Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics*

appeared in Euromicro Conf on Real-Time Systems, June 2001

Hakan Aydin, Rami Melhem, Daniel Mossé
Computer Science Department
University of Pittsburgh
Pittsburgh, PA 15260
aydin, melhem, mosse@cs.pitt.edu

Pedro Mejía-Alvarez†
CINVESTAV-IPN. Sección de Computación
Av. I.P.N. 2508, Zacatenco.
México, DF. 07300
pmejia@cs.cinvestav.mx

Abstract

In this paper, we provide an efficient solution for periodic real-time tasks with (potentially) different power consumption characteristics. We show that a task T_i can run at a constant speed S_i at *every* instance without hurting optimality. We sketch an $O(n^2 \log n)$ algorithm to compute the optimal S_i values. We also prove that the EDF (Earliest Deadline First) scheduling policy can be used to obtain a feasible schedule with these optimal speed values.

1 Introduction

With the advent of portable and hand-held computing/communication systems, power consumption has recently become a critical issue in system design. Extended battery life requirements in devices such as laptop computers, PCS telephones and other mobile embedded systems illustrate the need for efficient power-aware design and dynamic power monitoring techniques. Solar- and nuclear-powered systems, such as satellites and the MARS rover, also require serious power management considerations.

Nonetheless, in the last decade, the research community has addressed the low power system design problems with a multi-dimensional effort by introducing techniques involving VLSI/IC design, algorithm and compiler transformations, use of hierarchical memory systems and application specific modules, among others. For a detailed survey, the reader is referred to [16] and [7]. Such on-going research has important implications for real-time systems design, simply because most of the applications running on power-limited systems impose inherently temporal constraints on the response time (such as real-time communication and control tasks).

Hardware and software manufacturers have agreed to introduce standards such as the ACPI (Advanced Configuration and Power Interface) [11] for power management of laptop computers that allows several modes of operation, turning off some parts of the computer (e.g., the disk) after a preset period of inactivity. This simple predictive system shutdown technique [18] is usually used to turn off the power supply when the device is (and likely to stay) in idle mode for a considerable amount of time. However, due to the convex dependence between the supply voltage and the power consumption, this technique remains sub-optimal, even for a system with perfect knowledge of idle intervals. In other words, since a reduction in the supply voltage corresponds to a quadratic or cubic savings in energy [20, 10], the 0-1 voltage supply technique is clearly sub-optimal.

On the other hand, the *variable voltage scheduling* framework, which involves dynamically adjusting the voltage and frequency (hence the CPU speed), has recently become a major research area. This is perhaps the scheme used by Transmeta, in their new proprietary Crusoe processor: as far as we can gather, their approach takes advantage of past history on the rate of CPU activity to predict the future setting of the CPU speeds, for general purpose computing.

Processors operating at a range of voltages and frequencies are already available [10]. They are able to adjust their supply voltage, using a fine step, according to the required frequency. For such processors, the various voltages yield different execution delays, speeds,

*This work has been supported by the Defense Advanced Research Projects Agency through the PARTS (Power-Aware Real-Time Systems) project under Contract F33615-00-C-1736

†Work done while this author was visiting the Information Sciences and Telecommunications Department, University of Pittsburgh

and hence different energy consumptions. The ability to decrease the speed of the processor allows the system to reduce its power consumption.

In the realm of real-time systems, variable voltage scheduling focuses on minimizing energy consumption of the system, while still meeting the deadlines. The seminal work by Yao et. al [20] provided a static off-line scheduling algorithm, assuming aperiodic tasks and worst-case execution times. Heuristics for on-line scheduling of aperiodic tasks while not hurting the feasibility of off-line periodic requests are proposed in [9]. Non-preemptive power aware scheduling is investigated in [8]. Concentrating on a periodic task set with *identical periods*, the effects of having an upper bound on the voltage change rate are examined in [10], along with a heuristic to solve the problem.

Recent work in variable voltage scheduling includes the exploitation of idle intervals by slowing down the processor whenever there is a single task eligible for execution and its worst-case completion time is earlier than the first future arrival [17]. Although this *One Task Extension technique* was originally proposed in the context of Rate Monotonic Scheduling, it is easy to see that the idea can be applied to any periodic scheduling discipline. Cyclic and EDF scheduling of periodic hard real-time tasks on systems with two (discrete) voltage levels, including dynamic energy reclaiming heuristics have been investigated in [12].

