Minimizing Energy Under Performance Constraints on Embedded Platforms

Resource Allocation Heuristics for Homogeneous and Single-ISA Heterogeneous Multi-Cores

Connor Imes
University of Chicago
ckimes@cs.uchicago.edu

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

ABSTRACT

This paper explores the problem of energy optimization in embedded platforms. Specifically, it studies resource allocation strategies for meeting performance constraints with minimal energy consumption. We present a comparison of solutions for both homogeneous and single-ISA heterogeneous multi-core embedded systems. We demonstrate that different hardware platforms have fundamentally different performance/energy tradeoff spaces. As a result, minimizing energy on these platforms requires substantially different resource allocation strategies. Our investigations reveal that one class of systems requires a race-to-idle heuristic to achieve optimal energy consumption, while another requires a never-idle heuristic to achieve the same. The differences are dramatic: choosing the wrong strategy can increase energy consumption by over 2× compared to optimal.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques; I.2.8 [**Problem Solving, Control Methods, Search**]: Heuristic Methods, Scheduling

General Terms

Experimentation, Measurement, Performance

Keywords

Power-aware computing, energy-aware computing, resource allocation, optimization, heterogeneous architectures

This work was funded by the U.S. Government under the DARPA UHPC program and by the Dept. of Energy under DOE DE-AC02-06CH11357. 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.

EWiLi'14, November 2014, Lisbon, Portugal.

1. INTRODUCTION

Embedded systems require predictable performance such as real-time or quality-of-service guarantees to meet their design requirements. At the same time, these systems are often limited by available energy. Embedded systems designers must therefore contend with the constrained optimization problem of meeting performance guarantees while minimizing energy consumption.

To support systems designers, embedded processors are becoming increasingly configurable. These processors expose a number of components which can be managed in software to change the system's performance/power tradeoffs. Embedded operating systems could potentially reduce the burden of energy minimization by automatically scheduling resource usage to meet the performance constraint and minimize energy.

Unfortunately, scheduling multiple resources, each with their own unique power and performance tradeoffs, is a difficult problem. In practice, most designers turn to heuristic solutions that ensure the performance targets are met and approximate the minimal energy solution. The *race-to-idle* heuristic, in particular, is often employed because it is easy to implement and has achieved acceptable results in practice. This heuristic simply makes all resources available to an application until it completes and then idles the system, taking advantage of low-power states.

As the diversity of embedded systems increases, it is increasingly unlikely that any single heuristic will perform well across all systems. In this paper, we investigate heuristics for scheduling on configurable embedded architectures to meet application performance constraints and minimize energy. We first discuss a formulation of the scheduling problem as a linear program which is general enough to capture scheduling across a variety of configurable systems. We then discuss two heuristics and describe their relationship to the linear program. Specifically, we compare *race-to-idle* to *never-idle*, which tries to keep the system busy until the task deadline. We capture empirical data from four benchmarks on two embedded systems and derive an oracle representing optimal energy consumption for each benchmark and system. We then model the energy consumption of the race-to-idle and never-idle heuristics and compare them to the oracles.

Our two systems are: a Sony Vaio tablet, with an Intel Haswell processor, and an ODROID development board, with an ARM big.LITTLE processor. Our results indicate that no single heuristic is effective on both platforms, and the differences are striking. On the Vaio, the race-to-idle heuristic is near optimal, while the never-idle heuristic consumes 28% more energy than optimal on average. In contrast, on the ODROID, never-idle is within 15% of optimal, while race-to-idle consumes 2.43× more energy. These results hold despite the fact that the ODROID's idle power consumption is only a small fraction of the Vaio's. Although counter-intuitive, the race-to-idle approach is actually better on the system with higher idle power. Our study indicates that a general, portable resource allocator for embedded systems must be flexible enough to match the resource allocation strategy to the characteristics of the hardware.

2. BACKGROUND

Energy optimization under performance constraints is a widely studied problem both in theory and in practice.

Embedded hardware platforms support energy management by exposing *configurable* components. For example, dynamic voltage and frequency scaling (DVFS) gives software control over processor speed [20, 22], which can then be tuned to ensure that performance targets are met while energy is minimized. Theoretical results have produced optimal uniprocessor scheduling algorithms for DVFS [1].

