Real-Time Syst (2006) 34:37–51
DOI 10.1007/s11241-006-6738-6

Applying pinwheel scheduling and compiler profiling for power-aware real-time scheduling*

Hsin-hung Lin · Chih-Wen Hsueh

Published online: 3 May 2006

Abstract Energy consumption is a critical design issue in embedded systems, especially in battery-operated systems. Maintaining high performance while extending the battery life is an interesting challenge for system designers. Dynamic voltage scaling and dynamic frequency scaling allow us to adjust supply voltage and processor frequency to adapt to the workload demand for better energy management. Because of the high complexity involved, most solutions depend on heuristics for online power-aware real-time scheduling or offline time-consuming scheduling. In this paper, we discuss how we can apply pinwheel model to power-aware real-time scheduling so that task information, including start times, finish times, preemption times, etc, can be efficiently derived using pinwheel model. System predictability is thus increased and under better control on power-awareness. However, job execution time may be only a small portion of its worst case execution time and can only be determined at runtime. We implement a profiling tool to insert codes for collecting runtime information of real-time tasks. Worst case execution time is updated online for scheduler to perform better rescheduling according to actual execution. Simulations have shown that at most 50% energy can be saved by the proposed scheduling algorithm. Moreover, at most additional 33% energy can be saved when the profiling technique is applied.

Keywords Power-aware real-time scheduling · Pinwheel model · Dynamic voltage scaling · Embedded systems · Profiling

*This paper is an extended version of the paper Power-Aware Real-Time Scheduling using Pinwheel Model and Profiling Technique that appeared in the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.

H. H. Lin
Real-Time Systems Laboratory, Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan 621, R.O.C.
e-mail: lsh@cs.ccu.edu.tw

C. W. Hsueh
Institute of Information Science, Academia Sinica, 128 Sec. 2, Academia Rd, Nankang Taipei, Taiwan 115, R.O.C.
e-mail: cwhsueh@iis.sinica.edu.tw

1. Introduction

In recent years, embedded systems have been rapidly and widely spread, especially mobile systems and portable systems. People enjoy the convenience in all kinds of embedded applications, such as communication, industrial control, medical instrumentation, and entertainment, etc. However, most embedded devices are operated using batteries so that their working duration is limited. With the increasing demand of performance and computing power, such as video playback, energy consumption increases dramatically. Although battery technology also advances along with the devices, battery capacity seems never enough for users. Therefore, energy consumption has been a critical constraint in the design of embedded systems, especially in battery-operated systems such as laptops, PDAs, and cellular phones.

Dynamic voltage scaling (DVS) and dynamic frequency scaling (DFS) allow adjusting processor voltage and frequency at runtime. Usually, higher processor voltage and frequency leads to higher system throughput while energy reduction can be obtained using lower voltage and frequency. Recent trends in modern processor architecture provide support for these two mechanisms. For example, Intel Mobile Processors with SpeedStep technology (http://www.intel.com/index.htm, 2004), AMD Mobile Processors with PowerNow! technology (http://www.amd.com/us-en, 2004), Transmeta Crusoe (http://www.transmeta.com/index.html, 2004) and StrongARM (http://www.intel.com/index.htm, 2004), etc. Instead of lowering processor voltage and frequency as much as possible, power-aware real-time scheduling adjusts voltage and frequency according to some optimization criteria, such as low energy consumption or high throughput, while meeting the timing constraints of real-time tasks. However, reduction of processor voltage and frequency increases the circuit delay, causing slowdown in the execution of programs. Hence, power-aware real-time scheduling makes a trade off between energy saving and system performance. We need to schedule properly where to change the processor voltage and frequency to have the best power-aware performance.

DVS and DFS can be implemented at various levels of a system, such as in the processor, in the OS scheduler, in the compiler or in the application (Unsal and Koren, 2003; AbouGhazaleh et al., 2003). In this paper, we focus on task scheduling algorithms to meet timing constraints while minimizing system energy consumption. Operating system is the only component with an overview of the entire system, including task constraints and status, resource usage, etc. Therefore, we believe it would be one of the most effective and efficient approaches to reduce energy consumption with proper task scheduling algorithms.

The real-time scheduling problem with power optimization constraints is NP-hard (Chen and Muhlethaler, 1996). It is time consuming to find an optimal schedule where energy consumption is minimized and all timing constraints are met. Many previous works either proposed offline scheduling for large energy reduction, or used heuristic methods to reduce scheduling overhead. However, while the former approaches are inflexible and too costly to store in memory, the latter ones may not realize the full potential of energy savings.

As shown in Figure 1, the slack of a job with deadline at d_i at any time $t < d_i$ is equal to $(d_i - t)$ minus the time required to complete the remaining portion of the job (Liu, 2000). Conventional real-time systems are usually overestimated to schedule and provide resources using the worst case execution time (WCET). In average case, real-time tasks rarely execute up to their WCET. In many applications, best case execution time (BCET) is often a small fraction of their WCET (Ernst and Ye, 1997). However, such slack times are only known at runtime. It may be difficult or time-consuming for schedulers to determine whether the next job should utilize the slack time or the system enters sleep mode for the best energy saving.