1.1 Variable Voltage Scheduling Framework

An important component of power-aware real-time systems is to develop an efficient solution to the variable voltage scheduling. This is because, as mentioned above, simply committing to the maximum CPU speed and (predictively) shutting down the processor during idle intervals, despite its simplicity, yields sub-optimal results with respect to power consumption, in view of the convex power/speed relation. Since real-time systems are predictable (for example, advance knowledge of the worst case execution time of tasks is a necessary condition to guarantee feasibility), we can use such knowledge to produce a static (off-line) solution to compute the optimal speed for each task, assuming worst-case workload for each arrival.

Our target system is one in which the actual run-time of applications, under a given CPU speed, exhibits small to no variations and is most often equal to the worst-case execution time (WCET) of the task. Examples of these applications are often encountered in embedded satellite systems, such as image acquisition and image processing, in which the size of images is the determining factor of the WCET. Low-orbit satellites processing temporal data (e.g., audio packets) in multicast schemes are also excellent candidates for power-minimization while guaranteeing deadlines. Thus, with the help of a static assignment of the time/speed of each application, one can adjust (reduce) the speed of other tasks to take benefit of battery power. Clearly, care must be taken not to cause any deadlines to be missed.

1.2 Task-level Energy Minimization

The power consumption of an on-chip system is a strictly increasing *convex* function of the supply voltage V_{dd}, but its exact form depends on the technology[10]. For example, the dominant component of energy consumption in widely popular CMOS technology is the *dynamic power dissipation* P_d, which is given by:

$$P_d = C_{ef} \cdot V_{dd}^2 \cdot f \tag{1}$$

where C_{ef} is the *effective switched capacitance* and f is the frequency of the clock. The value of C_{ef} depends on two factors; namely, the capacitance C being charged/discharged and the activity weight α, a measure (or probability) of the *actual* switching activity:

$$C_{ef} = \alpha \cdot C \tag{2}$$

Changing supply voltage results in increased circuit delay, which is computed by,

$$D = k \cdot \frac{V_{dd}}{(V_{dd} - V_t)^2} \tag{3}$$

where k is a constant, and V_t is the threshold voltage (i.e., the minimum voltage that can be supplied to the processor allowing full and correct functionality). According to [10] the time overhead associated with voltage switching significant in the order of 100s to 10000s of instructions in modern microprocessors. Fortunately, the computation itself can continue during the voltage change period. From Equations (1) and (3), it is easy to see that one can reduce the power (and energy) consumed at the expense of increased delay (reduced speed). However, Equation (3) indicates a roughly linear reduction in speed while Equation (1) hints to *quadratic* gains in power savings with the decrease of V_{dd}. While exploiting this power/speed relation by dynamically adjusting the voltage and frequency, variable voltage scheduling research so far has almost invariably disregarded possible variations in the *effective* switched capacitance C_{ef} by assuming it to be uniform throughout the operation, and especially, *independent of the running task*. The immediate implication of

this assumption is to have *identical* power consumption functions for all the tasks, which largely simplifies the mathematical formulation, while impeding higher power savings, as we argue below.

In fact, as Equation (2) indicates, C_{ef} is proportional to the actual switching weight α, which represents the type of activities of the *running* task. Low-energy software, microarchitecture and compiler optimization research communities have already identified the largely variable relation between the characteristics of the code and the power dissipation. The power dissipation is dependent on the nature of the running software: the locality of reference exhibited, the frequency of bus/memory requests, the chip units on active use such as floating point unit(FPU) or digital signal processor(DSP), even the data structure choices affect the switching frequency in integrated circuits, hence the parameter α [14].

Finally, it is highly likely and desirable that aggressive low power techniques at the circuit, microarchitecture, compiler and OS/scheduler levels are to be integrated in a common framework, which points to the necessity of considering (potentially) different power dissipation functions for the active tasks. For instance, dynamically stopping the clock fed into modules such as FPU, DSP or application specific co-processor which are not used by the running task, is a powerful system-level energy management technique [19], resulting naturally in different switching activities for different tasks. Recently, tasks with different power functions have been explored in [13]; that work proposes an empirical way to carry out scheduling which takes into consideration the power and criticality of ready tasks. The authors base their experimental results on the measurements obtained from [5], which measures the different power consumptions of tasks in a Palm Pilot Professional.

