Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 428230, 9 pages
http://dx.doi.org/10.1155/2014/428230

Research Article

A Blocking-Aware Scheduling for Real-Time Task Synchronization Using a Leakage-Controlled Method

Mu-Yen Chen,[1] Da-Ren Chen,[1] and Shu-Ming Hsieh[2]

[1] *Department of Information Management, National Taichung University of Science and Technology, Taichung City 404, Taiwan*
[2] *Department of Computer Science and Information Engineering, Hwa Hsia Institute of Technology, New Taipei City 235, Taiwan*

Correspondence should be addressed to Da-Ren Chen; danny@nutc.edu.tw

Received 3 October 2013; Accepted 18 December 2013; Published 13 February 2014

Academic Editor: Hwa-Young Jeong

Copyright © 2014 Mu-Yen Chen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Due to the importance of power dissipation in the wireless sensor networks and embedded systems, real-time scheduling has been studied in terms of various optimization problems. Real-time tasks that synchronize to enforce mutually exclusive access to the shared resources could be blocked by lower priority tasks. While dynamic voltage scaling (DVS) is known to reduce dynamic power consumption, it causes increased blocking time due to lower priority tasks that prolong the interval over which a computation is carried out. Additionally, processor slowdown to increase execution time implies greater leakage energy consumption. In this paper, a leakage-controlled method is proposed, which decreases both priority inversion and power consumption. Based on priority ceiling protocol (PCP) and a graph reduction technique, this method can decrease more energy consumption and avoid priority inversion for real-time tasks.

1. Introduction

Power consumption of IC devices can be briefly classified into (1) *static* power consumption which consists of the power used when the transistor is not in the switching process and (2) *dynamic* power consumption which consists of the power used to charge the load capacitance and logic states of the device. In the last two decades, complementary metal-oxide semiconductor (CMOS) has emerged as a dominant technology due to its low static power dissipation. Leakage power in CMOS circuits is due to subthreshold current that flows through the transistors and contributes to a significant portion of the total power consumption. Figure 1 illustrates the trend at lower nodes that leakage power becomes almost comparable to active power. For 90 nm and below technologies, leakage is the main factor which dominates over the dynamic power and contributes to almost or more than 50% of total power dissipation.

There have been multiple techniques used in the past to reduce the power dissipation of embedded systems and have been implemented successfully through the different levels of design abstraction. At the system level, there are two primary ways to reduce power consumption: processor slowdown and shutdown. Slowdown techniques such as dynamic voltage scaling (DVS) are preferred due to the assumptions of low leakage power dissipation and quadratic dependence of power on voltage level. However, energy savings based on DVS come at the cost of increased execution time, which implies greater leakage power consumption. With the repaid increase in device leakage power due to the scaling down of technology nodes, the leakage power has been a major concern in the system design. Device scheduling is the key factor to have the best tradeoff between the power and characteristics such as resource sharing and system utilization.

Many previous works have addressed DVS based on performance requirements to minimize the dynamic power consumption [1–13]. While techniques to optimize the total static and dynamic power consumption have been proposed, they are still based on the ideal system environment without considering resource synchronization problem. Recently, researchers have started exploring energy-efficient scheduling with the considerations of leakage current [4–6, 14–16]. To reduce the leakage power, a system might be

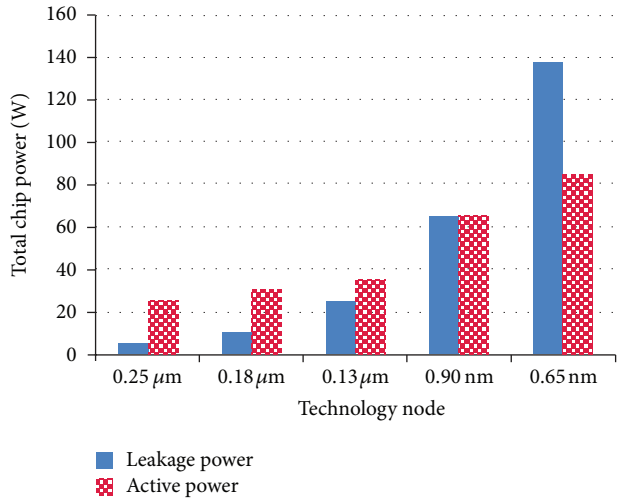


Figure 1: Leakage versus dynamic power trend [1, 19, 20].

