Racing and Pacing to Idle: Theoretical and Empirical Analysis of Energy Optimization Heuristics

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

***Abstract*—The problem of minimizing energy for a performance constraint (e.g., real-time deadline or quality-of-service requirement) 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 heuristic solutions, which achieve good results in practice but tend to be system-specific in efficacy. An example is the *race-to-idle* heuristic, which makes all resources available until a task completes and then idles. Theory predicts poor energy savings, but practitioners have reported good empirical results.**

To help bridge the gap between theory and practice, this paper presents a geometrical framework for analyzing the energy optimality of resource allocation under performance constraints. The geometry of the problem allows us to derive an optimal strategy and three commonly used heuristics: 1) *race-to-idle*, 2) *pace-to-idle* a near-optimal idling strategy, and 3) *no-idle* which never idles. We then implement all strategies and test them empirically for seven benchmarks on four different multicore systems, including both x86 and ARM. We find that *race-to-idle* is near optimal on older systems, but can consume as much as $3\times$ more energy than the optimal strategy. In contrast, *pace-to-idle* is never more than 12% worse than optimal.

I. Introduction

Many embedded and real-time computing applications have timing constraints defined by their interaction with the outside world; e.g., to keep up with a sensor or get timely results to a human. As energy concerns have grown increasingly important, an additional goal has arisen: meeting the timing constraints while minimizing energy consumption.

To support energy minimization, modern multicores 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 [32]. 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 [34]. Heterogeneous multicores expose different cores types where each has different power and performance tradeoffs [25].

Prior work has performed theoretical and algorithmic analysis of systems with both DVFS and dyamic power management (DPM), or low-power sleep states [1–3, 5, 9, 12, 15, 17, 39]. As noted above, however, modern multicores contain additional power saving mechanisms (such as heterogeneous core types) and their evaluation has thus far been largely empirical [6, 11, 14, 16, 18, 19, 21, 26–28, 31].

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* [6, 18, 19, 31]. The beauty of this heuristic is its simplicity: it does not require any runtime optimization calculations. Instead, it simply makes all resources available when a task enters the system, and then idles 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 [11, 19, 22, 27, 28]. 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 geometrical framework performance-constrained resource allocation strategies on multicores. We cast this problem as a linear-optimization and reason about the structure of optimal solutions. The key insight to this approach is its generality: it can describe DVFS and DPM based systems, but can also be used to evaluate multicores with power-gating, heterogeneous core types, and allocatable un-core resources such as memory.

Specifically, the geometric interpretation indicates that for any application and performance constraint, there is an energy-minimal solution which uses at most two configurations out of the entire configuration space (including all core types, speeds, un-core resources, etc). This realization motivates the definition of a family of solutions to the resource allocation problem. Each solution differs in how it selects the two states used to meet the performance constraint. The well-known race-to-idle strategy is simply a heuristic solution that uses the idle configuration and the configuration that allocates all resources to an application. In addition, we derive an optimal idling strategy, called *pace-to-idle*, which is the best possible strategy that uses the low-power idle state (i.e., DPM) as one of its configurations. We also derive a *no-idle* strategy that never uses the idle state. Finally, we derive an optimal strategy that selects two states with a minimal energy allocation.

Weimplement these strategies on four different multicore platforms, including three x86 and one ARM big.LITTLE system. We test each strategy on each platform with seven different parallel benchmark applications. Our empirical results confirm that the pace-to-idle strategy always beats race-to-idle

This work was funded by the U.S. Government under the DARPA PERFECT program, by the Dept. of Energy under DOE DE-AC02-06CH11357, and by the NSF under CCF 1439156. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

– by as much as 20% on x86 and 3× on ARM.