1.3 Contribution

In this paper, we address variable voltage scheduling of periodic real-time tasks. Our contribution is to provide an efficient solution for tasks with (potentially) different power consumption characteristics. Examples of systems that justify the use of different power consumption characteristics are those in which: a) some tasks will be larger than others and therefore use more of the memory system in addition to the cache; b) some tasks will use the floating point unit more than others; c) some will ship the task to specialized, low-power DSPs.

We note that accounting for different power dissipation functions at the task level is an important yet largely ignored issue. Therefore, our solution includes different power functions for each task. We formalize the scheduling problem as a nonlinear optimization problem and show that a task T_i can run at a constant speed S_i at *every* instance without hurting optimality. We sketch an $O(n^2 \log n)$ algorithm to compute the optimal S_i values. We also prove that the EDF (Earliest Deadline First) scheduling policy can be used to obtain a feasible schedule with these optimal speed values.

2 System Model

We consider a set $\mathbf{T} = \{T_1, \ldots, T_n\}$ of n periodic real-time tasks. The period of T_i is denoted by P_i, which is also equal to the deadline of the current invocation. We refer to the j^{th} invocation of task T_i as T_{ij}. All tasks are assumed to be independent and ready at $t = 0$.

Given a CPU speed determined by a voltage/frequency pair, the worst-case workload is represented by the traditional worst-case execution time (WCET) value. Note that, however, for variable voltage scheduling framework where the actual execution time is dependent on the CPU speed, the worst-case number of required CPU cycles of the workload is a more appropriate measure.

We denote the number of processor cycles required by T_i in the worst-case by C_i. Note that, under a constant speed S_i (given in cycles per second), the execution time of the task is $t_i = \frac{C_i}{S_i}$. A schedule of periodic tasks is **feasible** if each task T_i is assigned at least C_i CPU cycles before its deadline at every invocation. We assume that the CPU speed can be changed between a minimum speed S_{min} (corresponding to a minimum supply voltage level necessary to keep the system functional) and a maximum speed S_{max}. Without loss of generality, we assume $0 \leq S_{min} \leq S_{max} = 1$; that is, we normalize the speed values with respect to S_{max}.

The speed $S(t)$ of the processor at time t, can be dynamically varied by adjusting the supply voltage and the clock frequency. In case that there is an imposing bound on the rate of voltage change, the *voltage change overhead* can be incorporated in the worst-case workload of each task. The power consumption of the task T_i is given by $g_i(S)$, assumed to be a strictly increasing *convex* function, specifically a polynomial of at least second degree.

If the task T_i occupies the processor during the time interval $[t_1, t_2]$, then the *energy* consumed during this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g_i(S(t))dt$. The total energy consumed in the system up to $t = t_2$ is therefore $E(0, t_2)$. Finally, a schedule is **power-optimal** if it is feasible and the total energy consumption $E(0, P)$ is minimal, where P is the Least Common Multiple of $P_1 \ldots P_n$. Note

that it is necessary and sufficient to minimize the energy consumption during P, since the schedule repeats itself after each P time units.

3 Optimal Static Solution

In this section, we investigate the static optimal solution to the variable voltage scheduling problem, assuming that each task presents its worst-case workload to the processor at every instance. Slowing down the CPU to match the exact workload of the system is an attractive solution. For example, if $load = \sum(C_i/P_i) = 0.6$ (under S_{max}), the speed of the processor will be set to 0.6 for all tasks. This achieves a uniform speed, and allows for a very good result with respect to energy savings. The following example shows this approach in more detail.

Consider a set with two tasks, T_1 has $P_1 = 100, C_1 = 40$, and $g_1(S) = 3S^3$, while T_2 has $P_2 = 200, C_2 = 40$ and $g_2(S) = S^3$. (Note that, although in this example we consider the function g to be a polynomial of degree 3, other examples can be easily constructed with lower-degree power functions, to demonstrate the same point.) Note that, under S_{max}, the utilization of the set is 0.6. Since the energy consumed by task i executing continuously in interval of time I is $I \cdot g_i(S_i)$, the total energy consumed during the least common multiple of the task periods is $P \cdot (g_1 \frac{C_1}{S_1 P_1} + g_2 \frac{C_2}{S_2 P_2}) = P \cdot (3S_1^2 \frac{C_1}{P_1} + S_2^2 \frac{C_2}{P_2}) = 3S_1^2\ 2C_1 + S_2^2\ C_2$. This is because $\frac{P}{P_i}$ is the number of instances of task i within the least common multiple (LCM) of all task periods, and $\frac{C_i}{S_i}$ is the length of each instance.