turned off (to enter a dormant mode) when not used. For periodic real-time tasks, Jejurikar et al. [6] and Lee et al. [14] proposed energy-efficient scheduling on a uniprocessor by procrastination scheduling to decide when to turn off the system. Jejurikar and Gupta [5, 15] further considered real-time tasks that might complete earlier than its worst-case execution time (WCET) by extending the algorithms presented in [6]. For DVS processor with aperiodic real-time tasks, Irani et al. [4] proposed a 3-approximation algorithm to minimize leakage current on the uniprocessor with continuous available speeds. It guarantees to derive a solution with the energy consumption at most three times energy consumption of an optimal solution. For aperiodic real-time tasks without DVS, Baptiste [17] developed an algorithm that minimized the energy consumption resulting from the leakage current and the overhead to turn on/off the system. In the periodic real-time task systems, Quan et al. [13] and Niu and Quan [16] proposed a leakage-aware method that expands all the jobs in the hyperperiod of the given tasks and calculates their latest starting time on the fly. The computation overheads produced by their algorithms might be too much to be applied in an online scheduling. In addition to considering the leakage current of the processor, leakage power consumption produced by system devices was also considered in [18] as well as their preemption control [9]. The objectives of the above methods are proposed to reduce the system power and do not consider other advantages or purposes of leakage-aware techniques. For example, one can shut down a processor before the beginning of a lower priority task that intents to lock a shared resource required earlier by a higher priority task.

Work-demanded analysis has been extensively investigated in real-time systems in which sporadic tasks are jointly scheduled with periodic tasks [3, 7–9, 11, 12, 15, 21]. The purpose of work-demanded analysis is to improve the response times of aperiodic tasks or to increase their acceptance ratio. It also benefits the power-saving on both DVS and leakage-aware methods. Pillai and Shin [12] proposed a cycle-conserving rate-monotonic (ccRM) scheduling scheme that contains offline and online algorithms. The offline algorithm computes the worst-case response time of each task and derives the maximum speed needed to meet all task deadlines. It recomputes the task utilization by comparing the actual time for completed tasks with WCET schedule. In other words, when a task completes early, they have to compare the used actual processor cycles to a precomputed worst-case execution time schedule. This WCET schedule is also called canonical schedule [22] whose length could be the least common multiplier of task periods. ccRM is a conservative method, as it only considers possible slack time before the next task arrival (NTA) of current job. Gruian proposed a DVS method for offline task stretching and online slack distribution [3]. The offline part of this method consists of two separate techniques. One focuses on the intratask stochastic voltage scheduling that employs a task-execution length probability function. The second technique computes stretching factors by using a response time analysis. It is similar to Pillar and Shin's offline technique, but instead of adopting a stretching factor for all tasks before NTA, Gruian assigns different stretching factor to the individual task within the longest task period. Kim et al. [8] proposed a greedy online algorithm called the low-power work-demand analysis (lpWDA) that derives slack from low-priority tasks, as opposed to the method in [3, 12] that gains slack time from high-priority tasks. This algorithm also balances the gap in voltage levels between high-priority and low-priority tasks. Its analysis interval limited by the longest of task periods is longer than NTA. Thus, lpWDA gains more energy saving than the previous rate-monotonic (RM) DVS schemes applying NTA. Therefore, these slack reclamation methods are applicable not only to increase energy-saving and schedulability but also to the new methods on different scheduling criteria. For example, slack time can also be applied to avoid priority inversion incurred by task synchronization.