This paper makes the following contributions:

1) Proposing a geometric interpretation for allocating different resources to meet a timing constraint with minimal energy consumption. (See Section III-B.)
2) Deriving an optimal resource allocation algorithm and formal definitions of 3 common heuristics. (See Section IV.)
3) Deriving conditions under which processor idling (i.e., DPM) will result in optimal energy consumption. Importantly, this analysis also indicates when systems should not be idled. (See Section VI, Observation 6.1.)
4) Showing that the conditions where race-to-idle is optimal are degenerate, and thus, may not persist in future machines. (See Section VI, Observation 6.6.)
5) Proving that there is a near-optimal idling strategy, called *pace-to-idle* that is analytically and empirically better than race-to-idle. (See Theorem 6.4 and Section VII.)
6) Validating pace-to-idle empirically by demonstrating it achieves a 20% energy savings compared to race-to-idle on x86 and 3× energy reduction on ARM. Additionally, results indicate that pace-to-idle's energy consumption is never more than 12% greater than the true optimal strategy (See Section VII-C).
7) Empirical demonstration that the proposed technique outperforms DVFS and DPM based optimizations on modern systems. On our newest server system, solutions using only DVFS and DPM consume an average of 30% more energy than the proposed technique. On the ARM system, DVFS and DPM consume an average of 2.3× more energy than the proposed approach. **Our proposed approach saves energy because it is general enough to combine DVFS and DPM with reduced core usage and heterogeneous core scheduling.** (See Section VII-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 derives the race-to-idle, pace-to-idle, no-idle, and optimal resource allocation strategies from this geometrical interpretation. Section V describes how to construct a convex function describing available performance and power tradeoffs. Section VI observes several practical implications of the proposed problem formulation. Section VII presents our empirical evaluation. Section VIII 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 optimizing energy consumption under performance constraints for variable-speed processors. Such processors are equipped with *dynamic voltage and frequency scaling* (DVFS), which allows to dynamically set the speed/frequency depending on the current workload [24, 32]. 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 [24].

In this framework, theoretical models formulate the optimization problem as a deadline-based job scheduling problem incorporating DVFS [1, 3–5, 9, 17, 24, 33, 39]. 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 an implemented system. 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 [34]), 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 $P(s)$.

B. Practical Concerns and Empirical Studies

Algorithms based on theoretically-founded models are often not applicable in practice. In fact, researchers have found that 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[6, 31]. Part of the reason is that CMOS voltage and frequency scaling for combinational logic (that gives $P \propto v^2 f$) does not apply to tradition SRAM circuits, as reducing the voltage in these circuits leads to unacceptable error rates [7]. 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 [6, 18] and accounting for these other components has a dramatic effect on performance-constrained systems [13, 38, 40]. 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 VII).

Several empirical studies investigate the practical constraints of minimizing energy for a performance bound. Studies done in the early and middle part of the 2000's reinforce the notion that theoretical results are hard to realize in practice and race-to-idle is often near-optimal [6, 28, 31, 35]. More recent studies suggest that this trend is starting to reverse and they call into question the efficacy of the race-to-idle heuristic [11, 16, 19, 22, 27, 30]. 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 multicores (both homo- and heterogeneous) 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 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 formulate the dual LP, allowing a geometric interpretation of the problem and simple construction of the optimal solution.

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 $c \in \{0, \ldots, C-1\}$, 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 $0 \leq t_c \leq t$ 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, 20, 37]. Thus, the problem of minimizing energy while completing the work can be expressed as:

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

$$subject\ to$$

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

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

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

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

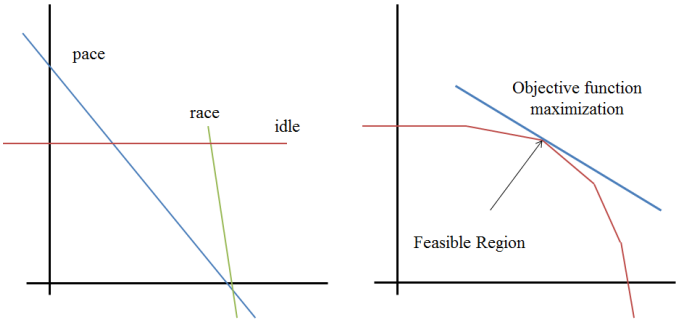

Fig. 1. (1) race, pace, and idle configurations in the dual space and (2) optimization in the convex hull formed by all configurations

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.

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 area under a line in the 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. 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.