The trivial approach would be to use always the maximum speed $S_{max} = 1$ and shutdown the processor whenever idle (favoring this scheme, assume that idle processors consume no energy). This would give a total energy consumption of 280. Alternatively, using the "uniform slow-down" method to fully utilize the CPU time available, we would set $S_1 = S_2 = 0.6$, yielding total energy consumption of 101. These are significant savings, which can be improved upon, as shown below.

An optimal algorithm may have more complex decisions to make. To start with, we observe that given a task worst case CPU requirement C_i and the time t_{ij} assigned by the scheduler (we make no assumption about the scheduling discipline) for executing T_i for the j^{th} invocation, the optimal speed assignment for this invocation is $S_{ij} = \frac{C_i}{t_{ij}}$. When one considers preemptions, the nature of this formula does not change: let instance T_{ij} run (with possible preemptions) during time intervals $[t_1, t_2], [t_3, t_4], \ldots, [t_{m-1}, t_m]$; we can safely commit to a constant speed S_{ij} given by:

$$S_{ij} = \frac{\int_{t_1}^{t_2} S(t)dt + \int_{t_3}^{t_4} S(t)dt + \ldots + \int_{t_{m-1}}^{t_m} S(t)dt}{(t_2 - t_1) + (t_4 - t_3) + \ldots + (t_m - t_{m-1})}$$

during the entire lifetime of S_{ij}. By doing so, we achieve a minimal amount of energy consumption: this follows from the convex nature of power (energy) / speed relation. However, the (potentially) non-identical power consumption functions and the feasibility requirement prevent us from reaching further conclusions at this moment (e.g., the speed settings for each invocation could be different or the speed of processor for all tasks could be the same).

Since the speed S_{ij} of an instance is constant throughout its execution, the energy consumed by T_{ij} (under the worst-case workload C_i), or E_{ij} is simply $g_i(S_{ij}) \cdot t_{ij} = g_i(S_{ij}) \cdot \frac{C_i}{S_{ij}} = E_i(S_{ij})$. We note that $E_i(S_{ij})$ is a strictly increasing convex function, since $g_i(S_{ij})$ is assumed to be a polynomial of at least second degree. At this point, we are ready to state the nonlinear optimization formulation of the variable voltage scheduling problem, which we denote by POW-OPT:

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{P/P_i} E_i(S_{ij}) \quad (4)$$

$$\text{subject to} \sum_{i=1}^{n} \sum_{j=1}^{P/P_i} \frac{C_i}{S_{ij}} \leq P \quad (5)$$

$$S_{min} \leq S_{ij} \leq S_{max},\ i \in [1, n],\ j \in [1, \frac{P}{P_i}] \quad (6)$$

$$\text{A feasible schedule exists with } \{S_{ij}\} \quad (7)$$

Above, the expression (4) indicates our aim to minimize the sum of energy consumptions over all the instances of all the tasks, during the LCM P. Recall that, given a number of cycles C_i to execute under speed S_{ij}, the execution time of a task T_{ij} is $t_{ij} = \frac{C_i}{S_{ij}}$. Thus, the constraint (5) merely states that the total processor demand during the LCM P should not exceed the available computing capacity, which is a necessary (but not sufficient) condition for schedulability. Further, the constraint set (6) encodes the fact that it is only natural to have lower and upper bounds on the CPU speed, imposed by the technology in use. Finally, we have to make sure that there exists at least one feasible schedule with S_{ij} values, as required by constraint (7). Note that we have deliberately omitted the scheduling policy above: it need not and cannot be specified at this point.

Note that, if $\sum_{i=1}^{n} \frac{P}{P_i} \frac{C_i}{S_{max}} > P$, or equivalently, if $\sum_{i=1}^{n} \frac{C_i}{P_i \cdot S_{max}} > 1$, there is no solution to POW-OPT. In other words, we should have at least one feasible schedule by running all the task instances at S_{max}. Hence,

hereafter, we assume that $\sum_{i=1}^{n} \frac{C_i}{P_i \cdot S_{max}} \leq 1$ and there exists at least one feasible schedule.