Though power-aware real-time scheduling problems are well studied, relatively few works of them considered task synchronization problems. Most embedded real-time applications have to share the resources such as I/O devices or database and mutually exclusive access to these shared objects. Concurrent access to shared data may result in data inconsistency. Mechanisms that force tasks to execute in a certain order so that data consistency is maintained are known as synchronization protocols. If a lower priority job has access to a resource, a higher priority job requesting the resource is blocked and can miss its deadline. Therefore, real-time task synchronization is developed to achieve data consistency with as little loss in schedulability as possible. Priority ceiling protocol (PCP) [23, 24] is a synchronization protocol for shared resources to avoid deadlock and unbounded priority inversion due to wrong nesting of critical sections. In PCP, each resource is assigned a priority ceiling, which is a priority equal to the highest priority of any task which may lock the resource. PCP has the interesting and very useful property that no task can be blocked for longer than the duration of the longest critical section of any lower-priority

task. The recent related work is an extension of PCP in frequency inheritance. Zang and Chanson [25] proposed a dual-speed (DS) algorithm: one is for the execution of a task when it is not blocked, and the other is adopted to execute the task in the critical section when it is blocked. Jejurikar and Gupta [26] compute two slowdown factors, which can be classified into static slowdown, computed offline based on task properties, and dynamic slowdown, computed using online task execution information. Chen and Kuo [2] proposed a DVS method using frequency locking concept which can be used to render energy-efficient to the existing real-time task synchronization protocols. However, DVS may decelerate the blocking tasks in the critical section and receive additional priority inversion which increases the difficulties of predictability of the schedule.

In this paper, we propose a blocking-aware task synchronization method, which decreases both priority inversion and energy-consumption of the processor. Graph theories have been applied in the scheduling which transforms an existing resource allocation graph (bipartite graph) to a weighted-directed graph (WDG). Our idea is based on PCP and available slack time in the schedule. We use WDG to analyze and present the available slack time, preemption points, and the lengths of priority inversions. The timing and its duration for switching processor status can be computed by traversing the WDG in which lower-priority tasks have intention to lock on some resources. The proposed method can utilize the existing work-demanded analysis methods such as lpWDA [8] and Gruian's methods [3] to squeeze additional slack time from lower priority tasks so as to further decrease priority inversion.

The remainder of this paper is organized as follows. Section 2 explains our system model. The motivations and basic idea on the task synchronization method are proposed in Section 3. Section 4 describes the proposed algorithm and its example. We present the performance evaluation in Section 5. Section 6 gives conclusions and their future work.

2. System Model

2.1. Task Model. This paper studies periodic real-time tasks that are independent during the runtime. Let $\mathcal{T}$ be the set of input periodic tasks and n denotes the number of tasks. Each task τ_i is an infinite sequence of task instances, referred to as *jobs*. The jth invocation (job) of task τ_i is denoted as $J_{i,j}$ whose actual start time is denoted as $S(J_{i,j})$. A three-tuple $\tau_i = \{T_i, D_i, C_i\}$ represents each task, where T_i is the period of the task, D_i is the relative deadline with $D_i = T_i$, and C_i denotes the worst-case execution time (WCET). The period length of task T_i is unique in order to have each task a unique priority index in the rate-monotonic (RM) scheduling. Notation $s_{i,j}$ denotes the available slack time for job $J_{i,j}$.

