Dynamic I/O Power Management for Hard Real-time Systems[1]

Vishnu Swaminathan[†], Krishnendu Chakrabarty[†] and S. S. Iyengar[‡]
[†]Department of Electrical & Computer Engineering
Duke University
Durham, NC 27708, USA
[‡]Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803, USA

ABSTRACT

Power consumption is an important design parameter for embedded and portable systems. Software-controlled (or dynamic) power management (DPM) has recently emerged as an attractive alternative to inflexible hardware solutions. DPM for hard real-time systems has received relatively little attention. In particular, energy-driven I/O device scheduling for real-time systems has not been considered before. We present the first online DPM algorithm, which we call **L***ow* **E***nergy* **De***vice* **S***cheduler (LEDES), for hard real-time systems. LEDES takes as inputs a predetermined task schedule and a device-usage list for each task and it generates a sequence of sleep/working states for each device. It guarantees that real-time constraints are not violated and it also minimizes the energy consumed by the I/O devices used by the task set. LEDES is energy-optimal under the constraint that the start times of the tasks are fixed. We present a case study to show that LEDES can reduce energy consumption by almost 50%.*

1 Introduction

Energy consumption is now recognized as an important design parameter for portable and embedded systems. Since the amount of power available to these systems is limited, it is desirable to minimize energy consumption such that the life of the battery or battery pack can be extended. Energy conservation is especially important for embedded systems, such as concealed sensors which are deployed in the field and which cannot be physically accessed.

There are several approaches to energy conservation via power management. Figure 1 classifies these approaches into several broad categories.

The bulk of system power is consumed by the processor (CPU) and the peripheral devices, especially hard disks [7, 11]. Processor designs incorporate several power management features. Several types of idle, standby and sleep modes suspend processor operation during periods of inactivity [4]. Designers also incorporate automatic power saving features into their processors. Unused execution units are shutdown automatically. Power consumption of the CPU is also directly related to the clock speed. Many processors use a variable-speed clock that may be tuned to achieve the optimum performance for the application while at the same time minimize power consumption.

Even more energy savings can be achieved if a software power management system is used to take advantage of the power reduction features of the hardware. In 1997, Intel, Toshiba America Information Systems and Microsoft developed the Advanced Configuration and Power Interface (ACPI) standard for desktop and notebook systems [1]. This transfers the power reduction responsibility from the hardware (BIOS) to the software (OS). The OS possesses information about new applications and appropriately schedules tasks and devices in a power-aware manner. Power management by the OS is commonly called *dynamic power management* (DPM). When applied to the CPU, DPM involves varying the clock speed dynamically (making use of the fact that power is quadratically proportional to the clock speed) to reduce power, while at the same time meeting the application's processor requirements [8]. A common DPM method used for minimizing the power consumed by I/O devices is based on timeouts—shutting down a device when it has been idle long enough. Other DPM approaches involve the observation of past requests to predict future idleness, stochastic models, and increasing the lengths of idle periods by re-ordering task execution [2, 3, 10].

Real-time systems are characterized by tasks that have hard deadlines. However, the inclusion of deadlines as an additional design constraint makes DPM for real-time systems difficult. So far, most research on real-time DPM has been CPU-centric [5, 6, 12, 13]. I/O-centric DPM has mostly been studied for non-real-time environments [2, 3, 9, 10]. Although these I/O-centric methods have resulted in power savings of over 50%, they cannot be applied to real-time systems due to their inherently probabilistic nature. In [9], a *device-utilization matrix* is used to keep track

[1]This research was supported in part by DARPA under grant no. N66001-001-8946 and in part by a graduate fellowship from the North Carolina Networking Initiative.