Theorem 1 (Equal speeds for all instances) *Given an instance of Problem POW-OPT, there exists an optimal solution where the speed of a task T_i is* constant *at* every *instance, that is,*

$$S_{ij} = S_{ik} = S_i \quad 1 \leq j \leq k \leq \frac{P}{P_i}.$$

Proof: Informally, the theorem can be justified by observing that committing to a constant S_i value for all instances of T_i hurts neither feasibility nor optimality. A set of periodic tasks with constant worst-case execution times, can be scheduled as long as the total utilization is less than 1, that is as long as the constraint (5) is satisfied. On the other hand the objective function (4) is also minimized by this choice, since the convexity favors identical execution times in the optimal solution.

To obtain a formal proof, one should observe that after replacing S_{ij} by $\frac{C_i}{t_{ij}}$ and making an initial allocation of $t_{i,min} = \frac{C_i}{S_{max}}$ (lower bound on execution time) to each task T_i, one can easily obtain another convex minimization (concave maximization) problem, which is analogous to the one occurring in the Reward-Based Scheduling Problem [1]. That work formally proves the optimality of identical service times/identical speed per instance given analogous constraints. □

The result of the previous theorem says that, for a particular task, all instances will execute for the same amount of time. This transforms the problem into a periodic real-time model, where the utilization of a task is defined by the ratio $\frac{C_i}{P_i}$. In this case, it is clear that the following corollary holds.

Corollary 1 *When used along with the optimal S_i assignments, any periodic hard real-time policy which can fully utilize the processor (e.g., EDF, LLF), can be used to obtain a feasible schedule.*

The above theorem and corollary guide us in the development of a solution for problem POW-OPT. Before we present the solution, we note that once we commit to identical S_i values at the task-level, we obtain the following optimization problem:

$$\text{minimize} \quad \sum_{i=1}^{n} \frac{P}{P_i} E_i(S_i) \tag{8}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{P}{P_i} \frac{C_i}{S_i} \leq P \tag{9}$$

$$S_{min} \leq S_i \leq S_{max} \quad i = 1, \ldots, n \tag{10}$$

Replacing S_i above by $\frac{C_i}{t_i}$, we transform the function E. Since C_i is a characteristic of task i, $E_i(S_i) = E_i(\frac{C_i}{t_i}) = E_i'(t_i)$ allowing us to re-write the optimization problem as:

$$\text{minimize} \quad \sum_{i=1}^{n} \frac{P}{P_i} E_i'(t_i) \tag{11}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \frac{P}{P_i} t_i \leq P \tag{12}$$

$$l_i \leq t_i \leq u_i \quad i = 1, \ldots, n \tag{13}$$

where $l_i = \frac{C_i}{S_{max}}$ and $u_i = \frac{C_i}{S_{min}}$ $(i = 1 \ldots n)$ are the lower and upper bounds, respectively. Finally, substituting $t_i' + l_i$ for t_i we get the optimization problem:

$$\text{minimize} \quad \sum_{i=1}^{n} \frac{P}{P_i} E_i'(t_i' + l_i) \tag{14}$$

$$\text{subject to} \sum_{i=1}^{n} \frac{P}{P_i} (t_i' + l_i) \leq P \tag{15}$$

$$0 \leq t_i' \leq u_i - l_i \quad i = 1, \ldots, n \tag{16}$$

which is in fact an instance of the concave maximization (convex minimization) problem OPT-LU, for which an optimal $O(n^2 \cdot \log n)$ algorithm has been provided in [1].

Summary of algorithm, derived from [1] Algorithm OPT-LU first notes that the optimal results are when relation (15) is an equality, since the other cases can be either dismissed or improved to comply with the equality constraint. Second, with inequality constraints, we solve this optimization problem using Lagrangian and auxiliary techniques complying with [3, 4].

Then, we address the problem that considers only the equality constraints without the inequality constraints (this problem is denoted by OPT). It is possible to find the solution of OPT, by using the solutions from Lagrangian. For many simple power functions, closed formulas can be obtained; in general, the solution can be obtained in time $O(n)$ [1].

