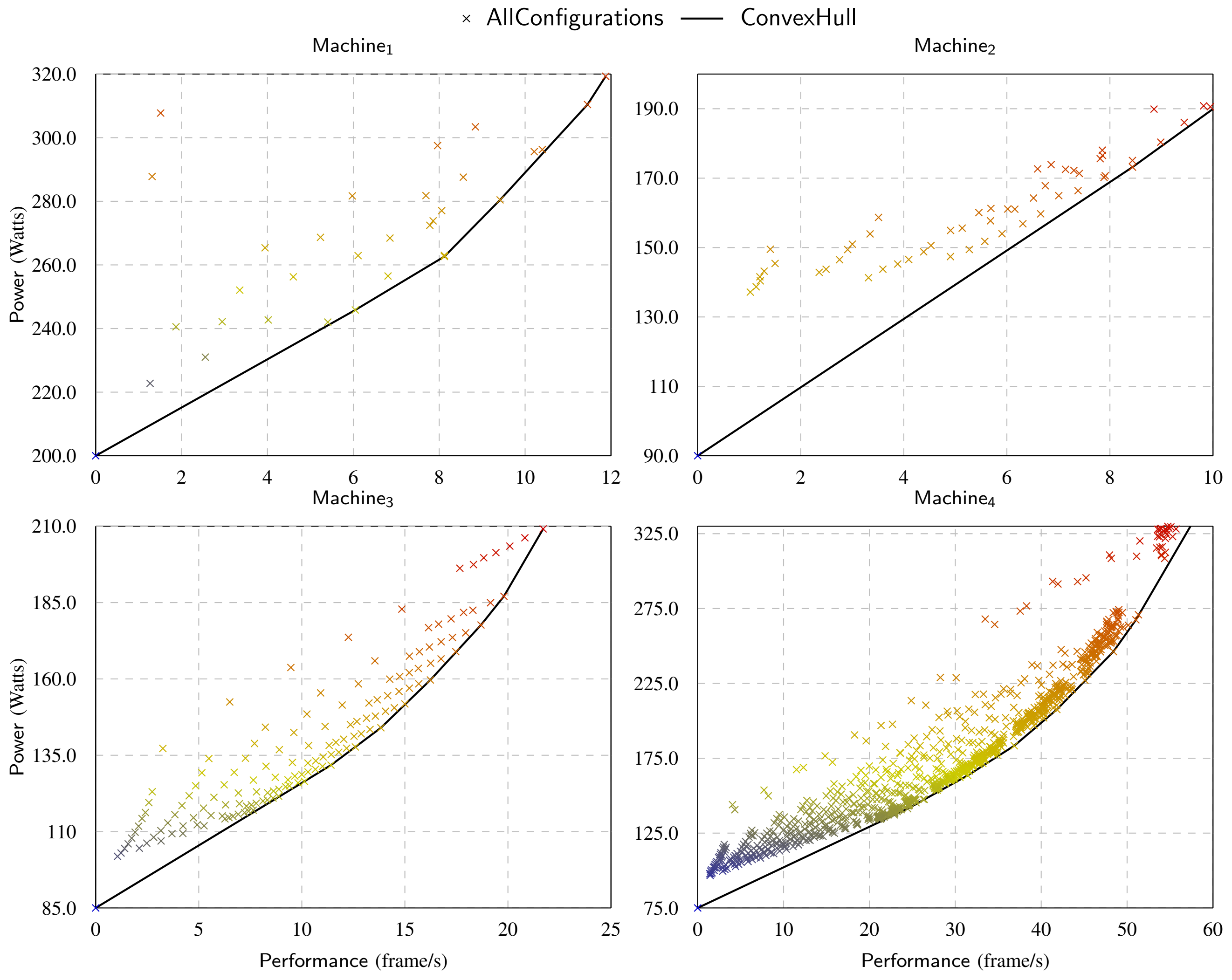

Fig. 5. Power functions for x264 on different machines.

of benchmark, hardware platform, and resource allocation strategy.

The results of this study are illustrated in Figure 6. Each bar chart shows the results for a different machine. The x-axis shows the application and the y-axis shows the energy consumption. For each benchmark, there is a bar showing the measured energy consumption under each heuristic. All results are normalized the the optimal energy consumption for that application on that hardware platform. Therefore, the lowest possible energy consumption is unity.