2.2. Power Model. All tasks are scheduled on a single processor which supports two modes: dormant mode and active mode. When the processor is switched to the dormant mode, the power-consumption of the processor is assumed $S_{\text{dorm}} = 0$ by scaling the static power consumption [27], while the system clock and chipset retain necessary functions to support motoring and waking up processor properly. When executing the jobs, the processor has to be in the active mode with speed S_{active}. A processor shutdown and wakeup have a higher overhead than the inherent energy/delay cost of turning on the processor. The processor loses the cache and register contents, when it switched to the deepest dormant mode. Therefore, prior to shutdown, all statuses of processor must be saved or flushed to main memory, resulting in an additional overhead. On wakeup, components such as data and instruction caches, data and translation look aside buffers, (TLBs) have to be initialized. This results in extra memory accesses and hence additional energy consumption. Let E_{sw} and t_{sw} denote energy and time overhead, respectively, for switching from dormant mode to active mode. Based on the idle power consumption, when processor is idle in the active mode, the processor executes NOP instruction at processor speed S_{idle} for low-power consumption referred to as $P(S_{\text{idle}})$. When processor is idle and the idle interval is longer than break-even time $E_{\text{sw}}/P(S_{\text{idle}})$, turning it to the dormant mode is worthwhile.

2.3. Resource Access Model. We assume that semaphores are used for task synchronization. All tasks are assumed to be preemptive, while the access to the nonpreemptive shared resources must be serialized. Therefore, a task can be *blocked* by lower priority tasks. When a task has been granted access to a shared resource, it is said to be executing in its critical section [24]. The kth critical section of task τ_i is denoted as $z_{i,k}$ which is properly nested. Each task knows the relative start time to access the required shared resources and their required WCETs. With the given information in [28], we can compute the maximum blocking time for a task. In the different resource synchronization protocol such as PCP [23], each job might suffer from a different amount of blocking time from lower-priority task due to access conflict. The goal of this paper is to propose a leakage-aware task synchronization protocol based on PCP, which reduces the priority inversion and energy-consumption of the processor. We propose a data structure called weighted directed graph (WDG) which expresses possible priority inversion online. By traversing WDG, we can postpone the intention to locking on the resources invoked by lower-priority tasks and therefore minimize the priority inversions.

3. Motivating Example

Suppose that we have three jobs J_1, J_2, and J_3 and three shared data structures protected by the binary semaphores z_0, z_1, and z_2 in the system. In accordance with PCP, the sequence of events is depicted in Figure 2(a). A line at a lower level indicates that the corresponding job is blocked or preempted by a higher-priority job while the processor mode is active. A line raised to a higher level denotes that the job is executing, and the absence of a line denotes that the job has not yet been initiated or has completed. A bold line at low level denotes that the processor has been switched in dormant mode. Suppose the following.

(i) At time t_0, J_3 is initiated and locks semaphore z_2 at t_0'.

(ii) At time t_1, J_2 is initiated and preempts J_3.

(iii) At time t_2, J_2 cannot lock z_2 and J_3 inherits the priority of job J_2 and resumes execution.

(iv) At time t_3, J_3 successfully enters its nested critical section z_1.

(v) At time t_4, J_1 preempts J_3 within z_1 and executes its noncritical section code.

(vi) At time t_5, J_1 attempts to enter its critical section z_0 and is blocked by J_3 due to priority ceiling.

(vii) At time t_6, J_3 exits its critical section z_1 and J_1 is awakened.

(viii) At time t_7, J_1 completes its execution and J_3 resumes its execution of z_2 due to its inherited priority.

(ix) At time t_8, J_3 exits its critical section z_2 and returns to its original priority, and J_2 is awakened.

The blocking intervals introduced by primitive PCP are $[t_2, t_4]$, $[t_5, t_6]$, and $[t_7, t_8]$.

