Racing and Pacing to Idle

An Evaluation of Heuristics for Energy-aware Resource Allocation

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

ABSTRACT

We examine the problem of assigning computing resources to an application to meet a performance goal while minimizing energy consumption. We present a general formulation of this problem as a linear program, discuss several potential heuristic solutions, and evaluate these heuristics on two real systems (one purchased in 2010, the other in 2013). We find that the well-known race-to-idle heuristic is close to the optimal solution on the older machine. On the newer machine, however, the optimal solution outperforms race-to-idle by over 35%. A generalization of race-to-idle, called pace-to-idle, is found to provide better results in a wider range of scenarios.

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

This work was funded by the U.S. Government under the DARPA PERFECT program. 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. It is approved for public release; distribution is unlimited.

HotPower '13, November 03 - 06 2013, Farmington, PA, USA

http://dx.doi.org/10.1145/2525526.2525854

1. INTRODUCTION

There is a large class of computing applications which interact with the outside world (*e.g.,* sensors, humans, other computers). The speed of this interaction defines a performance requirement (*i.e.,* real-time deadline, quality-of-service goal, or service-level agreement), placing restrictions on the time to accomplish a given task.

Most modern computing systems have components that expose tradeoffs between their performance and power consumption. Energy-aware resource allocators support applications with performance bounds by configuring component usage to meet the performance while minimizing energy consumption. This problem can be formulated as a general optimization problem of scheduling time spent in each configuration to ensure the work is completed on time while energy is minimized.

Remarkably, a simple heuristic solution, commonly known as *race-to-idle*, has consistently produced near-optimal results for this problem across a range of applications and systems. The race-to-idle solution completes the work as fast as possible and then idles the system until the next work item becomes available. This solution takes advantage of a lack of *energy proportionality*; *i.e.,* computing systems are traditionally most energy efficient in their highest performance configuration and their efficiency declines with their utilization [8]. Taking advantage of this inefficiency, energy-aware allocators do not solve optimization problems; instead, they simply allocate maximum resources and then idle the system when the task is complete.

However, there is great motivation to create energy proportional systems, but as such systems emerge, race-to-idle will no longer be a good approximation of the true optimal schedule. This paper examines the increasing complexity of energy-aware resource allocation for applications with performance goals. We formulate a linear optimization problem for meeting deadlines with minimal energy (Section 3). We consider several heuristic solutions to this problem including *pace-to-idle*, which completes the task in the most energy efficient configuration and then idles. We then evaluate the performance of these heuristics for several benchmarks on two real servers, one based on an Intel Xeon E5530 (purchased in 2010) and the other based on an Intel Xeon E5-2690 (purchased in 2013) (Section 4).

The results indicate a major difference in the effectiveness of the race-to-idle heuristic depending on the age of the machine. On the older machine race-to-idle is close to optimal for all tested benchmarks. On the newer machine, however, the optimal solution is over 30% better than racing to idle.

The pace-to-idle solution is better across machines, producing essentially optimal efficiency on the older machine and a 20% gain, compared to racing, on the newer machine. Interestingly, on the new machine, the optimal solutions never make use of the most energy efficient configuration; *i.e.,* the most energy efficient schedule for this machine does not use the single most energy efficient configuration available. We discuss the implications of these results for constructing resource allocators.

2. BACKGROUND

Most computer systems have multiple components that support different power and performance tradeoffs. Researchers have proposed many techniques for allocating these components, including tuning the duty cycle [7, 21], the DVFS setting [17, 19], the number of active cores [13], and the caches [1]. Additional techniques manage components outside the CPU; *e.g.,* DRAM [5, 22] and the disk [11].

It has been noted that coordinated allocation of multiple components should be more efficient than management of components in isolation [8, 9, 16]. Thus, researchers have built coordinated allocation schemes to achieve these efficiency gains (*e.g.,* managing cores and DVFS [4, 14], processor and DRAM [3, 6, 12], thread scheduling and DVFS [20]). Additional work has developed generalized techniques for managing arbitrary sets of resources [10].

Despite differences in mechanisms, these techniques all solve a common problem: select the best configuration for meeting a given goal. We examine a general formulation of this problem where the best solution is the one that meets a performance goal with minimal energy consumption.

3. THE OPTIMIZATION PROBLEM

This section formalizes the constrained optimization problem of completing a task by a deadline while minimizing energy consumption. This formulation is broadly applicable to many such tasks on systems with differing configurations. After presenting the optimization problem, we consider several heuristic solutions.