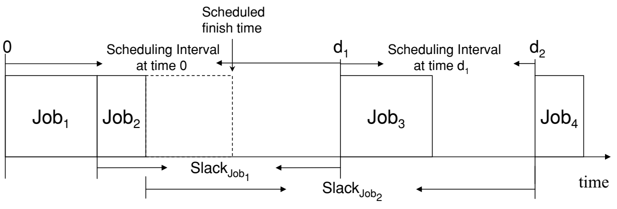

Fig. 1 System Slack Time

In this paper, we describe and analyze the adaption of pinwheel algorithm to power-aware real-time scheduling. Pinwheel scheduling uses a distance-constrained specialization technique to transform all task distance constraints to harmonic numbers (Hsueh and Lin, 2001), which we call pinwheel transformation in this paper. Pinwheel schedule can be generated in polynomial time and space. System predictability is increased and task execution becomes more deterministic. Sufficient scheduling information, such as job start times, finish times, preemption times, etc, can be derived efficiently in polynomial time. Power-aware real-time technique is then used online based on the information to make better scheduling decisions. Furthermore, we implement a profiling tool to provide timing information at program run-time for scheduler to adjust schedule online according to the actual execution time (AET). Slack time thus can be better utilized for power-aware real-time scheduling. We present some power-aware real-time scheduling algorithms using pinwheel model. Simulations have shown that the overhead and complexity of power-aware real-time scheduling is acceptable.

The rest of this paper is organized as follows. The next section introduces some background knowledge and previous works on power-aware real-time scheduling. Section 3 discusses how to apply pinwheel model to power-aware real-time scheduling. In Section 4, we analyze the benefits of adapting pinwheel algorithm. Finally, this paper is concluded in Section 5.

2. Background

2.1. Dynamic voltage scaling and dynamic frequency scaling

DVS and DFS allow the changing of processor voltage and frequency at runtime. Scheduling using DVS and DFS is to determine the voltage along with the frequency of a processor at runtime (Weiser et al., 1994; Govil et al., 1995). They are to support a both low-power and high-performance processor. In order to meet peak computational loads, the processor is operated at its maximum voltage and frequency. When the load is lower, the operating frequency can be reduced to at least meet the timing constraints.

CMOS circuit is currently used in almost every microprocessors. The average dynamic power which many researches in power reduction are concerned is given by the following formula:

$$P_{\text{dynamic}} = C_L * N_{SW} * V_{dd}^2 * f \tag{1}$$

where C_L is the load capacitance, N_{SW} is the average number of circuit switches per clock cycle, V_{dd} is the supply voltage and f is the clock frequency. Although power consumption can be reduced by the improvement in circuit technology (Burd and Brodersen,

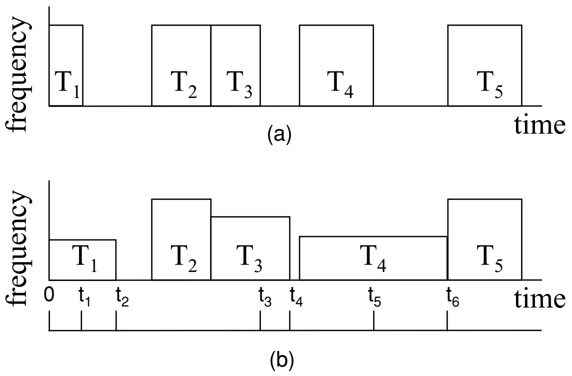

Fig. 2 Tasks running (a) without and (b) with DVS and DFS

1995; Chandrakasan et al., 1992), with the use of more complex designs, circuits have an increase in power dissipation. We can see also from the formula that energy consumed by the processor per clock cycle scales enormously with the operating voltage and frequency. Therefore, power-aware scheduling using DVS and DFS can have a significant impact on energy consumption.

Figure 2 shows that the idea of decreasing the processor frequency and supply voltage to reduce the power consumption. T_1, T_3 and T_4 are scheduled using lower frequencies which results in lengthening execution times in Figure 2(b). Finish times of T_1, T_3 and T_4 are extend from t_1, t_3 and t_5 to t_2, t_4 and t_6 respectively. Since lowering processor frequency results in lengthening task execution time, to lower the voltage or frequency, there should be enough idle time followed for the portion of lengthened execution time. If we can know efficiently where the idle time will be in the schedule, it helps the scheduler to decide which processor frequency would be the best for energy reduction. Supply voltage can then be adjusted accordingly.

2.2. Pinwheel algorithm and DCTS

In some real-time applications, tasks must be executed in a distance-constrained manner, rather than just periodically. That is, the temporal distance between any two consecutive executions of a task should always be less than a pre-defined value. Such a real-time system is called a Distance-Constrained Task System (DCTS). For example, video systems need to ensure the inter-arrival time between frame replays is always less than 33 ms. A video decoding task with period P_i and execution time e_i using a periodic model is shown in Figure 3. The distance of two consecutive jobs can be as long as $2P_i$ - e_i or as short as e_i. The distance in periodic task model is indeterministic and this degrades predictability of real-time systems. Assume the execution time of video decoding is 10 ms, the playing between two frames may be up to 56 ms. This is very different from receiving at least one frame in every 33 ms.