The solution proceeds by considering only the equality and lower bound constraints (call this the OPT-L problem). It invokes the solution to OPT, checks the inequalities (constraints), sets the times assigned to tasks that violate the constraints to zero, and re-invoke OPT for the remaining tasks. Although the number of invocations is bounded by n, we use a binary-search like technique that makes the complexity of the procedure $O(n \cdot \log n)$.

If no solution is found that complies with lower bounds, then there is no solution for our problem. On the other hand, if a solution is found for OPT-L **and** the upper-bounds are not violated, we are done. If the upper-bounds are violated, partial adjustments are done

in the solution (at least one variable is adjusted) and the problem OPT-L is invoked again. Since there is at least one adjustment per iteration, at the end of $O(n)$ iterations the procedure is done. Since the complexity of OPT-L is $O(n \cdot \log n)$, the complexity of OPT-LU is $O(n^2 \cdot \log n)$.

Returning to the example given above, our procedure will set $S_1 = 0.538$ and $S_2 = 0.776$, which are optimal speed assignments. This will yield a total energy consumption of 93 units for the example task set. Clearly, this is only an example, showing some of the savings that could be achieved by our optimal algorithm. Below we present preliminary simulation results to indicate the type of savings we can obtain with our algorithm.

4 Experiment Results

We implemented a simulator to assess the performance of the algorithm developed here. The metric used was power savings over the static power management scheme, which slows down the processor proportionally to the load imposed on the system. This conservative and fast scheme only uses the WCET and deadline information and distributes the slack proportionally among all the tasks in the system, prior to run time. In other words, each task is executed with speed $S_i = load$, where $load = \sum \frac{C_i}{P_i}$. For example, if the load on the system is 80%, the speed of the processor is set to 0.8 uniformly for all tasks. This conservative scheme is based on the principle that in the convex optimization problem given by (8), (9) and (10), the optimal solution consists in assigning equal speed values for each task, in the case that all objective functions are *identical*. That approach guarantees also that all tasks finish before their deadlines, but is sub-optimal in our settings where we provision for non-identical power consumption functions.

We created a synthetic task set with 30 tasks, and associated power functions with each of these tasks. The power functions are of the form kS^3, where k is a parameter of the simulator. To cover a spectrum of the power function, we looked at the following two cases:

- **Uniform Distribution**, where the coefficient of the power functions of the tasks are distributed uniformly between 1 and k. In the experiments $k \in [1, 8]$; in other words, the power functions are of the form kS^3, where k varies uniformly between 1 and 8.

- **Bimodal distribution**, represents the case in which there are two types of tasks in the system: those that consume little power and those that consume much power. More specifically, in our graphs, the bimodal distribution is created such that 60% of the tasks have a power function coefficient value of 1, while the remaining 40% have a value of k. In other words, if $k = 4$, 60% of the power functions are S^3 and 40% are of the form $4S^3$.

The use of the bimodal distribution for power functions comes from applications in the embedded device arena. For example, consider a satellite participating as a multicast router for multimedia sessions: some channels have more stringent timing constraints while others have less compute-intensive traffic. For example, video channels may have to send much more data in the same amount of time than the audio channels, therefore requiring special hardware to compress the data captured. This would increase the power consumption of the video applications considerably. Another application is when the underlying transport technology is an *ad hoc* network, in which location of satellites may change with time. In this case, if multicast groups follow a bimodal distribution [2], some groups are localized (requiring small amounts of energy) while other groups have to increase the power of the transmitter to reach satellites at farther locations. In general purpose computing systems implemented in small devices, this bimodal distribution can be seen when some tasks are small enough to fit on the L1 cache and do not need any external devices; in this case, their coefficient is 1. But if they do not follow this type of restriction, their coefficient is k.

For the first experiment, we set the worst-case utilization of the task set to 70%, under maximum speed (i.e., $S_{max} = 1$). The period and specific computation times were chosen randomly such that the task set utilization is under 70%; the periods were chosen to be always under 32,000 time units (assumed to be microseconds).

Figure 1 results on the performance improvement of our optimal static assignment of speeds, compared with that of the static load-based approach. We can see that, in general, when the power coefficient increases, the gain is larger. Note that a savings of approximately 4 times can be obtained in case of $k > 4$ for the bimodal distribution over the uniform distribution, and around 25 % over the static scheme. Realistically, in the examples of applications we have studied, the average of k is 4, and therefore a gain of up to approximately 13% can be obtained for the bimodal distribution. This means that, in a satellite that is scheduled to be in space for 1 year, by using our solution, the satellite would be able to orbit for approximately an extra 7 weeks.