3.1 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 r_c (work units per time) and a power consumption of p_c. There is a unique *idle* configuration $c = 0$ with $r_0 = 0$ and $p_0 = p_{idle}$, where p_{idle} is a system dependent (and application independent) value representing the system's power consumption when it is not executing computation. We assume that configuration $C-1$ represents allocating the maximum resources of the system. The goal is to assign a time t_c, where t_c $(0 \leq t_c \leq t)$, to each machine configuration. Note that configuration c will contribute $t_c \cdot r_c$ work at a cost of $p_c \cdot t_c$ energy. Thus, the problem of minimizing energy while completing the work can be expressed as:

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

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 all configurations is non-negative.

3.2 Heuristic Solutions

Ideally we would find the true optimal solution (referred to as **best** throughout the paper), but this may be computationally infeasible in practice [20]. So, we are also interested in four heuristic solutions. These approximations meet the constraints of Equations 2–4 (*i.e.,* ensure the performance goal is met) without guaranteeing minimal energy consumption (Equation 1). The heuristics are:

- **max**: Provide all resources until the task completes and then 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.
- **race**: Run in the fastest configuration, *race*, until the task is complete and then idle. The fastest configuration is the one that produces the highest computation rate r_c; *i.e., race* $\arg\max_c r_c$. This formulation represents the well known race-to-idle heuristic. Applying this heuristic requires finding the *race* configuration.
- **pace**: Run in the most energy efficient configuration, *pace*, until the task is complete and then idle. The energy efficiency e_c of configuration c is $e_c = r_c/p_c$. Therefore, the most energy efficient configuration is the one with the largest e_c; *i.e., pace* $= \arg\max_c r_c/p_c$. This formulation represents the pace-to-idle heuristic. Applying this heuristic requires finding the *pace* configuration.
- **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 $hi = \arg\min_{c \in \{k \in C | r_k \geq W/t\}} p_c$ and $lo = \arg\max_{c \in \{k \in C | r_k < W/t\}} r_c/p_c$. This heuristic requires finding the *hi* and *lo* configurations.

We note several properties of these solutions:

- Pace is a generalization of race; when the highest performance state is also the most energy efficient, they are equivalent.
- On systems where race-to-idle has been successful, the max, race, and pace heuristics are all equivalent and close (or equal) to the best solution.
- If a system idles with no power consumption, then pace is equivalent to the best solution. The energy consumption of pace is equal to the time spent in *pace* times its power consumption (p_{pace}) plus the time spent in idle times the idle power consumption, which is zero in this case. The average energy efficiency is r_{pace}/p_{pace}. The only way to improve the total energy consumption is if there is another configuration that would improve the average energy efficiency. The existence of such a state, however, contradicts our definition of *pace* as the most energy efficient state, meaning that the pace heuristic is equivalent to the best solution when the idle power is negligible.
- On systems with non-zero idle power, the best solution may not use the most energy efficient configuration

if a higher average energy efficiency can be achieved without idling. This property follows the same logic as above, but for non-zero idle power. In this case, idling consumes energy without producing work, so it may be possible to raise the total energy efficiency using two configurations that are individually less energy efficient than *pace*.

- Solutions are application dependent and may be input dependent if different inputs trigger different paths of control flow. That is, the highest performance, or most energy efficient configuration for one application may not be the best for another application. Likewise, it is possible that the fastest or most energy efficient configuration is input dependent.

4. CASE STUDY

This section evaluates the best (or true optimal solution) and compares it to the heuristic solutions on two separate systems. The key difference between the two systems is that the processors in the newer one come closer to achieving energy proportionality. The result is that the race-to-idle heuristic, which works well on the older machine, performs poorly on the newer machine.

4.1 Evaluation Platforms

We use two dual-socket Intel/Linux systems. The first, purchased in 2010, has two Xeon E5530 processors and runs Linux 2.6.26. Each processor supports four cores, seven DVFS settings (from 1.6 – 2.4 GHz), hyper-threading, and TurboBoost. The second, purchased in 2013, has two Xeon E5-2690 processors and runs Linux 3.2.0. These processors have eight cores, sixteen DVFS settings (from 1.2 – 2.9 GHz), hyper-threading, and TurboBoost.

Both systems are connected to WattsUp power meters that measure the full system power. The older machine has an idle power of approximately 90 Watts. The lowest measured active power is approximately 130 Watts and the highest is 220 Watts. The newer machine has an idle power of approximately 75 Watts. Its lowest active power is just over 95 Watts while its highest is 365 Watts.