We define three configurations of particular interest and describe their relationship to the convex hull. The *race* configuration is that with the highest performance. The *pace* configuration is that with the highest energy efficiency (race and pace may be equivalent, but often are not). The *idle* configuration is that in which the processor performs no work at all (this can correspond to a low-power sleep state). Figure 1 illustrates the interpretation of these constraints in the dual space. The race configuration corresponds to the steepest line, the pace configuration is the line with the smallest x-intercept, and the idle configuration is a horizontal line.

Observation 3.1: The configurations *race*, *pace*, and *idle* always appear on the convex hull of the feasible space.

This observation easily follows from the fact that these configurations represent extreme behaviors and thus can never be discarded by tighter constraints.

The convex hull is precisely the search space for the dual LP, where we find the point (x, y) which maximizes the

Algorithm 1 The Race-to-Idle Heuristic.

Require: $W \leftarrow$ a workload
Require: $t \leftarrow$ a deadline
Require: $C \leftarrow$ a set of C system configurations
$race = C - 1$
$idle = 0$
$t_{race} = \frac{W}{r_{race}}$
$t_{idle} = t - t_{C-1}$
$t_c = 0, \forall c \neq C - 1, idle$

objective function, $W \cdot x + t \cdot y$. Given the convex hull formed by the C constraints, finding the optimal point is simple as illustrated in Figure 1. 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 on it).

C. 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.

Observation 3.2: Optimal solutions use ≤ 2 configs.

From standard polyhedral theory, one can argue directly that a basic solution has number of non-zero values at most the number of constraints, which is two in our dual LP. Formally, 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-zero variables in the primal, and the non-binding constraints to variables that are zero [10].

Observation 3.2 is the key to understanding the rest of the paper. Practically, this observation states that any energy optimal resource allocation will spend time in at most two configurations. Thus, we can derive a series of solutions – from heuristics to the true optimal – that consider only two configurations. Further, we can combine observations 3.1 and 3.2 to develop a sound basis for deriving heuristic solutions to the optimization problem. In the next section, we derive both these heuristic solutions and the true optimal.

IV. Resource Allocation Strategies

In this section we use Observations 3.1 and 3.2 to derive a family of algorithms for resource allocation. We first derive the race-to-idle algorithm, then an optimal idling strategy (called pace-to-idle), a strategy that never idles, and finally a true optimal (i.e., minimal energy solution). For each algorithm, we describe how we select the two states that will comprise the solution, how we set the times to spend in each state, and then describe the computational complexity as well as implementation concerns.

A. Race-to-idle

This well-know heuristic provides all resources (i.e., use $C-1$) until the task completes and then idles the system until the next task is ready. We list the algorithm as Algorithm 1.

This solution is easy to implement in practice since it does not require any analysis of the characteristics of different configurations. That is, it can be implemented without actually knowing the values of s_c and p_c. In practice this heuristic is implemented by just waiting for the task to complete and then transitioning to the idle state. It thus requires $O(1)$ operations.

Algorithm 2 The Pace-to-Idle Heuristic.

Require: $W \leftarrow$ a workload
Require: $t \leftarrow$ a deadline
Require: $C \leftarrow$ a set of C system configurations
$pace = \arg\max_{c \in C}\{s_c/p_c : s_c \geq W/t\}$
$idle = 0$
$t_{pace} = \frac{W}{s_{pace}}$
$t_{idle} = t - t_{C-1}$
$t_c = 0, \forall c \neq C - 1, idle$

Algorithm 3 The No-Idle Heuristic.

Require: $W \leftarrow$ a workload
Require: $t \leftarrow$ a deadline
Require: $C \leftarrow$ a set of C system configurations
$hi = \arg\min_c\{p_c : s_c \geq W/t\}$
$lo = \arg\max_c\{s_c/p_c : s_c \leq W/t\}$
$t_{hi} = \frac{W - s_{lo} \cdot t}{s_{hi} - s_{lo}}$
$t_{lo} = \frac{s_{hi} \cdot t - W}{s_{hi} - s_{lo}}$

B. Pace-to-idle

This heuristic (Algorithm 2) runs in the most energy efficient configuration, *pace*, until the task is complete and then idles. In the case that *pace* cannot complete the workload in the given deadline, it chooses the most energy efficient configuration from those which have high enough work rates.