The gains for the uniform distribution of power were

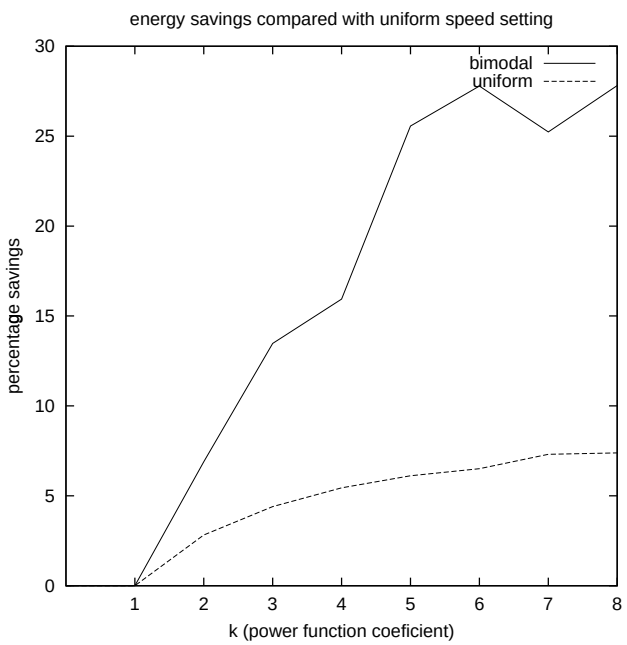

Figure 1: Energy savings compared with statically setting processor speed according to system load, load = 70%

less impressive; this is because when the consumption of energy by the tasks is evenly-distributed, the optimal speed settings is close the load-dependent schemes. However, there is still a 5-7% gain that can come from our algorithm.

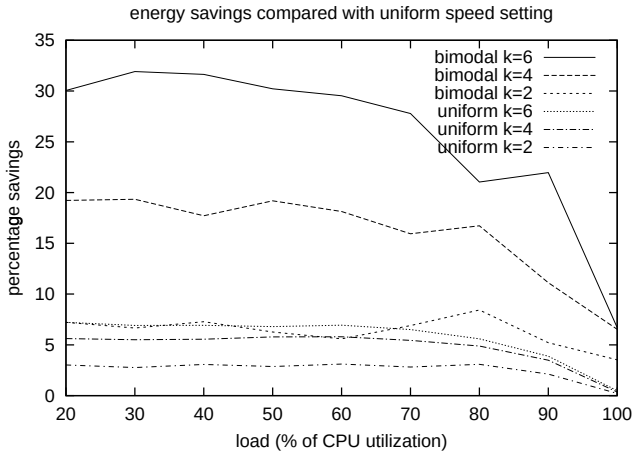

Figure 2: Energy savings compared with setting processor speed according to system load

The results shown above are for 70% load; we carried out simulations changing the load and observed the following behavior: load changes do not affect the results significantly, for loads less than 80%. This is because the static optimal solution uses the timeline fully and this is a schedule produced with information about the WCET of the applications. In Figure 2, we plotted the energy savings as a function of the load in the system, for $k = 2, 4, 6$. As we can see, the gains drop significantly for high loads, as described above, but are very consistent for lower values of load. Further, it is clear that regardless of the distribution of the power coefficients, there is always a gain by our optimal algorithm. The graph shows the trends for energy savings for the optimal algorithm.

5 Conclusion

We presented an algorithm for power-aware scheduling with variable voltage scheduling for periodic real-time tasks. Our $O(n^2 log n)$ algorithm considers different power consumption characteristics and shows that the gain over more simplistic power management algorithms is significant. Our solution is to formalize the scheduling problem as a nonlinear optimization problem and show that a task T_i can run at a constant speed S_i at *every* instance without hurting optimality. Our preliminary simulation results show that the more difference there is in power consumption among tasks in a system, the more the system can profit from optimal power management. Our motivating examples come primarily from satellite and other communication devices that require batteries with long lives, impacting not only the lifetime of the device, but also the quality of results generated by these devices. We also proved that the EDF (Earliest Deadline First) scheduling policy can be used to obtain a feasible schedule with the optimal speed values computed by our algorithm.