We use four benchmark applications, x264, bodytrack, and swaptions from PARSEC [2], plus the swish++ search engine [18]. These benchmarks are broadly representative of applications with performance requirements. x264 and bodytrack process video data and may be required to keep pace with a camera. swaptions prices financial instruments and latency is critical for high-frequency trading. Finally, the latency of swish++'s search requests is an essential metric of its success.

4.2 Methodology

We run each benchmark on both test systems in every configuration c, recording performance r_c and power consumption p_c. We then assign a real-time deadline (equal to twice the time it would take to complete in the fastest configuration) to each application and solve Equations 1–4 to determine the optimal energy efficiency for that application on each machine. Finally, we compute the energy efficiency for the best solution and for each heuristic solution described in Section 3.

4.3 General Results

Figure 1 shows results for the older machine in Figure 1a and for the new one in Figure 1b. Both charts show the benchmarks on the x-axis and energy efficiency (normalized to the race heuristic) on the y-axis. For each benchmark, a bar shows the achieved energy efficiency for each of max, race, pace, no-idle, and the best (true optimal) solution.

The results in Figure 1a confirm the efficacy of race-to-idle for the older machine. On average, best (the true optimal solution) provides an improvement of less than 2% compared to race. No-idle is the worst, achieving only 98% of optimal. Notably, the max, race, and pace solutions all use the same configuration for all benchmarks for this machine. This system does not provide energy proportionality, but it is extremely easy to allocate resources efficiently — simply provide all resources to any task and then idle.

Figure 1b, however, shows that the newer machine requires a different resource allocation strategy. Here, the optimal solution (best) is over 35% more energy efficient than race, while the no-idle heuristic exceeds race by about 32% on average and pace also outperforms race by 24%. Finally, the worst strategy is max (allocating all resources), which reduces energy efficiency by about 3% compared to race.

The difficulty of applying these solutions increases dramatically going from the older machine to the newer one. On the older machine, race and pace are application independent; they simply allocate all resources to the application. On the newer machine, however, applying race or pace requires finding the fastest (for race) or most energy efficient (for pace) configurations. This process is difficult because they are application dependent. For example, across the four benchmarks three distinct configurations are used to race and four distinct configurations are used for pace. On this machine, the most energy efficient state varies per application. The problem is even worse for the no-idle and best strategies which use more configurations. These results imply that, on older machines, energy-aware resource allocators could be application independent, but on newer machines, they need to tune their behavior to particular applications.

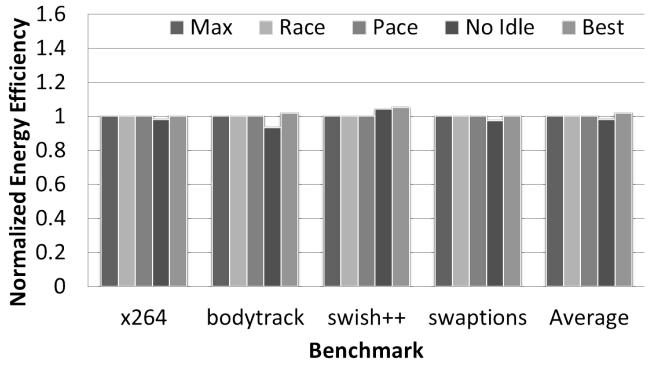
4.4 Video Encoding Results

We follow the same methodology as above, running the x264 application on every configuration of the two machines. In this case, though, we use six distinct video inputs and record both the performance and power for each. These videos are all 1080p resolution high-definition videos downloaded from `xiph.org`. Different videos cause the encoder to execute different paths of control flow. Therefore, the most energy efficient configuration for one video might not be the same for another.

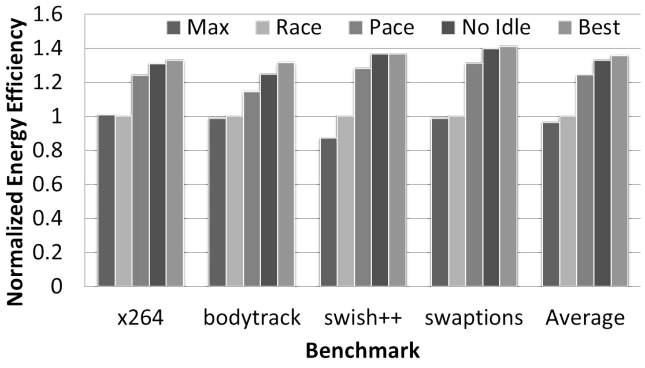
Figure 2 shows the results; Figure 2a shows the data for the older machine, while Figure 2b shows the data for the newer machine. These results follow the trends noted above. The max, race, pace, and best solutions are almost indistinguishable on the older system. On the newer system, the best solution is over 30% more energy efficient than race, pace is over 20% better than race, and the no-idle solution is the best of the heuristics.