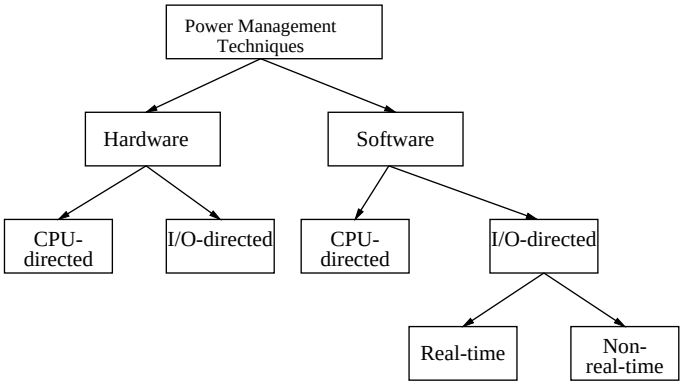

Figure 1. Power management methods.

of device usage. When the utilization of a device falls below a threshold, the device is put into the sleep state. This method is called *task-based power management*. In [10], tasks are reordered such that the I/O requests for each device are clustered together. This results in extended idle times for each device. Hence each device can be shut down for long periods of time resulting in greater energy savings. However, these methods implicitly assume there is no task execution penalty associated with the shutdown of devices. In real-time systems, the penalty for missing a task deadline is enormous. Therefore, it is critical in real-time systems to have I/O devices powered up and running at the correct times to guarantee that all application tasks meet their deadlines. Moreover, these methods are oriented towards minimizing energy consumption while maintaining low response time. These methods work well for Unix-like systems with a round-robin scheduling policy. With the inclusion of *hard* deadlines, probabilistic methods become inapplicable.

In this paper, we present a novel I/O-centric DPM algorithm for hard real-time systems. We are given a precomputed task schedule that guarantees that all tasks meet their respective deadlines. The start times of all the tasks are fixed and cannot be changed at run-time. Each I/O device can be in one of two states—a low-power sleep state and a high-power working state. Scheduling occurs at the start and completion of each task. We assume that the OS has complete knowledge of the device usage of each task. Our algorithm schedules the wake-up and sleep times for the devices such that the energy consumed by the devices is minimized, while guaranteeing that no task misses its deadline. The algorithm is optimal under the constraints that the start times of the tasks are fixed and may not be changed at run-time. To the best of our knowledge, this is the first attempt at minimizing the energy consumption of I/O devices for real-time systems.

A more precise description of the problem is given in the next section.

2 Problem Statement

In this section, we present our notation and the underlying assumptions. We are given a task schedule $S = s_1 s_2 s_3 \ldots s_n$ consisting of the start times for a set R of n tasks. Associated with each task $r_i \in R$ are the following parameters:

- its *release* (or arrival) time a_i, where $a_i \leq s_i$,
- its deadline d_i,
- its completion (or execution) time c_i, and
- a *device usage list* k_i, consisting of all the devices used by r_i.

We are also provided with a set K of devices used in the system. The following parameters are associated with each device $k \in K$:

- two power states—a low-power sleep state low_k and a high-power working state $high_k$,
- a transition time from low_k to $high_k$ represented by t^k_{wu},
- a transition time from $high_k$ to low_k represented by t^k_{sd},
- power consumed during wake-up P_{wu},
- power consumed during shutdown P_{sd},
- power consumed in the working state P_w, and
- power consumed in the sleep state P_s.

We assume that requests can be processed by the devices only in the working state. The start times $s_1, s_2, \ldots s_n$ of the tasks are fixed and cannot be changed. Each task must complete its execution by its associated deadline. Initially, at time $t = 0$, all devices are powered up. The power consumed by a device in the sleep state is less than the power consumed during transition, which in turn is less than the power consumed in the working state, i.e., $P_s < P_0 < P_w$. We assume without loss of generality that $t^k_{wu} = t^k_{sd} = t_0$ and $P_{wu} = P_{sd} = P_0$. The execution times $c_1, c_2, \ldots c_n$s of the tasks are all greater than the transition time t_0. The energy consumed by device k is $E_k = P_w t_w + P_s t_s + m P_0 t_0$, where m is the number of state transitions, t_w is the total time spent by the device in the working state, and t_s is the total time spent in the sleep state.

The problem we address is that of online device scheduling, i.e., determining a sequence of states for each device k such that the total energy $\sum_k E_k$ is minimized while ensuring that all tasks meet their deadlines.

In the following section, we present our approach to DPM for real-time systems and the basic underlying theory.