The results indicate that the resource allocation strategy makes only a small difference on Machine$_1$ and Machine$_2$. On both these machines race-to-idle is very close to optimal and the choice of strategy does not appear to make a large difference. In contrast, the choice of strategy starts to become significant on Machine$_3$. Here, race-to-idle consumes an average of 12% more energy than optimal. Pace-to-idle is slightly better at 8% more than optimal, but the no-idle strategy is very close to optimal, within 1%. Finally, Machine$_4$ shows the biggest impact of strategy choices. Here, race-to-idle consumes an average of 29% more energy than optimal. Pace-to-idle is significantly better, consuming only 7% beyond optimal, while the no-idle heuristic is closest to optimal at just over a 3% increase on average. The numeric results are included in Table III.

D. Discussion

These result provide empirical evidence that race-to-idle is not sufficient for minimizing energy consumption. In fact, no one heuristic is optimal on all machines or for all applications; however, pace-to-idle is generally good and always

TABLE II. POINTS ON THE CONVEX HULL FOR EACH MACHINE.

System	Points on the Convex Hull
Machine_1	(0, 200.0), (6.0, 245.8), (8.1, 262.6), (9.4, 280.4), (11.5, 310.4), (11.9, 319.3)
Machine_2	(0, 90.0), (8.4, 173.2), (10.5, 195.1)
Machine_3	(0, 85.0), (10.1, 126.5), (11.4, 131.8), (13.8, 144.0), (16.2, 159.7), (18.7, 177.7), (19.8, 187.1), (21.7, 109.1)
Machine_4	(0.0, 75.0), (24.4, 141.3), (31.4, 163.5), (36.9, 183.4), (41.8, 207.6), (48.4, 246.3), (51.0, 267.5), (58.4, 339.6)