As energy reduction has become increasingly important, processors have become increasingly configurable. For example, some processors augment DVFS with a *sleep state*, allowing a system with no work to reduce power consumption [12]. For even a single core system with a low-power idle state and DVFS, optimal energy scheduling is intractable, but approximation algorithms have been developed that are within a bounded distance from optimal [2].

Despite this difficulty, embedded systems now include not only DVFS and idle states, but also multiple processing cores and even heterogeneous multi-cores, each with different capabilities [13, 14]. Even for a single application, scheduling for multiple cores, core types, DVFS settings, and idle states is challenging, but worth pursuing. Empirical evidence has repeatedly shown that configuring across multiple resources provides greater energy efficiency than working with only a single component[7, 9, 10, 17, 18].

Case studies done in the early and middle part of the 2000s found that the complexity of theoretically optimal schedulers provided little to no benefit in practice [3, 16, 19, 21]. Instead, these studies indicated the well-known *race-to-idle* heuristic, which allocates all resources to complete a task and then idles until the next task is ready, is often close to optimal. More recent studies, however, suggest that this trend is starting to reverse; they call into question the efficacy of the race-to-idle heuristic [5, 8, 15]. 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.

In this paper, we express the problem of allocating resources in a configurable system as a linear optimization which minimizes energy subject to a performance constraint. We then consider two common heuristic solutions to the problem and compare their performance on two different embedded systems. Our results indicate that the effectiveness of the heuristics is entirely system dependent. Therefore, a generalized approach that works across a range of systems will need to be more sophisticated than either heuristic alone.

3. ENERGY OPTIMIZATION

Our goal is to generalize the problem of allocating system resources to complete a task by a deadline while minimizing energy. We assume a task consists of some amount of *work* to be accomplished. We assume that the system is configurable – a *configuration* represents an assignment of resources to a task. In a heterogeneous system, for example, a configuration could represent the assignment of a core type and clockspeed. As more resources can be assigned to a task, the number of possible configurations increases. Each configuration produces a different performance and power consumption. We assume that tasks can be assigned time with different system configurations. The energy optimization problem, then, is to assign a time to each possible configuration such that the total work accomplished is equal to the task's work and the total energy consumption is minimized. Some configurations may be assigned zero time, meaning the task will not make use of them.

3.1 Problem Formulation

We build on prior work that expressed this problem as a linear program [8]. We assume that a task starts at time 0, has a workload of W work units, and a deadline at time t. The system has C configurations such that $c \in \{0, \ldots, C-1\}$. Each configuration has a *performance* (work rate) r_c and a *power* consumption p_c. By convention, we denote $c = 0$ to be the system *idle* state with $r_0 = 0$ and $p_0 = p_{idle}$. This idle power consumption is a constant property of the system. Also by convention, we denote $c = C-1$ as the configuration that allocates all resources to the task. For example, in a two core system with four different clock speeds, there are nine configurations ($2cores \times 4speeds + idle$), and $c = 8$ is the assignment of all cores at the highest speed.

We schedule time t_c in each configuration where $0 \leq t_c \leq t, \forall c$. Each configuration contributes $t_c \cdot r_c$ work and consumes $t_c \cdot p_c$ energy. Energy minimization under a performance constraint can then be formulated as the following linear program:

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_c t_c \cdot p_c & (1) \\ \text{subject to} \quad & & \\ & \textstyle\sum_c t_c \cdot r_c = W & (2) \\ & \textstyle\sum_c t_c = t & (3) \\ & t \geq t_c \geq 0, \quad \text{for } c = 0, \ldots, C-1 & (4) \end{aligned}$$

Equation 1 is the objective function representing total energy consumption. Equation 2 is the constraint that all work is completed, while Equations 3 and 4 ensure the deadline is respected.

This is an entirely general formulation of the problem. It is applicable to any configurable system, deadline, and workload. The issue is that as system complexity (*i.e.,* the number of configurable components) grows, the number of configurations increases exponentially. Thus, this problem is often difficult to solve in practice. The great irony is that solving this problem would reduce energy consumption, but embedded systems are often too resource constrained to tackle such a difficult problem in the first place.