For small enough workloads, *pace* is fixed at the globally most energy efficient state. For higher workloads, this requires finding the local *pace* configuration. This requires scanning the configuration space for the *pace* configuration. A straightforward approach would search all configurations ($O(C)$ operations).

C. No-idle

This heuristic (Algorithm 3) completes the task at the deadline and never enters the idle state. It first forms two sets of configurations, one with work rates above the target and the second with work rates below. From the first set, it chooses hi the configuration from the first set with the lowest power. Note that this does not imply energy efficiency, as there may be other configurations using more power with even greater work rates increases. From the second set of configurations, this heuristic chooses *lo*, the most energy efficient state with work rate lower than the minimum required to complete the workload in the deadline.

The heuristic alternates between *hi* and *lo* such that the system never idles, and finishes exactly at the deadline using the two states. This requires scanning the configuration space to compute the *hi* and *lo* configurations, $O(C)$ operations .

D. Optimal

Observation 3.2 tells us an optimal solution can be obtained using only two configurations. In fact, we can exactly characterize the two configurations as the two corresponding constraints on the convex hull which have the closest slopes to $-W/t$ of the objective function from above and below, as in Figure 2.

Algorithm 4 The Minimal Energy Resource Allocation.

Require: $W \leftarrow$ a workload
Require: $t \leftarrow$ a deadline
Require: $C \leftarrow$ a set of C system configurations
$C' =$ the set of configurations on the convex hull of the dual space
$over = \arg\min_{c \in C'}\{s_c : s_c \geq W/t\}$
$under = \arg\max_{c}\{s_c : s_c \leq W/t\}$
$t_{over} = \dfrac{W - s_{hi} \cdot t}{s_{hi} - s_{lo}}$
$t_{under} = \dfrac{s_{hi} \cdot t - W}{s_{hi} - s_{lo}}$

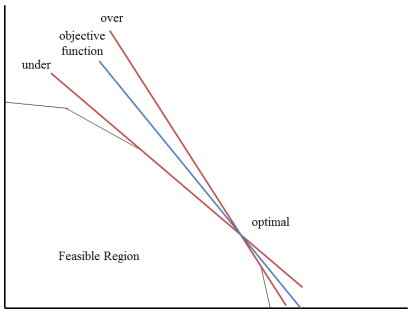

Fig. 2. Maximizing the objective function in the dual space

If C' is the set of configurations that comprise the convex hull in the dual space, then we have the following optimal configurations:

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

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

The optimal algorithm (Algorithm 4) computes the set of configurations forming the convex hull, then finds *over* and *under*. Once the two configurations have been found, it allocates time to the two configurations to get the work done exactly at the deadline. The times allocated to each configuration satisfy

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

$$t_{over} + t_{under} = t \tag{11}$$

which gives us

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

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

For implementation, one can precompute the convex hull C' in $|C| \log |C|$ time and for each instance of a workload and a deadline iterate through C' to find the *over* and *under* configurations.

V. The Convex Hull and the Power Function

We denote the *power-performance tradeoff space* as the plot of the configurations as points in the 2-dimensional plane, with the x-axis for performance and the y-axis for power. Intuitively, the best set of configurations in the power-performance tradeoff space is those forming a convex boundary in this space. It is shown in the Appendix that the configurations (constraints/edges) forming the convex hull in the dual space is equivalent to the configurations forming the convex boundary in the tradeoff space.

Thus, when analyzing heuristics, we may assume a convex, piecewise-linear power function whose range is bounded by the idle and race configurations. Note that we are assuming no switching costs for the system to alternate between different configurations: for any desired configuration in the line between two others in the tradeoff space, we may obtain the same outcome by taking the convex combination of the two.

Observation 5.1: Let $P(\cdot)$ be the convex, piecewise-linear power function, equivalent to the convex hull of the dual space. Then equations (12) and (13) directly imply that given W and t, the optimal energy consumption is $P(s_{avg}) \cdot t$, where $s_{avg} = W/t$.

VI. 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 and also show how race-to-idle and pace-to-idle may deviate far from the optimal. Then we give a simple condition to compare idling heuristics and show that pace-to-idle will always consume less energy than race-to-idle.