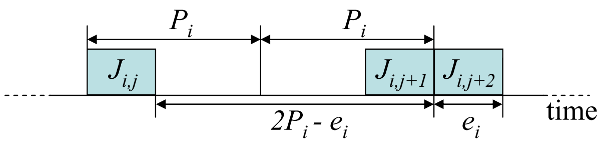

Fig. 3 Variation Distance between Consecutive Jobs of a Periodic Task

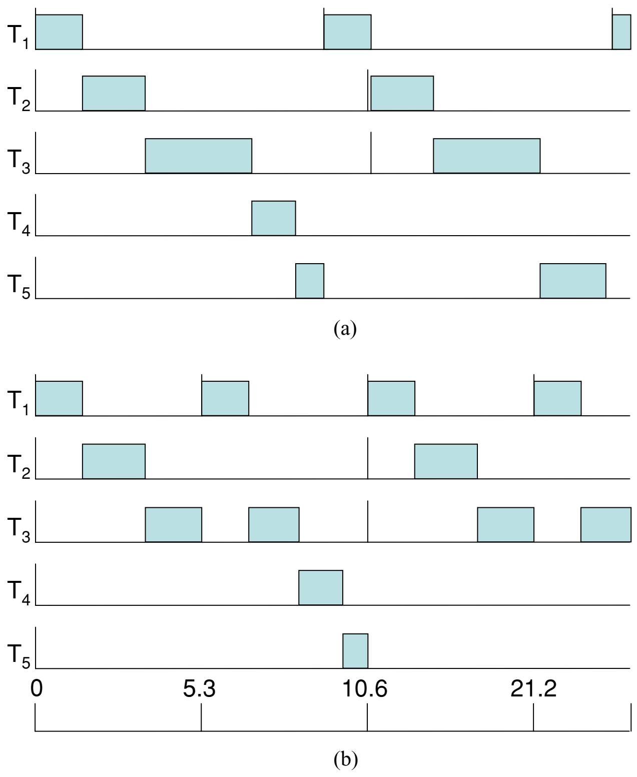

Fig. 4 The (a)RM and (b)Pinwheel schedule

Pinwheel scheduling uses a distance-constrained specialization technique to transform all task distance constraints to harmonic numbers (Hsueh and Lin, 2001; Hsueh and Lin, 1998). In a DCTS, it is necessary that the transformed distance constraints are less than or equal to the original distance constraints. Pinwheel schedulers transform the distance constraints into a set of special harmonic periods so that the density, sum of execution time divided by distance constraint of every task in the system, increase is minimized.

For example, suppose a system has five tasks with distance constraints $\{9.2, 10.6, 10.7, 21.4, 23.4\}$ respectively, and their execution times are $\{1.5, 2.0, 3.4, 1.4, 3.0\}$ respectively. After pinwheel transformation, the five tasks will have new distance constraints $\{5.3, 10.6, 10.6, 21.2, 21.2\}$ (Hsueh and Lin, 1998). These tasks then can be scheduled as periodic tasks using the new distance constraint as their periods. The RM[1] schedule using the original distance constraints as periods and the schedule of the task set after it is transformed by pinwheel algorithm is shown in Figure 4. Since the distance constraints are harmonic numbers, the execution schedule for each task has no jitter and meets the distance constraints. The system predictability, which is important in real-time systems, is increased, and thus complexity of power-aware real-time scheduling can be reduced.

[1] The most popular real-time scheduling approach.(Liu and Layland, 1973)

2.3. Related work

Recent researches focus on solving the challenge of online scheduling (Kim et al., 2003; Manzak and Chakrabarti, 2003). However, online scheduling is difficult, especially in priority-driven schedulers, due to the lack of task execution information. Shin and Choi proposed a Low Power Fixed Priority Scheduling (LPFPS) algorithm (Shin and Choi, 1999). Due to lack of workload information, simple technique is used. Processor voltage and frequency are changed only when there is one or no job ready for execution. Pillai and Shin presented several novel algorithms for realtime DVS that can achieve significant energy savings while simultaneously meeting real-time constraints (Pillai and Shin, 2001). The proposed cycle-conserving real-time DVS algorithms (ccRM, ccEDF) using dynamic schedulability tests to determine processor frequency. Although the algorithm is efficient, when a task is finished early, the slack time is only utilized by the remaining ready tasks for inter-task scheduling.

Daniel Mosse and et al. proposed power management hint (PMH) mechanism for compiler to provide runtime information to OS scheduler for better online scheduling (AbouGhazaleh et al., 2003; AbouGhazaleh et al., 2002; AbouGhazaleh et al., 2003). They also discussed the frequency of placing PMHs and rescheduling in order to reduce scheduling overhead (AbouGhazaleh et al., 2001). Ana Azevedo and et al. used a profile-based method for intra-task scheduling to adapt to the AET of a task (Azevedo et al., 2002). However, these works are either focused only on intra-task scheduling or lack of discussing about how OS scheduler uses these information to perform inter-task scheduling.

Our contribution is first introducing pinwheel model into power-aware real-time online scheduling to systematically utilize all slack times for energy saving and provide a computational feasible solution. In order to adapt to AET, a profiling tool is implemented to analyze a program and provide timing information as scheduling hints at runtime. The proposed scheduling algorithm can take advantages of these hints to dynamically adapt to system execution.