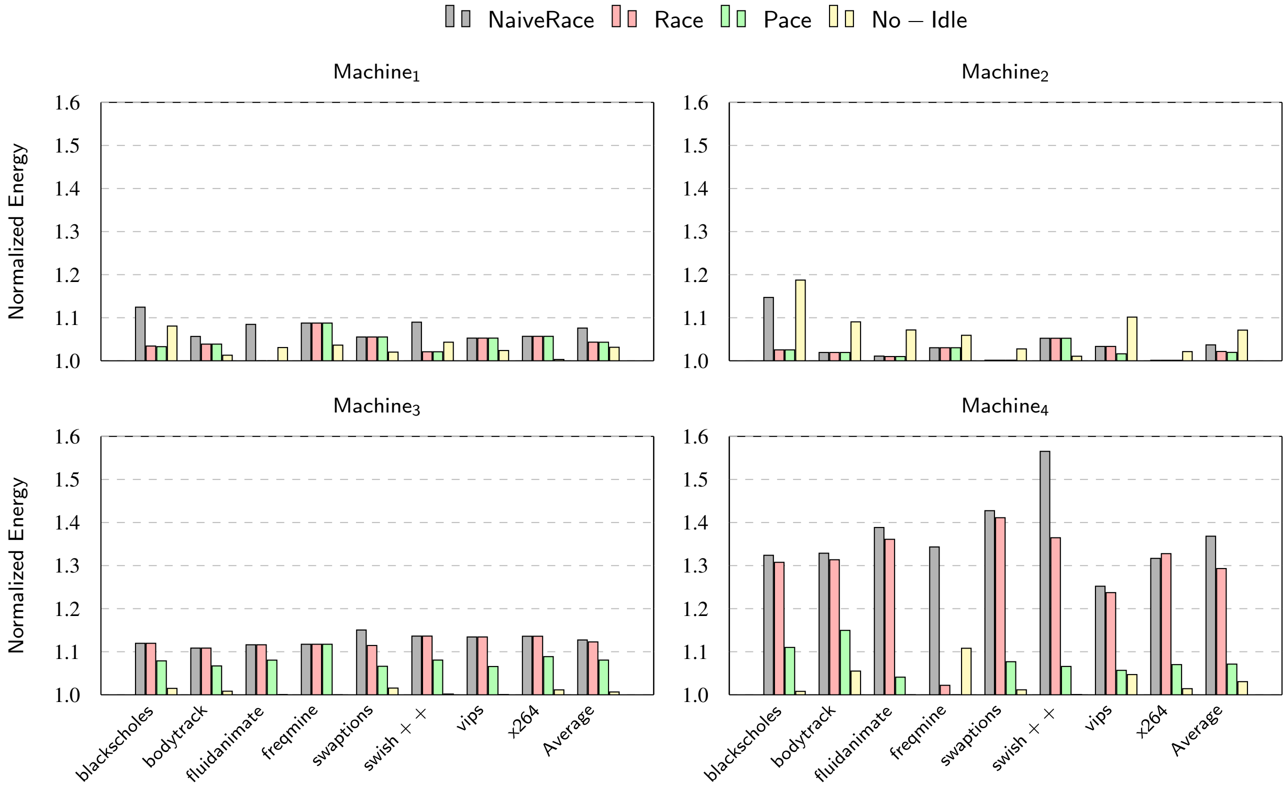

Fig. 6. Energy consumption of different resource allocation strategies on different hardware platforms. (Lower is better.)

TABLE III. AVERAGE ENERGY CONSUMPTION OF DIFFERENT HEURISTICS, 1 IS OPTIMAL.

	Average Energy Consumption			
System	NaiveRace	race	pace	no − idle
Machine_1	1.08	1.04	1.04	1.03
Machine_2	1.04	1.02	1.02	1.07
Machine_3	1.13	1.12	1.08	1.01
Machine_4	1.37	1.29	1.07	1.03

outperforms race-to-idle (validating the analytical framework). These results also confirm that low idle power does not imply that race-to-idle is optimal. Indeed, race-to-idle is worst on Machine_4, which has the lowest idle power.

Furthermore, year of manufacture seems to matter for energy consumption. Machine_1 was purchased in 2008 and Machine_2 was purchased in 2010. On both these machines race-to-idle is close to optimal, and the difference between race-to-idle and optimal is within 5%, making it unlikely that the benefit of more sophisticated strategies will outweigh their cost. In contrast, Machine_3 was purchased in 2012 and Machine_4 was purchased in 2013. While this is not enough data to confirm a trend, it seems possible that future machines will continue to expose a greater range of power and performance tradeoffs making more sophisticated strategies worthwhile.

VII. CONCLUSION

This paper explores the well-studied problem of allocating resources to minimize energy while respecting real-time performance constraints. This important problem requires understanding both algorithmic policies and the practical concerns that may limit these algorithms in on real platforms. As hardware and computer systems evolve it is important to revisit earlier assumptions and practices. Toward this end, the paper presents a combined analytical and empirical approach to studying this resource allocation problem. In particular, we have focused on heuristic solutions which can be incorporated into existing systems. We find that the well-known *race-to-idle* heuristic is consistently bettered (in both theory and practice) by the pace-to-idle heuristic. Additionally, we find that often not-idling is closer to optimal than an idling-based strategy. We believe that as future hardware exposes more and more configurations governing power/performance tradeoffs, similar studies will need to be repeated and resource allocation algorithms must be re-evaluated. We note that the dwindling practical efficacy of race-to-idle puts greater burden on the problem of implementing resource schedulers. Race-to-idle is particularly easy to implement in practice, but more sophisticated solutions require increasing computational complexity and greater understanding of the power/performance tradeoffs exhibited by a combination of application and system.

Acknowledgments: This work was performed under the Argo project sponsored by the U.S. Department of Energy (contract DE-AC02-06CH11357).

References

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

[2] S. Albers and A. Antoniadis. "Race to idle: new algorithms for speed scaling with a sleep state". In: *SODA*. 2012.

[3] H. Aydi, P. Mejía-Alvarez, D. Mossé, and R. Melhem. "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems". In: *RTSS*. 2001.

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

[5] N. Bansal, H.-L. Chan, T. W. Lam, and L.-K. Lee. "Scheduling for Speed Bounded Processors". In: *ICALP*. 2008.

[6] N. Bansal, H.-L. Chan, and K. Pruhs. "Speed Scaling with an Arbitrary Power Function". In: *ACM Transactions on Algorithms* 9.2 (2013).

[7] L. A. Barroso and U. Hlzle. "The Case for Energy-Proportional Computing". In: *IEEE Computer* 40 (2007).

[8] A. Bhavnagarwala, X. Tang, and J. Meindl. "The impact of intrinsic device fluctuations on CMOS SRAM cell stability". In: *Solid-State Circuits, IEEE Journal of* 36.4 (2001).