A. Optimal vs an idling heuristic

Let $P(s)$ be the 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 observation 5.1, we have $OPT = P(s_{avg}) \cdot t$, where the system possibly achieves minimal energy consumption with a combination of two states. On the other hand, a c-to-idle heuristic would complete W at c for time $t_c = (W/s_c)$ and idle for the remaining time. The total energy consumption of c-to-idle is therefore $ALG = P(s_c) \cdot (W/s_c) + P_{idle} \cdot (t - W/s_c)$, where ALG is the energy consumption of c-to-idle, P_{idle} is the power consumption of the *idle* configuration, and s_c is the work rate of c. Rearranging the terms, we have

$$OPT = P_{idle} \cdot t + (P(s_{avg}) - P_{idle}) \cdot t \tag{14}$$

$$ALG = P_{idle} \cdot t + (P(s_c) - P_{idle}) \cdot (W/s_c). \tag{15}$$

This directly gives the condition for optimality of an idling heuristic.

Observation 6.1 (Optimality condition): c-to-idle heuristic is optimal iff:

$$\frac{P(s_c) - P_{idle}}{s_c} = \frac{P(s_{avg}) - P_{idle}}{s_{avg}} \tag{16}$$

i.e., $(s_{avg}, P(s_{avg}))$ and $(s_c, P(s_c))$ are on a line from $(0, P_{idle})$ in the power-performance space.

Observation 6.2 (Optimality of Idling Heuristics): Let $s_{avg} = W/t$ and $e_{avg} = s_{avg}/P(s_{avg})$. Using (Equation 14) and (Equation 15), we get:

$$\frac{ALG}{OPT} = \frac{P_{idle}}{P(s_{avg})}\left(1 - \frac{s_{avg}}{s_c}\right) + \frac{e_{avg}}{e_c}. \tag{17}$$

The first term on the right-hand side of (Equation 17) can be interpreted as the penalty of ALG for using the *idle* configuration when $P_{idle} > 0$. The second term represents the penalty/gain coming from c having a worse/better energy efficiency compared to the optimal configuration.

Theorem 6.3 (Non-optimality of Idling Heuristics): Unless the power function of a given machine is a straight

line originating from the idle state, $(0, P_{idle})$, an idling heuristic is never optimal for all instances.

Simply put, unless the optimal configuration is a convex combination of c and *idle*, c-to-idle cannot be optimal.

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 can complete the given workload in the deadline, then idles. The *pace* configuration is characterized by having the highest energy efficiency, i.e., $e_{pace} \geq e_c$ for all feasible c with $s_c \geq s_{avg}$. The *race* configuration is the unique state with the highest work rate, i.e., s_{race} has the maximum work rate over all configurations.

From the optimality of idling heuristics in (Equation 17), we can directly compute and compare the optimality of two idling heuristics, in particular race-to-idle and pace-to-idle:

Theorem 6.4: Pace-to-idle is always better than race-to-idle.

Proof. By definition of *race* and *pace*, the penalty of using the idle configuration in the first term of the right-hand side in (Equation 17) is larger for race-to-idle than pace-to-idle. For the second term of the right-hand side in (Equation 17), by definition $e_{pace} \geq e_{race}$. □

We make observations of some special cases.

Observation 6.5: If $P_{idle} = 0$, pace-to-idle is the best idling strategy, but may still not be the optimal.

Proof. Clearly, (Equation 17) tells us that only the power efficiency, e_c, matters when $P_{idle} = 0$. *pace* was defined to be the configuration which achieves the best power efficiency, so the first part of the claim follows. Also, if *pace* is not the globally most energy efficient configuration, then $(s_{avg}, P(s_{avg}))$ will not be a convex combination of $(0, P_{idle})$ and (s_{pace}, P_{pace}), and pace-to-idle will not be optimal, despite being the best idling heuristic. □

Observation 6.6: Unless the power function is linear or close-to-linear, race-to-idle will be very far from the optimal, especially for workload and deadlines requiring a small average work rate. Thus, low idle power does not necessarily imply that race-to-idle is close to optimal. Rather, the optimality of racing is determined by the performance target and the convexity of the power function.