3. Power-aware algorithm using pinwheel model

In order to obtain better energy reduction, off-line scheduling is used in many systems. In off-line scheduling, knowledge of real-time jobs is known a priori, including periods, execution times, release times, etc. Complex algorithms are used in order to generate good schedules that can minimize energy dissipation. Although it can obtain better energy reduction but is considered inflexible. If the task set of the system changes, new schedule needs to be precomputed and stored for later use. Many on-line approaches are proposed for these more complicated and interactive systems. In this Section, we discuss the benefits obtained from pinwheel algorithm for online power-aware real-time scheduling.

3.1. Benefits obtained from pinwheel model

In opposition to offline scheduling, online scheduling makes each scheduling decision dynamically at necessary rescheduling points, such as when a task finishes its job, or a high-priority task (higher than the current one) is ready for execution. Therefore, online scheduling can accommodate dynamic variations in system availability and user demands. Online scheduling is very suitable for a system whose workload is unpredictable. However, the cost of flexibility and adaptability is that the scheduler cannot generate optimal schedule without prior knowl-

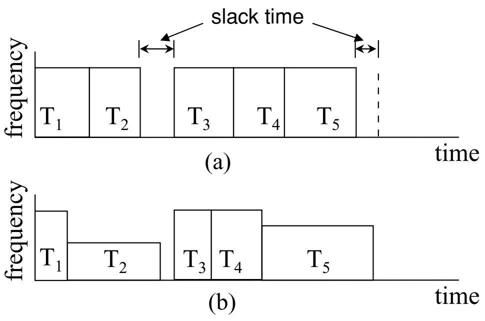

Fig. 5 The (a)RM and (b)LPFPS Schedule

edge. The benefits obtained from pinwheel model in power-aware real-time scheduling can be summarized as follows:

- Tasks information can be known a priori for better online and offline scheduling.
- Pinwheel schedule can be generated in polynomial time and space.
- The rescheduling points within the hyperperiod can be massively reduced.

The key idea of power-aware scheduling is to manage slack times in order to reduce energy dissipation. There may be still plenty of slack times in the system when tasks running in maximum processor frequency, especially when the system utilization is low. Under the restriction that no task misses its deadline, according to formula (1), we can lower the processor voltage and frequency to reduce energy consumption. However, there is no way for on-line scheduler to know slack times in advance so that only some simple heuristic algorithms can be used to partially solve the problem.

For example, LPFPS schedules using RM algorithm when there are more than one tasks in the ready queue. If there is only one task in the ready queue, the slack time until the next ready task can be derived. DVS and DFS are then applied if possible for energy saving. When there is no task in the scheduling queue, the system enters sleep mode to reduce energy dissipation. An example of LPFPS is shown in Figure 5(b), where T_1 and T_3 finish earlier then their WCETs, but only T_2 and T_5 execute using lower processor frequency to make use of the slack times. T_4 executes using the maximum frequency because it is not the only one task left in the ready queue.

As we can see, DVS and DFS are only conducted when there is less than or equal to one ready task in LPFPS. Further energy saving is still possible if DVS and DFS can be conducted before every task execution to more effectively manage slack times. Since lowering the processor frequency results in lengthening the execution time of the current executing job, it is necessary to make sure that no other job violates its timing constraints due to the change. However, this is difficult or time-consuming in priority-driven systems.

The pinwheel model can provide more timing information for online scheduling and helps to make better scheduling decisions. Start times, finish times, preemption times and resume times of real-time jobs within every period can be determined and have the same distance related to their release times. In other words, system predictability is increased. Schedulers can schedule based on not only information of the current task or system utilization but consider other tasks as well. It helps to reduce some difficulties of online power-aware real-time scheduling due to non-determinism.

Power-aware real-time scheduling using pinwheel model can be divided into two phases. In the first phase, we generate and store the pinwheel schedules. For a given real-time task set with task execution times and periods, we first perform pinwheel transformation using HSx(Hsueh and Lin, 1996). With the help of pinwheel algorithm, all task periods are transformed to harmonic integers. Real-time jobs in the generated pinwheel schedule are assumed to be executed using the maximum processor frequency and supply voltage by default.

In the second phase, schedulers can perform more precise power-aware scheduling according to their scheduling policies based on the timing information obtained from the pinwheel schedule and then dynamically perform DVS and DFS to reduce energy consumption on every rescheduling point. This is difficult or time-consuming in priority-driven systems due to lack of information of other real-time tasks. Furthermore, rescheduling points within a hyperperiod of a schedule are reduced such that the number of context switch and scaling of processor voltage and frequency is massively reduced, which leads to better energy reduction.

3.2. Greedy method

LPFPS is simple and easy to implement in most priority-driven real-time systems. However, the chance of changing processor frequency is low and cannot fully explore the potential of energy savings. We adopt pinwheel algorithm to solve this problem. Pinwheel algorithm transforms periods of real-time tasks into harmonic numbers and a pinwheel schedule can be generated in polynomial time and space for providing further power-aware scheduling information. The scheduling interval of time t is the time between t and the end of the next idle time in the generated pinwheel schedule (Figure 1). LPFPS can be extended to perform DVS and DFS at every rescheduling point instead of at the time when executing the last task in the ready queue. We call the extended version of LPFPS greedy method and the idea of the algorithm is as follows. Whenever rescheduling is required, we allow the next ready job to try to greedily use up the idle time within this scheduling interval so that the processor frequency is lowered as much as possible.