4.4.1 Input Sensitivity

For the six different video inputs, the pace-to-idle solution uses five separate configurations. For example, the most energy efficient state for the `ducks_take_off_1080p` video uses

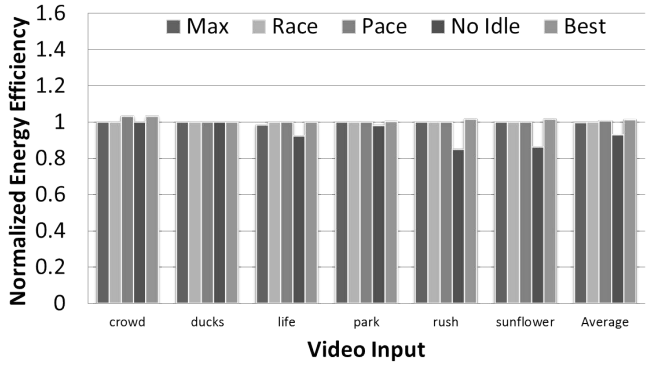


(a) Intel Xeon E5530

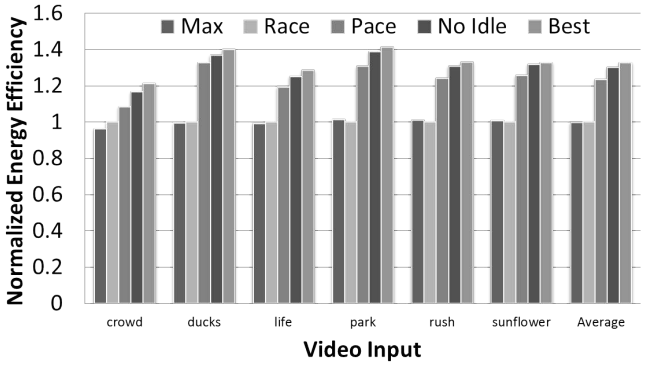


(b) Intel Xeon E5-2690

Figure 1: Energy efficiencies for different benchmarks and resource allocation strategies on different machines.



(a) Intel Xeon E5530



(b) Intel Xeon E5-2690

Figure 2: Energy efficiencies for different video inputs and resource allocation strategies on different machines.

Table 1: Potential energy efficiency losses when using the most energy efficient state for one video to encode another.

Video	Loss (%)	Video	Loss (%)
`crowd_run_1080p`	6	`park_joy_1080p`	19
`ducks_take_off_1080p`	20	`rush_hour`	7
`life`	8	`sunflower`	11
Average	11.8		

14 cores at 2.8 GHz, and turns off hyper-threading. For the `life` video, the most efficient configuration is 16 cores at 2.4 GHz, also without hyper-threading. These differ slightly in their resource usage, but significantly in their energy efficiency. For example, using the most energy efficient configuration for `life` to encode `ducks_take_off_1080p`, energy efficiency drops by almost 20%.

Table 1 summarizes the penalties for using the most energy efficient state for one video to encode a different video. For each input the table shows the loss of energy efficiency for using the best configuration from another video. These results indicate that an energy-aware allocator needs to be responsive, not just to different applications, but even different inputs for a given application.

4.5 Sensitivity to Other Parameters

The solution to the energy-aware allocation problem is sensitive to both application and input. There are two other key parameters which affect the results significantly: idle power and performance goal.

4.5.1 *Idle Power*

As the idle power decreases, the best solution and the pace solution converge. There is a key crossover point where no-idle becomes worse than pace. Unfortunately, this point is difficult to find analytically because it is a function of all available configurations, the deadline, the amount of work to be done and the idle power.

Even at low idle power, however, we find that the pace-to-idle strategy outperforms race-to-idle on the E5-2690 system. We demonstrate this by performing the same experiment as above but replacing the true idle power by reporting zero idle power. These results are shown in Figure 3. In this case, the pace strategy is, as expected, equivalent to the optimal solution, and both out perform the race-to-idle strategy by over 20%. Also, the no-idle heuristic is by far the worst.