3.2 Heuristic Scheduling Strategies

Given the difficulty of solving this problem in practice, we consider heuristic solutions. Interestingly, several well-known heuristics map directly onto the structure of this optimization problem. These heuristic solutions meet the constraints – they complete the work by the deadline – but may consume more energy than the true optimal solution. Specifically, this paper considers two heuristics:

- **race-to-idle:** Also known as *race-to-complete* or *race-to-halt*, this heuristic makes all resources available (*i.e.,* schedules all time in configuration $c = C - 1$) until the task completes and then idles (*i.e.,* schedules remaining time in $c = 0$) until the next task arrives. Formally, the time spent in each configuration is:

$$t_{C-1} = \frac{W}{r_{C-1}} \tag{5}$$

$$t_{idle} = t - t_{C-1} \tag{6}$$

$$t_c = 0, \forall c \neq C-1, idle \tag{7}$$

- **never-idle:** This heuristic attempts to keep the system busy (but perhaps not fully utilized) and complete the work just at the deadline. This strategy schedules time in two configurations: the lowest power state that is just faster than necessary, hi, and the most efficient state that is slower than necessary, lo:

$$hi = \arg\min_{k \in C}\{p_k | r_k \geq W/t\} \tag{8}$$

$$lo = \arg\max_{k \in C}\{r_k/p_k | r_k < W/t\} \tag{9}$$

The time spent in each state is then:

$$t_{hi} = \frac{W - r_{lo} \cdot t}{r_{hi} - r_{lo}} \tag{10}$$

$$t_{lo} = \frac{r_{hi} \cdot t - W}{r_{hi} - r_{lo}} \tag{11}$$

$$t_c = 0, \forall c \neq hi, lo \tag{12}$$

These two strategies place different burdens on implementers. *Race-to-idle* is exceptionally easy to implement. It simply allocates all resources and then transitions to the idle state when the work completes as defined in Equations 5 and 6. Implementing race-to-idle requires no insight into the performance or power delivered by the system – it only requires recognizing when the task has finished. This heuristic is also portable – the same strategy can be easily implemented on any system. In contrast, the *never-idle* heuristic requires more insight into the application's performance and system power in different configurations. Specifically, it requires selecting the hi and lo states and it may be harder to port from system to system as the interfaces to different configurations may change. In addition, some configurable components may not be available on different systems.

The next section evaluates these heuristics.

4. EMPIRICAL EVALUATION

In this section we describe two embedded platforms and our approach to characterizing their properties. We then analyze the results and discover that the two platforms have contrasting performance/energy tradeoff spaces that require different heuristics to achieve optimal energy efficiency.

4.1 Evaluation Platforms

Two different embedded platforms are evaluated - one with homogeneous multi-cores and the other with single-ISA, heterogeneous multi-cores. The homogeneous system is a Sony VAIO SVT11226CXB Tablet PC with a dual core Intel Haswell processor with hyperthreading that supports eleven DVFS settings ranging from 600 MHz to 1.501 GHz. The heterogeneous system is an ODROID-XU+E [6], an ARM big.LITTLE development platform with the Samsung Exynos5 Octa SoC containing Cortex-A15 and Cortex-A7 quad core CPUs. The Cortex-A15, the big cluster, supports nine DVFS settings ranging from 800 MHz to 1.6 GHz, while the Cortex-A7, the LITTLE cluster, supports eight DVFS settings ranging from 500 MHz to 1.2 GHz. Both devices run Ubuntu Linux 14.04 LTS, with the Vaio using Linux kernel 3.13.0 and the ODROID using Linux kernel 3.4.91.

Table 1: System power characteristics.

System	Idle Power	Min Power	Max Power
Vaio	2.50 W	3.04 W	12.20 W
ODROID	0.12 W	0.17 W	10.16 W

Table 2: System configurations.