Simply put, with high convexity in the power function and a small, required average work rate, it is a bad strategy to complete the work so fast and transition to an idle state. Rather, as shown by OPT, it is better to alternate in two states which stay close to the average work rate required and steadily process the workload.

VII. Empirical Study

This section evaluates the algorithms derived in Section IV on 4 real hardware platforms with 7 different applications.

A. Machines and Applications

We use four hardware platforms, summarized in Table I. On Server1, idling consumes 69% of the lowest measured active power (90 W idle, 131 W lowest active power). On Desktop, idling consumes 85% of the lowest active power (85W to 100W). On Server2, idling consumes 75% of the lowest power (75W to 100W). On Mobile idling consumes 70% of the lowest power (0.12W to 0.17W). The highest measured power consumption for these machines are 225W, 235W, 430W, and 6W respectively. To measure power consumption, all machines are connected to power meters which report total system power consumption.

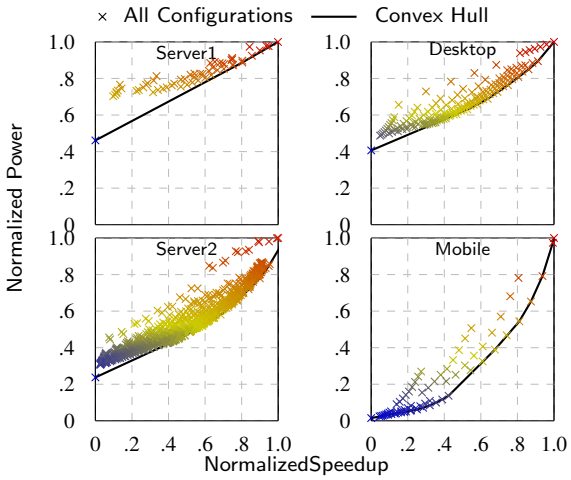

Fig. 3. Power functions for x264 on different machines.

We evaluate seven applications: blackscholes, bodytrack, facesim, ferret, swaptions, vips, and x264 from PARSEC [8]. These benchmarks are broadly representative of performance sensitive applications. blackscholes and swaptions price financial instruments and represent computations with latency constraints. bodytrack and x264 process video data and must meet a camera' throughput and latency constraints. facesim must render graphics at a predictable frame rate. ferret is a search engine for media (images, rather than text) and must meet latency constraints to satisfy users. vips is a high-end printing application which has a latency requirement.

B. Power Functions

The geometric representation 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 3. The figure consists of a plot for each machine showing speedup on a normalized scale on the x-axis and normalized power consumption on the y-axis. These plots are normalized so that unity represents maximum speedup or maximum power, allowing comparison of geometries across platforms. The points represent each possible configuration. The solid lines represent the convex hull of optimal solutions to resource allocation problems.

The geometric structure of these figures immediately suggests which algorithms will perform well on each machine. Server1 has a nearly linear convex hull, so Observation 6.6 suggests that Algorithm 1 will be near-optimal. Desktop and Server2 have greater convexity, suggesting that Algorithm 3 will be the better heuristic approach. Interestingly, Mobile has both very low idle power and a highly convex power function and demonstrates a case where Observation 6.5 holds and pace-to-idle is not optimal despite low idle power.

We evaluate these tradeoffs analytically using Equations 16–17. Table II shows the points that make up the

TABLE I. HARDWARE PLATFORMS USED IN EMPIRICAL EVALUATION.

Name	Processor	Big Cores	Little Cores	Mem. Controllers	Speeds (GHz)	TurboBoost	HyperThreads	Idle Power (W)	Configs.
Server1	Xeon E5520	8	0	2	1.596–2.395	yes	yes	90	57
Desktop	Xeon E5-1650	6	0	1	1.2–3.2	yes	yes	85	145
Server2	Xeon E5-2690	16	0	2	1.2–2.9	yes	yes	75	1025
Mobile	Exynos5 Octa	4	4	1	.8–1.6 (A15) .5-1.2(A7)	no	yes	0.12	69