Figure 6 shows an example schedule of three tasks T_1, T_2, and T_3 where their execution times are 9, 6, 6 and deadlines are 31.5, 31.5 and 63 respectively. The scheduling results of the pinwheel and the greedy algorithm are shown in Figure 6(a) and Figure 6(b) respectively. The greedy method reschedules at every rescheduling point to determine processor voltage and frequency. Although it is efficient and easy, the energy reduction may depend on the given real-time task set due to the processor frequency steps are discrete. As the example shown in Figure 6(b), there is still some idle time at time 30 and no other jobs can make use of it. This is because the remainder of the system slack time is not enough for adjustments of

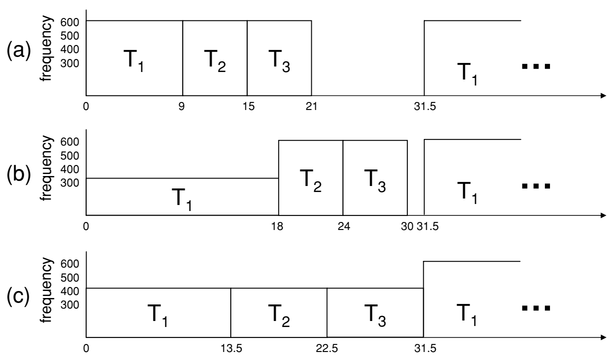

Fig. 6 The difference between (a) Pinwheel, (b) Greedy Method and (c) LP Method

other jobs. However, it is still possible for further energy reduction if the slack time between time 21 and 31.5 can be fully utilized.

3.3. LP method

Since lowering processor frequency leads to longer execution time, energy savings can be achieved by exploiting the slack time such that it is as little as possible. Due to the available processor frequencies are not continuous, the problem can be mapped into an Integer Linear Programming (ILP) problem to find an optimal solution such that remaining slack time is minimized. However, ILP problem is NP-complete and can not be solved in polynomial time. Our heuristic algorithm uses Linear Programming (LP) to approximate the optimal solution. In a given schedule, LP method is applied at the beginning of every scheduling interval to find a feasible solution which guarantees timing constraints of tasks and utilizing slack time within the scheduling interval as much as possible. Task execution, processor frequency and supply voltage scaling are changed accordingly. Finding a feasible solution which satisfies all the constraints to a LP problem can be solved in polynomial time (Cormen et al., 2001). General speaking, the time to solve a LP problem may take up to few milliseconds or even more, which is considered too long for a real-time scheduler. However, since the number of real-time tasks is limited, the overhead is acceptable as we will show in the next section.

In LP method, we utilize slack time more aggressively. For a given processor contains m frequency steps q_1 (minimum frequency), $q_2, \ldots, q_m$ (maximum frequency) and a given scheduling interval with length T. There are n jobs $J_1, J_2, \ldots, J_n$ with deadlines $D_1, D_2, \ldots, D_n$, execution times $C_1, C_2, ..., C_n$ and start times $t_1, t_2, \ldots, t_n$ respectively in the given interval. Every job has a period equal to its deadline. Assume that the given task execution time is the worst case execution time when executing using the maximum frequency and the length of task execution time has a linear relationship with the reverse of processor frequency. Therefore, for each task, execution time C_i using the j-th frequency, C_{ij}, is defined as

$$C_{ij} = \frac{q_m}{q_j} * C_i \text{ for } i = 1, 2, \ldots, m. \tag{2}$$

We can utilize slack time by reducing the speed of the processor, i.e. lowering the processor frequency along with lowering the supply voltage. We determine processor frequency of each job in the scheduling interval to utilize slack time as much as possible.

Let $X_i = \frac{q_m}{q_i}$, which approximates the ratio of lengthened task execution time when lowering processor frequency from q_m to q_i. We wish to find n real numbers $X_1, X_2, \ldots, X_n$ such that the sum of execution times within the scheduling interval ($\sum_{i=1}^{n} C_i * X_i$) is maximum. In other words, LP method tries to fully utilize slack times. The LP formulation for the LP method based on fully utilizing slack times for energy optimization is as follows.

Maximize

$$\sum_{i=1}^{n} C_i * X_i \tag{3}$$

subject to

$$X_i \geq 1 \qquad \text{for } i = 1, 2, \ldots, n \tag{4}$$

$$X_i \leq \frac{q_m}{q_1} \qquad \text{for } i = 1, 2, \ldots, n \tag{5}$$

$$C_i * X_i \leq D_i - t_i \quad \text{for } i = 1, 2, \ldots, n \tag{6}$$

$$\sum_{i=1}^{n} C_i * X_i \leq T \quad \text{for } i = 1, 2, \ldots, n. \tag{7}$$

Constraint (4) restricts the maximum available frequency to q_m and constraint (5) restricts the minimum available frequency to q_1. Constraint (6) makes sure that the execution time of every job is less or equal to its period so that the system will not utilize more slack time than it has. In other words, every job in the scheduling interval will not miss its deadline. The last constraint ensures that the sum of execution times within the scheduling interval will not exceed the length of the interval. The algorithm determines processor frequency of job J_i in the scheduling interval according to X_i. The i-th frequency derived from the processor frequency closest to but less than X_i is selected.