Vaio		
Configuration	Settings	Max Speedup
clock speed	11	2.72
cores	2	1.81
hyperthreads	2	1.10

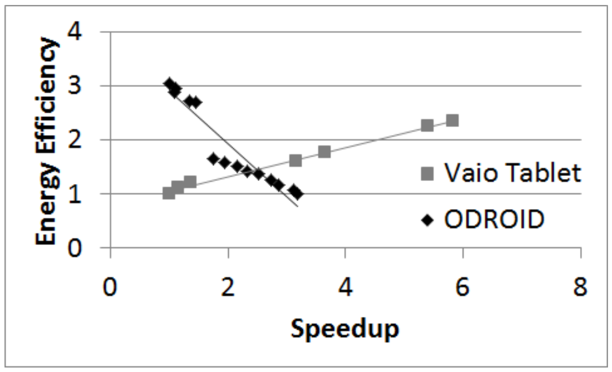
ODROID		
Configuration	Settings	Max Speedup
big cores	4	6.10
big core speeds	9	1.97
LITTLE cores	4	3.94
LITTLE core speeds	8	2.40

To measure power on the Vaio we use the Haswell processor's Model-Specific Register (MSR) which tracks energy consumption. The ODROID has four integrated sensors [11] that we use to monitor the A15 cluster, the A7 cluster, the DRAM, and the GPU. Table 1 presents the power characteristics captured from both devices, including the idle power as well as the minimum and maximum power usage during execution. The cpufrequtils interface is used to control the DVFS settings on each device.

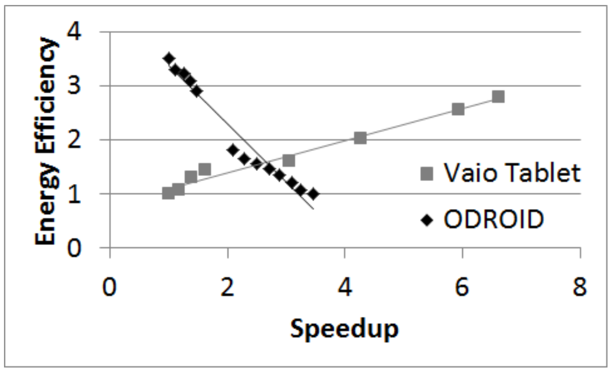
We use four benchmark applications from PARSEC [4] - bodytrack, ferret, swaptions, and x264. These benchmarks represent a variety of applications with performance requirements that one can reasonably expect to use on an embedded system. Both bodytrack and x264 process video data and could be required to match the performance of an on-board camera. Ferret is a content-based search engine for non-text data types. Swaptions is a financial pricing application that is often run with performance and energy constraints to ensure that the prices it outputs are timely while minimizing the electricity cost of producing them.

4.2 Methodology

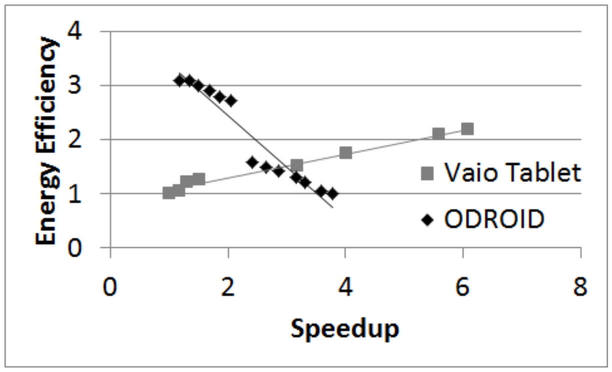
For these experiments, a *configuration* is a unique combination of system components and their settings. Table 2 presents the configuration parameters for each system. In total there are 44 configurations on the Vaio and 68 on the ODROID. On both platforms a clockspeed setting is applied to all active cores concurrently, meaning there are no configurations that include different clockspeeds on different cores. The reason there are not more configurations on the ODROID is because the system does not support executing on



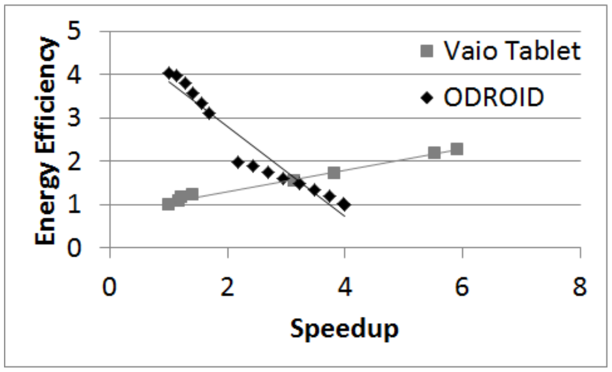
(a) bodytrack



(b) ferret



(c) swaptions



(d) x264

Figure 1: Speedup and normalized energy efficiency for different benchmarks on the Vaio and ODROID.

the big and LITTLE clusters simultaneously.