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

System	Points on the Convex Hull
Server1	(0, 90.0), (8.4, 173.2), (10.5, 195.1)
Desktop	(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)
Server2	(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)
Mobile	(0.0, 0.12), (0.18, 0.43), (0.21, 0.49), (0.23, 0.57), (0.26, 0.67), (0.29, 0.79), (.32, .93), (0.35, 1.08), (0.55, 3.08), (0.66, 4.32), (0.72, 5.17), (0.77, 6.28), (0.81, 7.72, (0.82, 7.93)

convex hull for each machine[1]. Assuming we have a performance requirement equal to the worst case latency using maximum resources on each machine, then we can easily solve Equations 16–17. Indeed, the analytical results confirm our visual interpretation: 1) race-to-idle will be nearly optimal on Server1, 2) not idling will be optimal (with significant savings) on Desktop and Server2, and 3) on Mobile, the optimal strategy improves over race-to-idle by over a factor of 3, meaning battery life could be improved by $3\times$ if we switched from idling-based strategies to the optimal algorithm presented here.

Energy is saved here because the required work rate is lower than the maximum. 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 [6]. 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 [36]. 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 [20]. PTRADE uses feedback control to allocate resources such that applications meet real-time performance constraints [29]. PTRADE's control system continually computes the *speedup* necessary to ensure performance goals are met. We evaluate the algorithms proposed in Section IV, by integrating them into PTRADE – when the controller produces a desired speedup, our algorithms determine how to turn that speedup into resource usage. In addition to comparing heuristics, we compare to an approach that uses DVFS (dynamic voltage and frequency scaling) and DPM (dynamic power management) only but does not use other features like core scheduling and heterogeneity.

We deploy PTRADE on each machine. Then, for each benchmark, we assign a performance constraint (equal to the worst case latency when using maximum resources). We then record the energy consumption for each combination of benchmark, hardware platform, and resource allocation algorithm.

[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 optimal for a particular application.

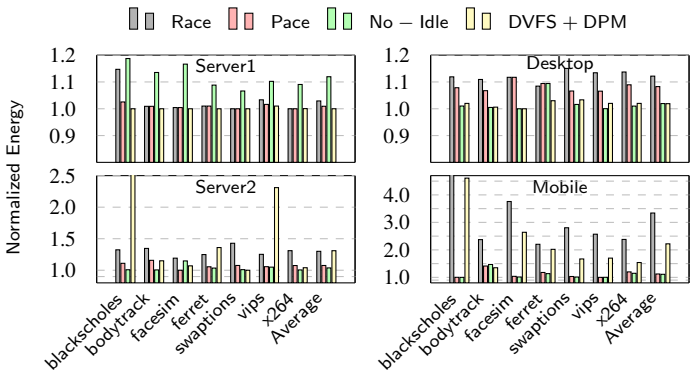

Fig. 4. Energy consumption of different resource allocation strategies on different hardware platforms. (Lower is better.)

The results of this study are illustrated in Figure 4. 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 Server1. On both this machine 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 Desktop. 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, Server2 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.

The proposed approach's average energy consumption is lower than using DVFS and DPM only. On Server1 and Desktop the difference is small and may not be meaningful. However, the energy savings of the proposed approach compared to DVFS and DPM is very large on both Server2 and Mobile. Specifically, on Server2 the proposed approach reduces energy by 30% compared to DVFS and DPM. On Mobile the energy reduction is $2.2\times$. These tremendous energy savings come from increased generality. On Server2 the proposed approach saves energy by scheduling cores to help reduce energy. On Mobile, the proposed approach saves energy by

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

	Average Energy Consumption			
System	race	pace	no − idle	DVFS + DPM
Server1	1.04	1.02	1.07	1.02
Desktop	1.13	1.08	1.01	1.02
Server2	1.37	1.07	1.03	1.32
Mobile	3.34	1.12	1.11	2.21

aggressively scheduling on the more energy efficient LITTLE cores. Systems that combine only DVFS and DPM cannot take advantage of these scheduling savings. These results confirm prior empirical studies showing the importance of scheduling based on core type for heterogeneous systems [23].

D. Discussion

These results 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 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 Mobile, which has the by far the lowest idle power. In addition, on Mobile no heuristic approach is within 10% of optimal energy, making the true optimal algorithm presented here extremely valuable for extending battery life.