Using the example illustrates in Figure 6(c), LP method can obtain further energy reduction than the greedy method. Since the frequency of a processor can not be continuously adjustable, LP method may not always make full use of whole the slack time. However, with the extra timing information provided by pinwheel schedule, the LP method is the first systematic method to utilize all slack times for energy saving.

3.4. Adaptive from WCET to AET

When a program is executing, it may not always use up to its WCET. Therefore, the schedule generated by a scheduling algorithm using WCET may not be able to optimize the energy consumption and may even be a bad schedule. In order to solve this problem, we implement a profiling tool which can derive time information and pass it to scheduler at runtime. The tool analyzes the call flow graph (CFG) and inserts codes in the binary file of a program. The inserted codes issue a pre-defined rescheduling system call to update the current WCET. AET can be obtained by using the updated WCET. The system will then reschedule according to the new WCET to adapt the AET.

Since scheduling using LP method is relatively more costly, it would not be a good idea to perform LP scheduling at every rescheduling point. We can use a hybrid method that schedules using LP method to schedule once every scheduling interval. For applications that usually do not use up to their WCET, further energy saving is possible. When the rescheduling system call is issued, we can then schedule using greedy method to further utilize slack times at runtime. Therefore, the proposed approach can maximize system energy reduction while minimizing the scheduling overhead.

3.5. Issue and solution of applying pinwheel model

Although there are many advantages applying pinwheel model for power-aware real-time scheduling. However, when periods of real-time tasks are transformed into harmonic numbers using pinwheel algorithm, new periods will be equal to or shorter than the original ones. The job execution may become too frequent and thus the behavior of the tasks is changed. Such changes in periods might not be acceptable in some applications. For the example illustrated in Figure 7(a), in a video system, one frame is decoded and replayed every 33 ms to play back a video clip smoothly. After pinwheel transformation, the video system may decode and replay a frame every 22 ms as shown in Figure 7(b). The task executes more time than it is supposed to, such as in the first and the third of the original period. The video clip would look like playing fast-forward and is not acceptable.

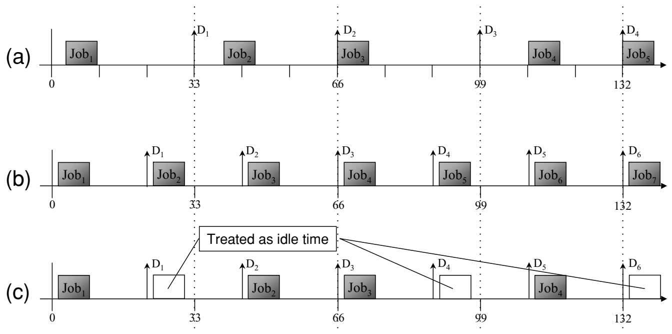

Fig. 7 Issue and Solution of applying Pinwheel Model

In (Yu et al., 2004), we proposed and proved the feasibility of preserving the original job execution frequency in a pinwheel schedule. The execution time of a task in a pinwheel schedule within any original period is larger than or equal to the original execution time. As shown in Figure 7(c), if, in some systems, changing periods is not acceptable in one or many real-time tasks, we can make a task idle in order to prevent it from being executing too frequently. Again, the idle times can be exploited as slack time for further energy saving.

4. Simulation results

In this section, we evaluate the benefits of power-aware real-time algorithms using pinwheel model on energy reduction. The results are simulated using the hardware specification based on Transmeta processors (http://www.transmeta.com/index.html, 2004) in order to obtain more practical results. The simulations are performed on system utilization between 10% to 70% with total 140000 test sets. The number of real-time tasks of a test set is between 2 to 8. We compare the results of greedy and LP method using pinwheel model with RM, ccRM and LPFPS on all test sets.

We perform simulations using Transmeta TM55EL-667 and TM58EX-933 specification. The frequencies, corresponding voltages and maximum power consumption of the processors are from the processor data books and are listed in Table 1. We take the energy consumption using RM scheduling as base and evaluate the energy reduction of these power-aware real-time algorithms. The normalized energy reduction is shown in Figure 8 and Figure 9. With

Table 1 Power Specification

Frequency	Voltage	TM55EL-667	TM58EX-933
Sleep	0.8 V	0.35 W	0.35 W
300 MHz	0.8 V	1.7 W	1.7 W
400 MHz	0.9 V	2.6 W	2.6 W
500 MHz	0.975 V	3.3 W	3.3 W
600 MHz	1.05 V	4.3 W	4.3 W
700 MHz	1.15 V	N/A	5.6 W
800 MHz	1.2 V	N/A	6.8 W
900 MHz	1.3 V	N/A	8.8 W