3 Background and Approach

There are several approaches to I/O DPM for real-time systems. The best solution, i.e., one that has the least energy consumption while guaranteeing that all tasks meet their respective deadlines) can only be found by examining the entire taskset and re-ordering the tasks. This will allow I/O devices to be left in the sleep (low-power) state for as long as possible. However, for large tasksets, this is not possible

Task	Arrival time	Completion time	Deadline	Device list
r_1	0	3	4	k_1, k_3
r_2	1	2	6	k_3
r_3	3	5	10	k_2, k_1
r_4	14	3	18	k_3
r_5	17	3	21	k_1, k_2, k_3

Table 1. Taskset example.

online since the scheduler must schedule tasks with minimum delay.

A simpler but non-optimal approach is to schedule devices given a predetermined task schedule. In this scenario, it is possible to adjust the start times of tasks within the constraints of their arrival time and deadline. This approach is less complex than the previous one but still not practically feasible for an online algorithm.

For large tasksets that need to be scheduled online, our algorithm guarantees optimality given that the task start times are fixed. This ensures that I/O device energy consumption can be reduced without missing task deadlines or incurring scheduling overhead. The remainder of this section discusses and explains the underlying concepts that are used in our algorithm.

Table 1 is a simple taskset of 5 tasks and 3 devices that will be used as a running example. We assume that $t_0 = 1$ unit, $P_0 = 3$ units, $P_w = 5$ units and $P_s = 1$ unit. The schedule for this taskset is shown in Figure 2. Arrival times are marked with upward-pointing arrows and deadlines with arrows that point downward.

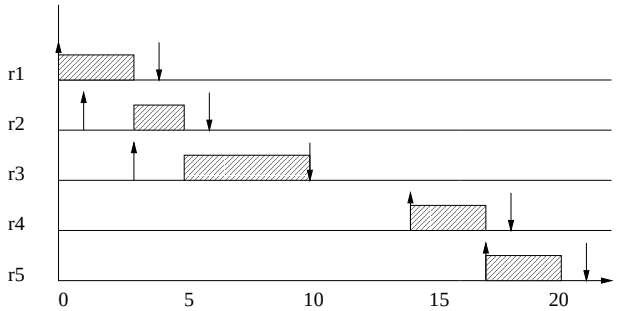

Figure 2. Task schedule for the taskset of Table 1.

From this example, we can easily see that without a priori knowledge of future device usage, it is not possible to guarantee timely completion of tasks. A naive algorithm that shuts devices down when they are unused cannot be used for real-time tasks. At $t = 5$ (start of task r_3), if devices k_2 and k_1 are shutdown (since they are not used by r_2), r_3 will miss its deadline. Devices k_2 and k_1 take 1 unit of time to enter the working state and this causes r_3 to start at $t = 6$ units. Since $c_3 = 5$ units, r_3 completes at $t = 11$ units, which is greater than the deadline.

The following theorem shows that if task start times are fixed a priori, timeliness can be ensured with a limited amount of look-ahead for I/O device usage. This allows dynamic power management with low scheduling overhead.

Theorem 1 *Given a task schedule for a set R of n tasks with completion times $c_1, c_2, \ldots c_n$, a set K of I/O devices, and the device utilization for each task, it is necessary and sufficient to look ahead only m tasks to guarantee timeliness, where $\sum_{i=1}^{m} c_i \geq t_0$.*

Proof: First we prove necessity. Let us assume that at some scheduling instant s_i, task r_i is being scheduled and r_i uses device k_l. Also, suppose task $r_j, j \neq i$, uses device $k_p, p \neq l$, $s_j - s_i < t_0$ and $s_j + c_j = d_j$. Further, k_l is powered up and k_p is in the sleep state. We can easily see that without look-ahead, at s_j, k_p is not powered up and cannot serve requests for r_j. This means k_p needs to be powered up before r_j can start. This results in $s_j + c_j + t_0 > d_j$. Moreover, if $s_j - s_i < t_0$, the device will not have enough time to wake up before s_j. Hence we need to look ahead m tasks such that $\sum_{i=1}^{m} c_i \geq t_0$.