This study is motivated by the significant priority inversion that is incurred from task synchronization. When the available static slack time (unused time in the WCET schedule) or dynamic slack (occurred in the early-completed task) is larger than break-even time, the lower-priority task intent to lock a semaphore can be postponed until the start time of a higher-priority task. A practical approach is to postpone the task execution by switching processor to dormant mode. During the sleeping time, system still has awareness of the arrival of other jobs and awakes processor at proper time. The example in Figure 2(b) postpones the request of lower-priority task intent to lock a semaphore. At time t_1, J_3 has available slack in interval $[t_{11}, t_{12}]$ with length longer than break-even time. When a system is conscious that J_3 has intent to lock z_2, it computes the upcoming start time of higher priority tasks that might be blocked by J_3 according to PCP. In the example, only J_2 could be blocked by J_3 due to z_1 and z_2, and the length of interval $[t_1, t_2]$ is less than the available slack, that is, $[t_{10}, t_{11}]$ in Figure 2(a). Therefore, processor can be switched to dormant mode at time t_1 until the start time of J_2. At time t_2, processor becomes *active* and J_2 preempts J_3 such that J_3 is still unable to lock z_2, and thus J_2 could lock z_2 at time t_3. Additionally, the J_2's noncritical section also is not blocked by J_3 because J_3 cannot inherit the priority of J_2 due to failure in locking z_2 at t_1. Comparing to the result of Figure 2(a), this idea can reduce (maximum) priority inversion of intervals $[t_2, t_4]$, $[t_5, t_6]$, and $[t_7, t_8]$. Table 1 presents the length of blocking time in a systematic way [28]. It lists only the nonzero elements; all the other elements are zero, since jobs are not blocked by higher priority jobs. The elements labeled with "$*$" in this table are zero as well.

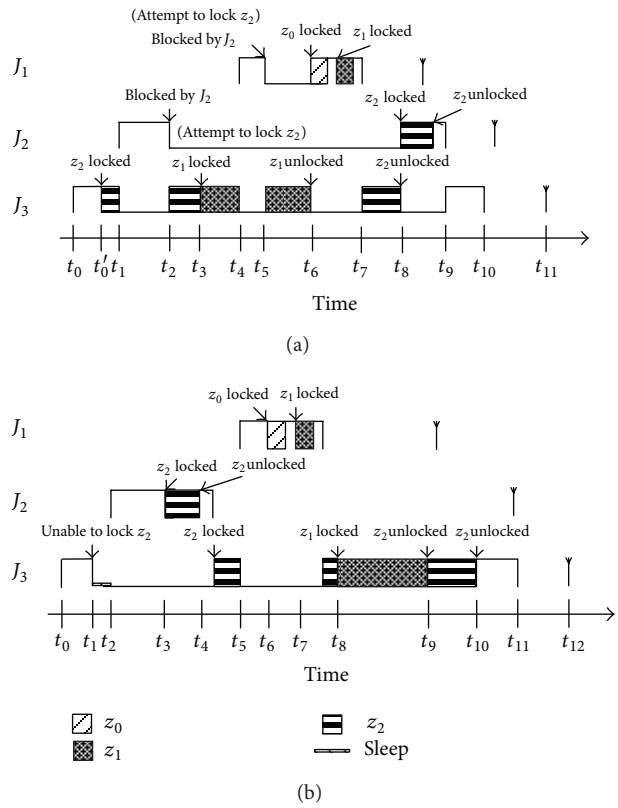


Figure 2: The task synchronization of (a) primitive PCP and (b) latency locking method.

4. Latency Locking PCP

In this study, PCP is extended with the concept of latency locking, referred to as LL-PCP. The idea is to perform preanalysis of possible priority inversion and available slack time in the schedule. LL-PCP derives the best timing and duration for switching processor to dormant mode and thus reduces priority inversion. To understand and control the sequence of intent to lock resources, tasks are organized as a weighted directed graph (WDG) reduced from the resource allocation bipartite graph in [28]. In the bipartite graph, each indirect edge is labeled with the time required to access the resources. Let $G = (U, V, E)$ denote a bipartite graph whose partition of vertices has two subsets U and $V \cdot E$ denotes the set of edges of G, and U denotes a task set $\mathcal{T}$. Let WDG($\mathcal{T}$, A) denote a weighted directed graph whose vertices in $\mathcal{T} \subseteq U$ are arranged according to their task indices. For each edge $e_{u,v} \in E$, $\tau_u \in U$, and $z_v \in V$, the set of arcs A in WDG are generated as follows.