Future work Although technological advances are rapidly making the adjustment of CPU speed in a continuous spectrum a reality [6, 15], some systems are still designed with a finite (2 to 4) voltage/speed levels[1]. These values are chosen as specific points on the power/speed curve. Hence, although in principle we may again apply our continuous solution to find optimal S_i values, these may not correspond to any of the speed levels available on the system. However, it is easy to prove that we can achieve S_i (in fact, any desired S_i) with at most one voltage/speed switch during the exe-

[1] The Crusoe processor from Transmeta has the ability to set the CPU speed from 200MHz to 666MHz, which falls between these two ends of the spectrum.

cution of any given instance. This issue is subject of an upcoming paper.

References

[1] H. Aydın, R. Melhem, D. Mossé and P.M. Alvarez. Optimal Reward-Based Scheduling of Periodic Real-Time Tasks. In *Proceedings of 20th IEEE Real-Time Systems Symposium (RTSS'99)*, Phoenix, December 1999.

[2] Kenneth Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu and Yaron Minsky. Bimodal Multicast. ACM Transactions on Computer Systems, 17(2), pp 41-88, May 1999.

[3] J. K. Dey, J. Kurose, D. Towsley, C.M. Krishna and M. Girkar. Efficient On-Line Processor Scheduling for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. *Proceedings of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems,* May 1993.

[4] J. K. Dey, J. Kurose and D. Towsley. On-Line Scheduling Policies for a class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks. *IEEE Transactions on Computers* 45(7):802-813, July 1996.

[5] C. Ellis. The case for higher-level power management. In *IEEE 7th Workshop on Hot Topics in Operating Systems (HotOS-VIII.* pp. 162-167, 1999.

[6] V. Gutnik and A. Chandrakasan. An Efficient Controller for Variable Supply Voltage Low Power Processing. *Symposium on VLSI Circuits*, pp.158-159, 1996.

[7] P. J. M. Havinga and G. J. M. Smith. Design Techniques for Low power systems. *Journal of Systems Architecture*. Vol. 46:1, 2000

[8] I. Hong, D. Kirovski, G. Qu, M. Potkonjak and M. Srivastava. Power optimization of variable voltage core-based systems. In *Design Automation Conference*, 1998.

[9] I. Hong, M. Potkonjak and M. B. Srivastava. On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor. In *Computer-Aided Design (ICCAD)'98*. pp. 653-656.

[10] I. Hong, G. Qu, M. Potkonjak and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, December 1998.

[11] Intel, Microsoft, and Toshiba. Advanced Configuration and Power Management Interface (ACPI) Specification, 1999. www.intel.com/ial/WfM/design/pmdt/acpidesc.htm.

[12] C. M. Krishna and Y. H. Lee. Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems. In *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, Washington D.C., May 2000.

[13] T. Ma and K. Shin. A User-Customizable Energy-Adaptive Combined Static/Dynamic Scheduler for Mobile Applications. In *Proceedings of 21th IEEE Real-Time Systems Symposium (RTSS'00)*, pp 227-236, 2000.

[14] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen and D. Ghosh. Techniques for low Energy Software. In *Proceedings of the 1997 International Symposium on Low Power Electronics*. August 1997, Monterey, CA.

[15] W. Namgoang, M. Yu and T. Meg. A High Efficiency Variable-Voltage CMOS Dynamic DC-DC Switching regulator. *IEEE International Solid-State Circuits Conference*, pp.380-391

[16] M. Pedram. Power Minimization in IC Design: Principles and Applications. *ACM Transactions on Design Automation of Electronics Systems*. 1:1 - pp. 3-56, January 1996.

[17] Y. Shin and K. Choi. Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems. In *Proceedings of the 36th Design Automation Conference, DAC'99*, pp. 134-139.

[18] M. Srivastava, A. P. Chandrakasan and R. W. Brodersen. Predictive System Shutdown and other Architectural Techniques for Energy Efficient Programmable Computation. *IEEE Transactions on VLSI Systems*, 4(1): 42-55, 1996.

[19] G. E. Tellez, A. Farrahi and M. Sarrafzadeh. Activity-driven clock design for low power circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, 1995.

[20] F. Yao, A. Demers and S. Shankar. A Scheduling Model for Reduced CPU Energy. *IEEE Annual Foundations of Computer Science*, pp. 374 - 382, 1995.