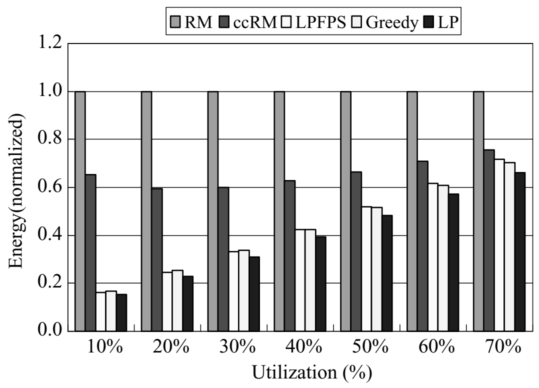

Fig. 8 Energy Reduction on TM58EL-667

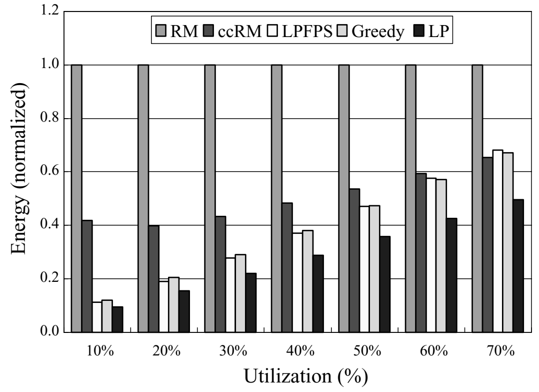

Fig. 9 Energy Reduction on TM58EX-933

the increasing of system utilization, system slack time decreases. Certainly, it would be more difficult to obtain energy saving with high system load, since most jobs need to be executed at the highest frequency to avoid violation of their timing constraints. Although greedy method gains considerable energy saving, LP method obtains the best results among all. LP method reduces an average of 37% and 56% of energy on TM55EL-667 and TM58EX-933 respectively at 70% utilization. Energy is further reduced on TM58EX-933 than on TM55EL-667. This is because when there are more frequency steps for a scheduler to select, it will be more flexible to adjust CPU frequencies for real-time tasks and can thus further utilize system slack time.

ccRM determines processor frequency by compute the utilization using the actual computing time consumed by tasks. The selected frequency guarantees that all tasks will meet their deadlines. However, due to the processor frequency is not continuously adjustable and lacks of timing information of tasks, such as release times, preemption times, resume times, etc, ccRM may not fully utilize the slack time using the selected frequency. On the other hand, with the sufficient information obtained from pinwheel algorithm, greedy and LP method are able to make better scheduling decisions.

Figure 10 shows scheduling overhead of each algorithm. RM, LPFPS, ccRM and Greedy method consume very little time to schedule and the overhead remains constant among different system utilization. On the other hand, LP method needs more time to schedule. While the scheduling time increases along with the increasing of the number of real-time tasks, there is an average scheduling overhead of 53 μs using the LP method. Directly using

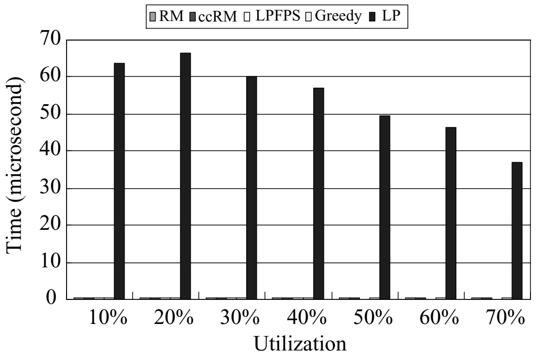

Fig. 10 Scheduling Overhead

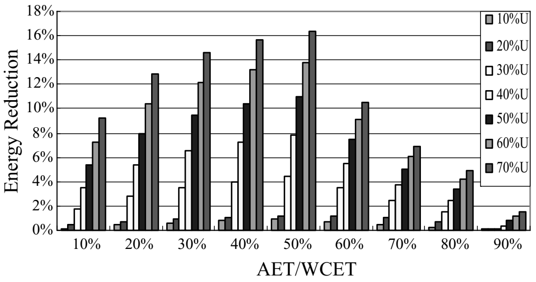

Fig. 11 Further Energy Reduction at Runtime on TM58EL-667

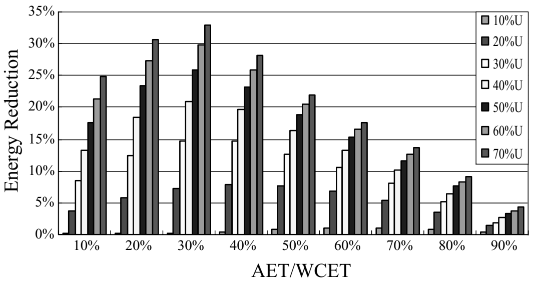

Fig. 12 Further Energy Reduction at Runtime on TM58EX-933

linear programming to determine processor frequency in a schedule leads to large scheduling overhead (few milliseconds or more) which is hardly accepted in real-time systems. Due to the nature of harmonic numbers, a smaller hyperperiod is obtained. The number of rescheduling points is thus reduced. If the system can not accept this online overhead, we could add hardware to speed up Linear Programming in the system. Or at least, it is easy to see the overall overhead of LP method using pinwheel model is massively reduced due to much less rescheduling points.

As shown in Figure 11 and Figure 12, by the help of the profiling tool, more energy can be saved according to task execution at runtime. Instead of idling the unused time, the proposed scheduler reschedules when the codes inserted by the profiling tool are executed. When AET is only a small portion of the WCET, there is some additional energy saving. This is because system utilization is low so that processor is idled at most of the time. As the ratio of AET/WCET is increased, there is great opportunity of further adjustment and thus additional energy saving can be considerable. There is at most 33% of additional energy saving when AET/WCET is 50% at 70% utilization. The average energy saving is 17.85%.