To collect performance and power metrics, we modify the Heartbeats API [10] to capture energy readings in addition to performance statistics. Each benchmark is then edited to produce heartbeats at intervals appropriate for their respective tasks. The modified Heartbeats library captures energy readings from the MSR on the Vaio and uses the ODROID's embedded power sensors to calculate energy consumption. The fact that one device records energy while the other offers power measurements already demonstrates some complexity in designing a general-purpose solution for meeting performance targets with power/energy constraints. Indeed, this additional complexity compelled us to abstract the component that reads (or calculates) energy from the rest of the Heartbeats library in order to build a more generic system capable of capturing these metrics on diverse platforms.

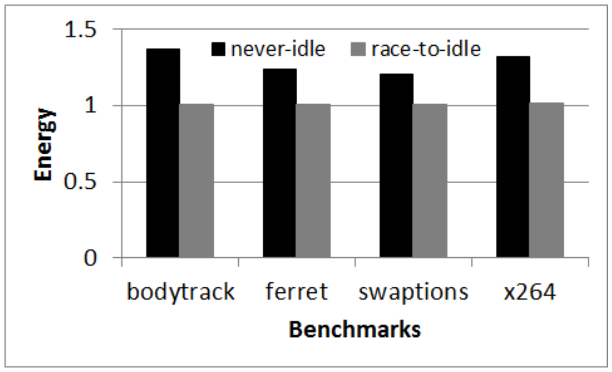
By varying the settings in Table 2, we execute the four benchmarks in all possible configurations on each system. For each benchmark the number of worker threads is fixed to match the maximum number of virtual cores (four in both cases). This choice is predicated on the assumption that the software either has a predetermined number of threads to match the platform it is designed for, or the software is able to determine the hardware configuration prior to performing its main task. In each execution, the Heartbeats library records the performance and power.

4.3 Performance/Energy Tradeoff Spaces

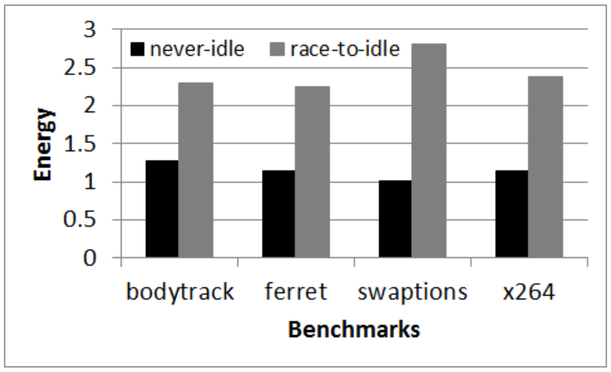
As mentioned previously, the four benchmarks are executed in all configurations on each device, and we measure power and performance. Our goal is to understand the relationship between energy efficiency and performance for each device.

In Figure 1 we plot the performance vs. energy efficiency curves for the two devices on the same chart for each benchmark. To make the charts readable, any state that delivers less performance for the same energy efficiency is discarded (so there are fewer points than configurations in the figures). The X-axis (Speedup) describes the performance improvement for each state, normalized to the lowest performing state along each curve. Using the x264 benchmark as an example, the amount of *work* to be done is the number of frames to process and the *performance* (work rate) is measured in frames per second (FPS). The Y-axis (Energy Efficiency) is defined as performance/power and is normalized to the least energy-efficient state for each curve. In the x264 benchmark, energy efficiency is categorized as the number of frames processed per Joule of energy ($\frac{FPS}{Watts} = \frac{frames}{Joule}$). For both axes, larger values are better.