We prove sufficiency as follows. As long as there is a look-ahead of m, we can guarantee that there is at least one valid scheduling instant between the start times of two tasks that require different devices. This leads to the conclusion that with a look-ahead of m, there is sufficient time to wake a device up before the start time of a task requesting it. This completes the proof of the theorem. □

In most practical cases, the completion times of all tasks in the taskset are greater than the transition times t_0 of the devices. This leads to the following corollary to Theorem 1.

Corollary 1 *Given a task schedule for a set R of tasks with completion times $c_1, c_2, \ldots c_n$, a set K of I/O devices, and the device utilization for each task, it is necessary and sufficient to look ahead one task to guarantee timeliness if the completion times of all tasks $r_i \in R$ is greater than the transition time t_0 of the devices.*

We next show that if start times of the tasks are fixed, the energy consumed by the I/O devices is minimum if the limited look-ahead given by Theorem 1 is used.

Theorem 2 *Given a task schedule for a set R of tasks and a set K of devices, the schedule generated using the look-ahead strategy of Theorem 1 is energy-optimal if the task start times are fixed.*

Proof: We prove the theorem by contradiction. The energy consumption is not optimal if some devices are left in the powered up state when they should in fact be in the sleep state. But by construction of the algorithm, we ensure that any devices not used in the immediate future (i.e., by the currently scheduled task and the task immediately following

it) are in the sleep state. Hence, we see that no device is left in the powered up state unnecessarily. □

To summarize, we have shown that without a look-ahead of m tasks such that $\sum_{i=1}^{m} c_i \geq t_0$ for the general case (one task if all completion times are greater than t_0), it is not possible to use DPM for real-time systems. This implies a certain amount of book-keeping overhead for the OS in keeping track of task order and per-task device usage lists. In the next section, we explain our algorithm and provide an example of the operation of the algorithm.

4 The LEDES Algorithm with an Example

We call our scheduling algorithm the **L**ow **E**nergy **De**vice **S**cheduler, or LEDES, and in this section, we explain LEDES and its operation. Figure 3 provides the pseudo-code for it. We assume that initially (at time $t = 0$) all devices in the system are powered up. All completion times are greater than t_0.

4.1 The LEDES algorithm

Procedure LEDES()
begin
Repeat forever
At s_1:
    $\forall k \notin k_1^{'} \cup k_2^{'}$
        Shutdown k
    $\forall k \in k_2^{'} - k_1^{'}$
        if $s_2 - (s_1 + c_1) \geq t_{wu}$ *shutdown* k
At $s_1 + c_1$:
    $\forall k \in k_2^{'}$
        Wake up k
At $s_i, i \neq 1$:
    $\forall k \in k_{i+1}^{'} - k_i^{'}$
        if $s_{i+1} - (s_i + c_i) \geq t_{wu}$ *shutdown* k
        else *Wake up* k
    $\forall k \in k_{i-1}^{'} - k_i^{'} - k_{i+1}^{'}$
        if $c_i \geq t_{sd}$ *shutdown* k
At $s_i + c_i, i \neq 1$:
    $\forall k \in k_{i+1}^{'}$
        Wake up k
    $\forall k \in k_i^{'} - k_{i+1}^{'}$
        if $s_{i+1} - (s_i + c_i) \geq t_{sd}$ *shutdown* k
end

Figure 3. The LEDES algorithm

The algorithm operates as follows. At the first scheduling instant (i.e., the start of the first instance of the first task), all devices not used by the next "immediate" tasks r_1 and r_2 are put in the sleep state (low_k). The "slack" (difference between the start time of the next task and completion time of the present task) is checked. If the slack is greater than the wake-up time t_{wu}, those devices used by r_2 and not by r_1 are put into the sleep state. Since scheduling instants are at every task's start *and* completion times, devices used before r_2 are guaranteed to be awake before r_2's start.

At the next scheduling instant ($s_1 + c_1$), all devices used by r_2 are woken up (since start times of tasks are fixed, we cannot defer scheduling to the next scheduling instant, which is s_2). Further, those devices used by r_1 and not used by r_2 are put in the sleep state if the slack between s_2 and $s_1 + c_1$ is more than the shutdown time t_{sd} of the device. The check is performed so that at the next scheduling instant (s_2), the algorithm must guarantee that the device is fully in the sleep state if there is a need for it to be woken up again.