4.5.2 *Performance Goal*

The deadline and the amount of work to be performed

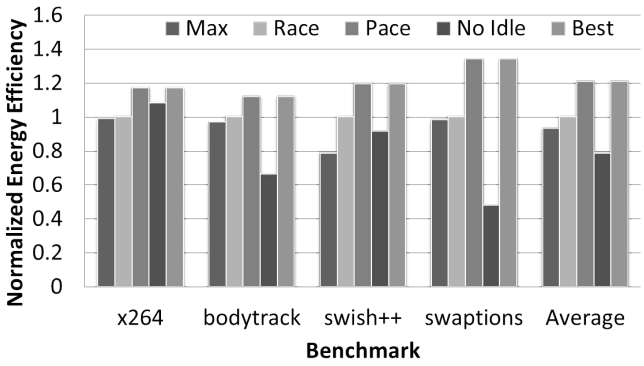


Figure 3: Energy efficiencies for zero idle power.

define a performance requirement. As the performance requirement approaches the capacity of the machine, then all strategies become equivalent, as they must all use the highest performance configuration. Similarly, as the performance requirement approaches zero, then all strategies become equivalent as idle power consumption dominates the power spent performing computation. Thus, the results presented here, which target a performance in between the minimum and maximum, represent the case where the different strategies will have the most notable effects.

5. CONCLUSIONS

This paper considers the problem of minimizing energy consumption for applications with performance requirements. We characterize this as a general linear optimization problem and consider several heuristic solutions (including race-to-idle, pace-to-idle, and no-idle). We evaluate these heuristics on two real systems for several representative benchmarks. The results indicate three trends that will be important to take into account when designing future energy-aware resource allocators:

- The race-to-idle heuristic is close to optimal on machines that are not energy proportional, but it becomes less effective as systems support richer energy/performance tradeoffs. On a current commercial processor, the optimal allocation is 35% better than race-to-idle.
- No single heuristic is uniformly best, but the pace-to-idle heuristic works well in several situations. On older machines without energy proportionality, pace-to-idle is equivalent to race-to-idle. On newer machines, pace-to-idle provides over 20% better energy efficiency than race-to-idle. On potential future machines with no, or extremely low [15], idle power, pace-to-idle is equivalent to optimal.
- The exact configuration to be used in the pace-to-idle strategy is both application and input dependent. These dependencies create challenges for energy-aware allocation, which must adapt to the needs of the current input.

6. REFERENCES

[1] R. Balasubramonian et al. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *MICRO*, 2000.

[2] C. Bienia et al. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, Oct 2008.

[3] J. Chen and L. K. John. Predictive coordination of multiple on-chip resources for chip multiprocessors. In *ICS*, 2011.

[4] R. Cochran et al. Pack & cap: adaptive dvfs and thread packing under power caps. In *MICRO*, 2011.

[5] B. Diniz et al. Limiting the power consumption of main memory. In *ISCA*, 2007.

[6] W. Felter et al. A performance-conserving approach for reducing peak power consumption in server systems. In *ICS*, 2005.

[7] A. Gandhi et al. Power capping via forced idleness. In *Workshop on Energy-Efficient Design*, June 2009.

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

[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] X. Li et al. Performance directed energy management for main memory and disks. *Trans. Storage*, 1(3):346–380, Aug. 2005.

[12] X. Li et al. Cross-component energy management: Joint adaptation of processor and memory. *ACM Trans. Archit. Code Optim.*, 4(3), Sept. 2007.

[13] M. Maggio et al. Controlling software applications via resource allocation within the heartbeats framework. In *CDC*, 2010.

[14] M. Maggio et al. Power optimization in embedded systems via feedback control of resource allocation. *IEEE Transactions On Control Systems Technology*, 21(1):239–246, 2013.

[15] D. Meisner et al. Powernap: eliminating server idle power. 2009.

[16] D. Meisner et al. Power management of online data-intensive services. *ISCA*, 2011.

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

[18] SWISH++. `http://swishplusplus.sourceforge.net/`.

[19] M. Weiser et al. Scheduling for reduced CPU energy. *Mobile Computing*, 1996.

[20] J. A. Winter et al. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *PACT*, 2010.

[21] X. Zhang et al. A flexible framework for throttling-enabled multicore management (temm). In *ICPP*, 2012.

[22] H. Zheng et al. Mini-rank: Adaptive dram architecture for improving memory power efficiency. In *MICRO*, 2008.