Furthermore, these results indicate that relying on DVFS and DPM alone cannot provide optimal energy savings on the more modern systems. On Server2, the optimal energy schedules take advantage of DVFS, DPM and reduced core usage – for applications that do not scale linearly with number of cores this addition provides great energy savings. On Mobile, the optimal energy schedules take advantage of low-power LITTLE cores to save energy. Incorporating these kinds of features will be essential for energy savings under timing constraints on future systems.

VIII. 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 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.

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 et al. "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems". In: *RTSS*. 2001.

[4] N. Bansal et al. "Average Rate Speed Scaling". In: *Algorithmica* 60.4 (2011).

[5] N. Bansal et al. "Speed Scaling with an Arbitrary Power Function". In: *ACM Transactions on Algorithms* 9.2 (2013).

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

[7] A. Bhavnagarwala et al. "The impact of intrinsic device fluctuations on CMOS SRAM cell stability". In: *Solid-State Circuits, IEEE Journal of* 36.4 (2001).

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

[9] E. Bini et al. "Minimizing CPU energy in real-time systems with discrete speed management". In: *ACM Trans. Embedded Comput. Syst.* 8.4 (2009).

[10] S. Bradley et al. *Applied mathematical programming*. Addison-Wesley Pub. Co., 1977.

[11] 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.

[12] H.-L. Chan et al. "Optimizing throughput and energy in online deadline scheduling". In: *ACM Transactions on Algorithms* 6.1 (2009).

[13] 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).

[14] J. Choi et al. "Algorithmic Time, Energy, and Power on Candidate HPC Compute Building Blocks". In: *IPDPS*. 2014.

[15] Z. Du et al. "Energy-Efficient Scheduling for Best-Effort Interactive Services to Achieve High Response Quality". In: *IPDPS*. 2013.

[16] V. W. Freeh et al. "Analyzing the Energy-Time Trade-Off in High-Performance Computing Applications". In: *IEEE Trans. Parallel Distrib. Syst.* 18.6 (June 2007).

[17] S. Funk et al. "A Global Optimal Scheduling Algorithm for Multiprocessor Low-power Platforms". In: *RTNS*. 2012.

[18] 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.

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

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

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

[22] C. Imes and H. Hoffmann. "Minimizing Energy Under Performance Constraints on Embedded Platforms: Resource Allocation Heuristics for Homogeneous and Single-ISA Heterogeneous Multi-Cores". In: *EWiLi*. 2014.

[23] C. Imes et al. "POET: A Portable Approach to Minimizing Energy Under Soft Real-time Constraints". In: *RTAS*. 2015.

[24] S. Irani et al. "Algorithms for Power Savings". In: *ACM Trans. Algorithms* 3.4 (Nov. 2007).

[25] B. Jeff. "Big.LITTLE system architecture from ARM: saving power through heterogeneous multiprocessing and task context migration". In: *DAC*. 2012.

[26] K. Kant et al. "Willow: A Control System for Energy and Thermal Adaptive Computing". In: *IPDPS*. 2011.

[27] 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.

[28] J. D. Lin et al. "Real-energy: A New Framework and a Case Study to Evaluate Power-aware Real-time Scheduling Algorithms". In: *ISLPED*. 2010.

[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] N. Mishra et al. "A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints". In: *ASPLOS*. 2015.

[31] A. Miyoshi et al. "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling". In: *ICS*. 2002.

[32] T. Pering et al. "The simulation and evaluation of dynamic voltage scaling algorithms". In: *ISLPED*. 1998.

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

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

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

[36] Y. Shin et al. "28nm High- Metal-Gate Heterogeneous Quad-Core CPUs for High-Performance and Energy-Efficient Mobile Application Processor". In: *ISSCC*. 2013.

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

[38] C.-Y. Yang et al. "System-Level Energy-Efficiency for Real-Time Tasks". In: *ISORC*. 2007.

[39] F. F. Yao et al. "A Scheduling Model for Reduced CPU Energy". In: *FOCS*. 1995.

[40] H. Yun et al. "System-wide energy optimization for multiple DVS components and real-time tasks". In: *Real-Time Systems* 47.5 (2011).