The process is similar at other scheduling instants, with a few minor changes. At other start times the first check is not performed. This is because at the last scheduling instant ($s_n + c_n$), only those devices used by r_1 will be powered up. Instead, it is modified slightly to consider the effect of previous scheduling decisions. The modified rule ensures that at some start time s_i, devices that are not used by either the present task or the succeeding task but were used by the previous task are shut down correctly. This entire process continues in a periodic manner.

We now apply this algorithm to the taskset shown in Table 1. Figure 4 shows the state of the devices through the "hyperperiod".

4.2 Example

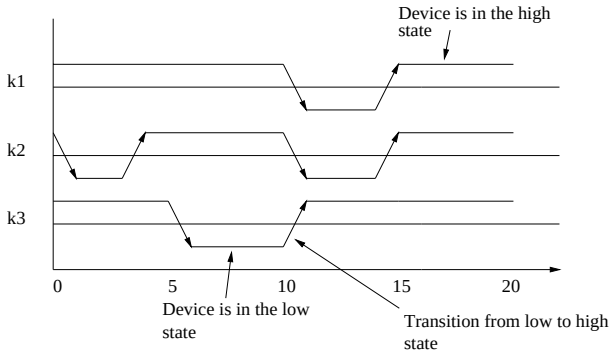

Figure 4. Device schedule for the taskset in Table 1.

A full walk-through of the example taskset shown in Table 1 is both tedious and boring. So we will first comment on it and then point out the salient features that warrant attention.

First, we observe that our algorithm does indeed schedule devices such that all task deadlines are met. This shows that LEDES is a robust device scheduling algorithm in that no real-time constraints are violated. Next, we observe that all devices are powered up before the start times of the tasks that utilize them. This may not be energy-optimal, but the energy savings accrued is still enormous. The underlying reason behind non-optimality is that the start times of the

Task	Arrival time	Completion time	Deadline	Device list
r_1	0	3	5	k_3
r_2	2	7	10	k_2
r_3	11	6	20	k_1, k_5
r_4	20	4	25	k_4
r_5	20	5	30	k_1, k_3, k_5
r_6	30	3	35	—
r_7	31	4	38	k_1, k_2, k_5
r_8	40	2	45	—

Table 2. Taskset with relaxed deadlines.

tasks may not be modified. Finally, we see that the energy saved is about 26%. This may appear to be low, but the difference from an optimal (minimum energy) solution is only 1.8%.

The time points that stand out in the example are $t = 10$, $t = 14$ and $t = 17$. At $t = 10$, since schedule modification is not allowed, device k_3 has to be woken up in order for it to be in the powered up state before r_4's start time. This results in greater energy consumption. To obtain a minimum-energy schedule, we can exploit the fact that both r_4 and r_5 have a difference of 1 time unit between their respective deadlines and completion times. This will have result in the devices k_1, k_2, and k_3 sleeping for an extra time unit. At $t = 14$ all three devices can be scheduled for wake-up. Task r_4 will now start at $t = 15$ and still meet its deadline. Similarly, r_5 will start 1 time unit after the "scheduled" start time and still finish on time. From this, it becomes apparent that to obtain a minimum-energy schedule,
(i) we must allow start times of tasks to be changed, and
(ii) we must consider the device usage of *all* tasks in the system.

In the next section, we present some experimental results. We evaluated LEDES for two separate tasksets, one with relaxed deadlines and the other with very tight deadlines. These tasksets consist of 8 tasks with 5 devices each.