In each chart of Figure 1 the Vaio's energy efficiency increases over the baseline along with its performance. In contrast, the ODROID's energy efficiency drops as it moves to higher performing states. The area without points approximately halfway through the ODROID curve on each plot is a result of the big.LITTLE architecture and the disconnect in performance and energy efficiency between the ODROID's smaller and larger cores. Clearly, these devices have completely different performance and energy efficiency



(a) Vaio



(b) ODROID

Figure 2: Never-idle and race-to-idle heuristics for the Vaio and ODROID with a performance target of 50% of their respective maximums. The Y-axis is normalized to optimal.

tradeoffs, so each will need a different resource allocation strategy to minimize energy.

The next section demonstrates how much energy can be saved by picking the right strategy.

4.4 Energy Optimization Using Heuristics

Using the prior results we derive an oracle that, given a performance target, calculates the minimum amount of energy required to complete each benchmark on each system. Energy consumption is then modeled for the never-idle and race-to-idle heuristics on both platforms for each of the benchmarks. Figure 2 presents the results for a performance target of 50% of each device's maximum performance on each benchmark. Energy is normalized to the oracle's minimum energy consumption for each benchmark and platform. Lower values are better.

Figure 2a indicates that the race-to-idle heuristic is near-optimal for the Vaio tablet. Figure 2b shows that the ODROID is more energy efficient using the never-idle heuristic than race-to-idle. Therefore, the ODROID requires a different approach to completing these tasks on time. These results follow from the tradeoffs spaces in Figure 1. On the Vaio, speeding up increases energy efficiency, so it is better to complete work faster using race-to-idle. On the ODROID, slowing down increases energy efficiency, so it is better to complete the work just at the deadline. This relationship holds despite the fact the ODROID has much lower idle power.

The effects of choosing the right strategy are dramatic. On the Vaio, race-to-idle is only slightly worse than optimal for all benchmarks., but the never-idle strategy consumes 28% more energy than optimal on average. On the ODROID, never-idle consumes 14% more energy than optimal, but race-to-idle increases energy consumption by 2.43× on average. These results indicate that choosing the wrong strategy is not a trivial issue, but severely impacts energy consumptions.

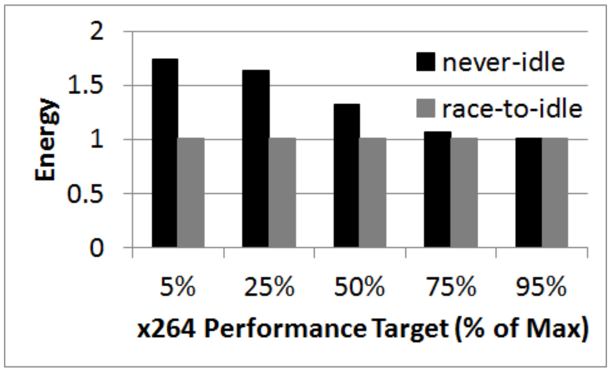
4.5 Sensitivity to Performance Target

In the previous section we set a performance target of 50% of each system's performance capacity. This section evaluates each heuristic's sensitivity to this target.

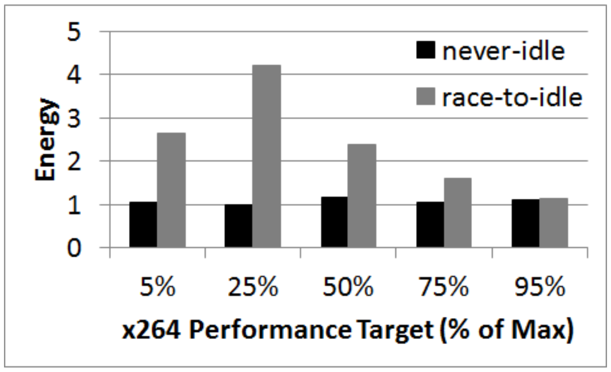
For simplicity, we begin by considering two simple theoretical systems. All else being equal, the energy efficiency of the first system scales proportionally to its resource consumption, and the energy efficiency of the second scales inversely to its resource consumption. Both of these theoretical systems obviously require different strategies for minimizing energy usage while meeting performance targets. For maximum energy efficiency, the first system will want to use its relatively efficient high-power states as long as possible, so it benefits from using race-to-idle. Conversely, the second system prefers to use its relatively efficient low-power states as long as possible, so it benefits from using never-idle.