If AET is very close to the WCET, there is few unused time for online schedule adjustment so energy saving is not obvious.

5. Conclusion

In this paper, we discuss and analyze applying pinwheel model to power-aware real-time scheduling. We show that the harmonic nature of pinwheel model can be applied to many power-aware real-time scheduling and thus benefits from the pinwheel model for deterministic task execution. Systems are under better control due to the increasing of predictability. Scheduling and space complexity can be decreased to build fast online schedulers and various techniques can be used to fully utilized whole system slack times. Simulation shows that power-aware real-time scheduling using pinwheel model achieves considerable energy reduction and their scheduling overhead is manageable. We also implement a profiling tool to analyze a program and insert codes to provide runtime execution information for better power-aware real-time scheduling. We believe pinwheel model provides a systematic approach and a computational feasible solution to fully utilize the system slack times so as to minimize energy consumption.

Acknowledgements Supported in part by research grants from the ROC National Science Council under Grants NSC-93-2220-E-194-003 and NSC-93-2220-E-194-010

References

"Intel internet homepage. http://www.intel.com/index.htm," 2004
"Amd internet homepage. http://www.amd.com/us-en/," 2004
"Transmeta internet homepage. http://www.transmeta.com/index.html," 2004
Unsal, O. S. and Koren, I. (2003) System-level power-aware design techniques in real-time systems. Proceedings of the IEEE 91(7), pp. 1055–1069
AbouGhazaleh, N., Mosse, D., Childers, B., Melhem, R., and Craven, M (2003) Collaborative operating system and compiler power management for real-time applications, In IEEE Real-Time Embedded Technology and Applications Symposium (RTAS)
Chen, K., and Muhlethaler, P (1996) A scheduling algorithm for tasks described by time value function. Journal of Real-Time Systems, 10(3), pp. 293–312
Liu, J. W (2000) *Real-Time Systems*. Prentice Hall
Ernst, R. and Ye, W (1997) Embedded program timing analysis based on path clustering and architecture classification. In DIEEE/ACM International Conference on Computer-Aided Design pp. 598–604
Hsueh, C.W and Lin, K.-J (2001) Scheduling real-time systems with end-to-end timing constraints using the distributed pinwheel model. In IEEE Transactions on Computers (SCI) 50(1), http://www.cs.ccu.edu.tw/ chsueh/papers/DSr.ps
Weiser, M., Welch, B., Demers, A. J., and Shenker, S (1994) Scheduling for reduced CPU energy. In Operating Systems Design and Implementation pp. 13–23
Govil, K., Chan, E., and Wasserman, H (1995) Comparing algorithm for dynamic speed-setting of a low-power CPU. In ACM International Conference on Mobile Computing and Networking pp. 13–25
Burd, T.D. and Brodersen, R. W (1995) Energy efficient CMOS microprocessor design. In Hawaii Intel Conference System Sciences pp. 288–297
Chandrakasan, A., Sheng, S., and Brodersen, R. (1992) Low-power cmos digital design. IEEE Journal of Solid-State Circuit 27(4): 473–484
Hsueh, C., and Lin, K.-J. (1998) On-line schedulers for pinwheel tasks using the time-driven approach. In 10th EUROMICRO workshop on Real TIme Systems, Berlin Germany
Liu, C. L. and Layland, J. (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of the ACM 10 (1): 46–61

Kim, W., Kim, J., and Min, S. L. (2003) A dynamic voltage scaling algorithm for fixed-priority hard real-time systems using slack time analysis. In Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED) pp. 396–401

Manzak, A. and Chakrabarti, C. (2003) Variable voltage task scheduling algorithms for minimizing energy/power IEEE Transactions on Very Large Scale Integration Systems 11 (2): 270–276

Shin, Y. and Choi, K. (1999) Power conscious fixed priority scheduling for hard real-time systems. In Design Automation Conference pp. 134–139

Pillai, P. and Shin, K. G. (2001) Real-time dynamic voltage scaling for low-power embedded operating systems. In ACM Symposium on Operating Systems Principles, pp. 89–102

AbouGhazaleh, N., Mosse, D., Childers, B., and Melhem, R. (2002) Compilers and Operating Systems for Low Power. Kluwer Academic Publishers

AbouGhazaleh, N., Childers, B., Mosse, D., Melhem, R., and Craven, M. (2003) Energy management for real-time embedded applications with compiler support. In ACM SIGPLAN Languages, Compilers,and Tools for Embedded Systems

AbouGhazaleh, N., Mosse, D., Childers, B., and Melhem, R., (2001) Toward the placement of power management points in real time applications. In Workshop on Compilers and Operating Systems for Low Power

Azevedo, A., Issenin, I., and Cornea, R. (2002) Profile-based dynamic voltage scheduling using program checkpoints. In Conference on Design, Automation and Test in Europe

Hsueh, C. and Lin, K.-J. (1996) An optimal pinwheel scheduler using the single-number reduction technique. In Proc. IEEE Real-Time Systems Symposium, Washington, DC, pp. 196–205

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001) Introduction To Algorithms, Second Edition. Cambridge, Massachusetts London, England: The Mit Press

Yu, S. hung, Lin, H. hung, and Hsueh, C. wen (2004) An application using pinwheel scheduling model. In The Tenth International Conference on Parallel and Distributed Systems