5 Experimental Results

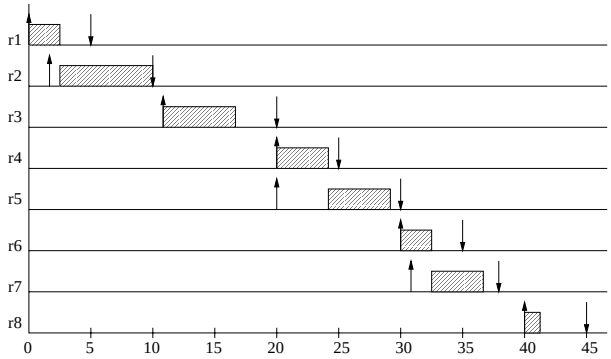

Figure 5. Task schedule for the taskset in Table 2.

We assume a P_w of 5 units, P_s of 1 unit, and a $t_{sd} = t_{wu}$ of 1 unit. We have only considered the case where the completion times of all tasks is greater than or equal to the transition time.

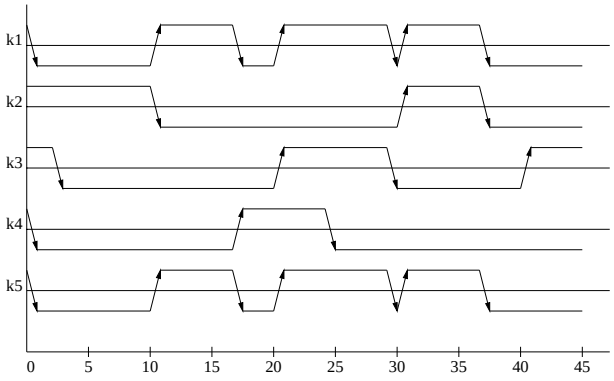

Figure 6. Device schedule for the taskset in Table 2.

Task	Arrival time	Completion time	Deadline	Device list
r_1	0	3	4	k_5
r_2	1	3	6	k_1, k_3
r_3	3	14	20	k_2, k_4, k_5
r_4	10	4	24	k_2, k_3
r_5	20	3	27	k_1, k_5
r_6	25	7	35	k_1, k_2, k_4
r_7	33	6	40	k_5
r_8	40	5	45	k_3, k_2

Table 3. Taskset with tight deadlines.

The two tasksets are shown in Tables 2 and 3. Table 2 shows a taskset with relaxed deadlines. This taskset also contains CPU-intensive tasks (tasks that don't use any devices). Figure 5 shows the task schedule generated by the earliest deadline first (EDF) algorithm. We can also see that there are intervals of slack in the schedule ($t = 10, 15$ and 37) which may be used to achieve greater energy savings. Figure 6 shows the sequence of states each device is in during the hyperperiod. Note that there is a large amount of switching activity. This is because the completion times of the tasks are realtively small, and also because of the slack present in the schedule. The energy consumption through DPM results in 583 units. Compared to the energy consumption of the taskset with all devices in the powered up state at all times (1125 units), this results in energy savings of almost 50%.

Table 3 is an example of a taskset that is more I/O-intensive. Figures 7 and 8 show the corresponding task and device schedules. Observe that device k_5 is used in every alternate task, and that there is no slack between tasks, i.e., there is always a task ready to execute the instant the previous task finishes. In such a situation, k_5 is in the powered up state throughout the hyperperiod. The reduction in energy in this case is not as much as in the example with relaxed deadlines. We see that the schedule generated by LEDES has an energy consumption of 909 units. The energy consumption in the case where all devices are powered up is 1125 units. This is an energy reduction of about 20%. Moreover, compared to the device schedule in Figure 6, the switching activity is much less. This is due to the fact that

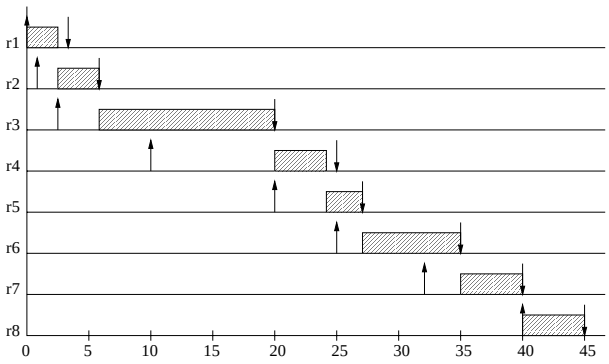

Figure 7. Task schedule for the taskset in Table 3.