For both theoretical systems, as the performance target increases toward capacity and a race-to-idle heuristic is used, more time is required in high-power execution states to complete the task. Less time is then spent in the low-power idle state. This is already known to be optimal for the first system. The second system, however, now spends less time in its relatively energy-inefficient low-power idle state, and a higher energy efficiency is achieved than at lower performance targets where more time is spent idling. Likewise, if a never-idle heuristic is used on both systems, a larger percentage of each system's resources are required in order to meet the performance target. The gap between the utilized resources and total available resources closes, resulting in fewer resources being unused. This is already known to be optimal for the second system. Now the first system uses its relatively energy-efficient high-power states and thus achieves a higher energy efficiency than at lower performance targets where less energy-efficient execution states are used. For both race-to-idle and never-idle heuristics, there is less opportunity to save energy at higher performance targets. We then conclude that the heuristic strategy becomes less crucial as the performance target increases toward system capacity.

To verify this conclusion, we model the energy usage for a variety of performance targets across the range of attainable targets for the Vaio and ODROID, each of which we believe exhibit similar properties to one of the two theoretical systems. Using the x264 benchmark as an example, Figure 3 presents the energy used by each heuristic relative to the oracle for performance targets ranging from 5% to 95% of each system's performance capacity (lower values are better). It is obvious again which heuristic is better for each system -



(a) Vaio



(b) ODROID

Figure 3: Normalized energy for various performance targets using the x264 benchmark.

the Vaio benefits from race-to-idle and the ODROID from never-idle. We see for both systems that the lower performance targets suffer the most excess energy consumption over the baseline when an inefficient heuristic is used. As the performance target approaches capacity, the two heuristics converge at the optimal energy usage for that target, as anticipated.

Results for the other three benchmarks follow the same trends, but are omitted for space. This sensitivity analysis demonstrates that our conclusions hold across a range of performance targets. Picking the right strategy is essential for any system except those that are always fully utilized.

5. CONCLUSIONS

This paper evaluates two embedded platforms that demonstrate a diversity in performance and energy efficiency tradeoff spaces. One platform improves in energy efficiency as it moves to higher performance states while the other experiences a reduction in energy efficiency as its performance increases. Applying a race-to-idle heuristic to the first achieves near-optimal energy consumption, but a never-idle heuristic achieves comparable results on the other. Different resource allocation strategies are clearly necessary for platforms exhibiting these differing characteristics in order to attain optimal energy efficiency while meeting performance targets. These results indicate that a general-purpose solution to this problem must be able to support inherently different performance/energy tradeoff spaces.

It is also reasonable to expect that, even on a single platform, applications with different processing requirements may have different performance/energy tradeoff spaces. Therefore, a single heuristic may not be near-optimal for all applications on that platform. In such a scenario, a general-purpose solution must also support different behavior for various applications. Future work will identify applications and systems that exhibit behavior different than those presented here. We will then evaluate whether new behaviors require new heuristics or whether the heuristics are robust for a given platform.

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] L. A. Barroso and U. Hölzle. “The Case for Energy-Proportional Computing”. In: *IEEE Computer* 40 (2007).

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

[5] 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.

[6] HardKernel. `http://www.hardkernel.com/main/products/prdt\_info.php?g\_code=G137463363079`.

[7] 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.

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

[9] H. Hoffmann et al. “Self-aware computing in the Angstrom processor”. In: *DAC*. 2012.

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

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

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

[13] B. Jeff. “Big.LITTLE system architecture from ARM: saving power through heterogeneous multiprocessing and task context migration”. In: *DAC*. 2012, pp. 1143–1146.

[14] R. Kumar et al. “Processor Power Reduction Via Single-ISA Heterogeneous Multi-Core Architectures”. In: *Computer Architecture Letters* 2.1 (2003), pp. 2–2. ISSN: 1556-6056. DOI: `10.1109/L-CA.2003.6`.

[15] 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.

[16] 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.

[17] 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).

[18] D. Meisner et al. “Power management of online data-intensive services”. In: *ISCA* (2011).

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

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

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

[22] M. Weiser et al. “Scheduling for reduced CPU energy”. In: *Mobile Computing* (1996).