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

[10] E. Bini, G. C. Buttazzo, and G. Lipari. "Minimizing CPU energy in real-time systems with discrete speed management". In: *ACM Trans. Embedded Comput. Syst.* 8.4 (2009).

[11] S. Bradley, A. Hax, and T. Magnanti. *Applied mathematical programming*. Addison-Wesley Pub. Co., 1977.

[12] A. Carroll and G. Heiser. "Mobile Multicores: Use Them or Waste Them". In: *Proceedings of the 2013 Workshop on Power Aware Computing and Systems (HotPower'13)*. Farmington, PA, USA, 2013, p. 12.

[13] H.-L. Chan, J. W.-T. Chan, T. W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong. "Optimizing throughput and energy in online deadline scheduling". In: *ACM Transactions on Algorithms* 6.1 (2009).

[14] 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).

[15] S. Funk, V. Berten, C. Ho, and J. Goossens. "A Global Optimal Scheduling Algorithm for Multiprocessor Low-power Platforms". In: *RTNS*. 2012.

[16] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 1st. Morgan and Claypool Publishers, 2009.

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

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

[19] 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.

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

[21] 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.

[22] J. D. Lin, W. Song, and A. M. Cheng. "Real-energy: A New Framework and a Case Study to Evaluate Power-aware Real-time Scheduling Algorithms". In: *ISLPED*. 2010.

[23] 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.

[24] T. Pering, T. Burd, and R. Brodersen. "The simulation and evaluation of dynamic voltage scaling algorithms". In: *ISLPED*. 1998.

[25] P. Pillai and K. G. Shin. "Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems". In: *SOSP*. 2001.

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

[27] S. Saewong and R. Rajkumar. "Practical voltage-scaling for fixed-priority RT-systems". In: *RTAS*. 2003.

[28] Y. Shin, K. Shin, P. Kenkare, R. Kashyap, H.-J. Lee, D. Seo, B. Millar, Y. Kwon, R. Iyengar, M.-S. Kim, A. Chowdhury, S.-I. Bae, I. Hon, W. Jeong, A. Lindner, U. Cho, K. Hawkins, J. Son, and S. Hwang. "28nm High- Metal-Gate Heterogeneous Quad-Core CPUs for High-Performance and Energy-Efficient Mobile Application Processor". In: *ISSCC*. 2013.

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

[30] 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).

[31] C.-Y. Yang, J.-J. Chen, C.-M. Hung, and T.-W. Kuo. "System-Level Energy-Efficiency for Real-Time Tasks". In: *ISORC*. 2007.

[32] F. F. Yao, A. J. Demers, and S. Shenker. "A Scheduling Model for Reduced CPU Energy". In: *FOCS*. 1995.

[33] H. Yun, P.-L. Wu, A. Arya, C. Kim, T. F. Abdelzaher, and L. Sha. "System-wide energy optimization for multiple DVS components and real-time tasks". In: *Real-Time Systems* 47.5 (2011).

Appendix

A. Equivalence of the Convex Hall in the Dual Space and the Power-Performance Space

We show that the configurations forming the edges of the convex hull in the dual space are exactly the configurations that form the convex hull defining the power function.

We prove by induction that given an edge in the dual space and a vertex in the power-performance space, the next incident edge appearing in the convex hall of the dual space is exactly the same configuration of the adjacent vertex of the previous vertex in the convex hull of the power function.

Base Case. In the dual space, the idle configuration creates the unbounded edge of the convex hall at the top left corner. In the power-performance space, the point on the y-axis, at performance $s_0 = 0$, is the idle configuration.

Induction hypothesis: assume the claim holds for all edges and corresponding points in the convex hulls of the two spaces up to the k'th edge and configuration, starting from the idle state. Suppose we are at e_k in the dual space and $c_k = (s_k, p_k)$ in the power-performance space, where e_k has slope $-s_k$ and y-intercept p_k.

Then c_{k+1} in the power-performance space is the configuration which gives $\arg\min_c\{(p_c - p_k)/(s_c - s_k)\}$. The next edge in the dual space e_{k+1} can be shown to be the edge which intercepts with e_k at the minimum x-coordinate. This can be shown to be the constraint which gives precisely the same condition for c_{k+1}. $\square$