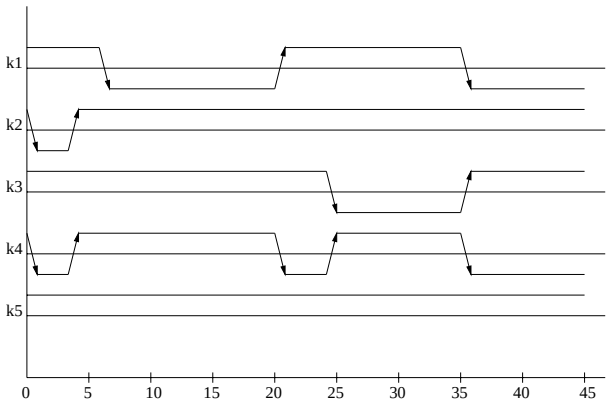

Figure 8. Device schedule for the taskset in Table 3.

the completion times in this taskset is relatively large and there is no slack in the system.

6 Conclusions

In this paper, we have presented a novel device scheduling algorithm (LEDES) for dynamically managing power consumption of I/O devices in a hard real-time system. We have shown that device scheduling for real-time tasksets is not possible without future knowledge of device requests. We have also shown that low-energy I/O schedules can be obtained while at the same time guaranteeing timeliness. Low-overhead online scheduling is achieved by restricting the amount of look-ahead. Further, we have shown that LEDES is energy-optimal under the condition that task schedules cannot be modified. We have shown that DPM for real-time tasks can result in energy savings of over 50%. Finally, we conclude that DPM for I/O devices can provide tremendous energy savings and is of great practical importance for embedded and portable systems.

We are currently extending the LEDES algorithm and evaluating it for scenarios involving more practical I/O devices with upto four power states. Results for these experiments will be presented at the CODES symposium.

References

[1] ACPI. *http://www.teleport.com/∼acpi*

[2] L. Benini, A. Bogliolo, and G. D. Micheli, "A survey of design techniques for system-level dynamic power management", *IEEE Trans. on VLSI Systems*, vol. 8, no. 3, June 2000.

[3] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli, " Policy optimization for dynamic power management", *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 16, no. 6, pp. 813–833, June 1999.

[4] Motorola MPC823 Users' Manual. *ebus.motsps.com/collateral/M951448674345collateral.html*

[5] I. Hong, M. Potkonjak, and M. B. Srivastava, "On-line scheduling of hard real-time tasks on variable-voltage processor", *Proc. Intl. Conf. on Computer-Aided Design*, pp. 653–656, 1998.

[6] C. Hwang and A. C-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation", *Proc. Intl. Conf. on Computer-Aided Design*, pp. 28–32, 1997.

[7] K. Li, R. Kumpf, P. Horton, T. Anderson, "A quantitative analysis of disk drive power management in portable computers", *Proc. Usenix Winter 1994 Conf.*, pp. 279–292, 1994.

[8] J. R. Lorch and A. J. Smith, "Software strategies for poertable computer energy management", *Personal Communications*, vol. 5, no. 3, pp. 60–73, 1998.

[9] Y-H. Lu, L. Benini and G. De Micheli, "Operating system directed power reduction", *Proc. Intl Conf. on Low-Power Electronics and Design*, pp. 37–42, 2000.

[10] Y-H. Lu, L. Benini, and G. De Micheli, "Low-power task scheduling for multiple devices", *Proc. Intl. Workshop on Hardware/Software Codesign*, pp. 39–43, 2000.

[11] M. Newman and J. Hong, "A look at power consumption and performance of the 3Com Palm Pilot", *http://guir.cs.berkeley.edu/projects/p6/finalpaper.html*

[12] V. Swaminathan and K. Chakrabarty, "Investigating the effect of voltage-switching on low-energy task scheduling in hard real-time systems", *To appear in Proc. Asia-South Pacific Design Automation Conf.*, January 2001.

[13] M. Weiser, B. Welch, A. Demers and S. Shenker, "Scheduling for reduced CPU energy", *Proc. Symp. on Operating System Design and Implementation*, pp. 13–23, 1994.