Step 1. For any pair of vertices $\tau_x, \tau_y \in U$, and $x > y$, a solid arc $a(x, y) \in A$ is directed from τ_x to τ_y if there exist two or more edges $e_{x,v}$ and $e_{y,v}$ in G, where $z_v \in V$.

Step 2. For any pair of vertices $\tau_x, \tau_w \in U$, and $x > w$, a dotted arc $a(x, w) \in A$ is directed from τ_x to τ_w if there exists a vertex $\tau_y \in U$, $w > y$, and τ_x and τ_y satisfy Step 1.

Step 3. In WDG, for any pair of vertices with multiple arcs, eliminate the dotted arcs having the same blocking time as that of one of their solid arcs.

Table 1: Example illustrating the computation of blocking times.

	Directly blocked by		Priority-inher blocked by		
	J_2	J_3	J_2	J_3	max
J_1		3			3
J_2	*	4	*	3	4
J_3	*	*	*	*	*

Different from the bipartite graph, each vertex in the WDG corresponds to a task but resource. A task τ_L directly blocking a higher-priority task τ_H is represented by a solid arc $a(\tau_L, \tau_H)$ from task vertex τ_L to τ_H, while an indirect block is represented by a dotted arc. In Figure 3(a), the bipartite graph is derived from Figure 2(a) and can be reduced to the WDG in Figure 3(b). The maximum priority inversion of J_2 is composed of a direct and indirect blocking incurred by J_3. The values of these two arcs can be combined because the indirect blocking is due to priority inheritance of job J_1. We may label each arc by a 3-tuple; the first element of each 3-tuple gives the actual starting time of higher-priority tasks and is defined as $S(\tau_H)$, while the second element gives the locking time of τ_L on semaphore z and is denoted as $L(\tau_L, z)$. The last element specifies the duration of the maximum priority inversion and is denoted as $I(\tau_L, \tau_H)$. The first two elements are updated during runtime while the third element is derived according to PCP. The example of the 3-tuple labels is illustrated in Figure 3(b). In accordance with the parameters, the proposed evaluation functions are defined as shown in Algorithm 1.

Definition 1 (expected sleeping interval, ESI). In order to prevent job J_L from blocking job J_H, the expected sleep interval for J_L is defined as

$$\mathrm{ESI}_{L,H} = [L(J_L, z), S(J_H)) - \bigcup_{\forall \tau_\lambda \in \mathfrak{J}, \lambda < L} NC_\lambda^{[L,H]}, \quad (1)$$

where $NC_\lambda^{[L,H]}$ denotes the set of noncritical-section intervals of job J_λ in interval $[L(J_L, z), S(J_H))$. The length of $\mathrm{ESI}_{L,H}$ is denoted as

$$\alpha_{L,H} = S(J_H) - L(J_L, z) - \sum_{\forall \tau_\lambda \in I, \lambda < L} \left| NC_\lambda^{[L,H]} \right|. \quad (2)$$

Definition 2 (reduction of priority inversion time, RPI). The expected reduction of priority inversion time due to the processor sleeping in the $\mathrm{ESI}_{L,H}$ is defined. The value of RPI is derived from

$$\beta_{L,H} = I(\tau_L, \tau_H) - \alpha_{L,H}. \quad (3)$$

According to (1), (2), and (3), we define a reward function for each arc in WDG.

Definition 3. A *reward* function for each arc in WDG is defined as

$$REW(\tau_L, \tau_H) = \frac{\beta_{L,H}}{\alpha_{L,H}}. \quad (4)$$

The reward for an arc referred to the reduction of priority inversion time if the processor is switched to sleep during the interval $\mathrm{ESI}_{L,H}$. Whenever a new job J_i arrives, the value of $REW(\tau_L, \tau_i)$ with respect to each arc is refreshed. The larger the value of *REW*, the longer the priority inversion in the schedule will be avoided. For example, in Figure 3(b), the values of $\alpha_{3,1}$ and $\alpha_{3,2}$ are, respectively, $t_4 - t_0'$ and $t_1 - t_0'$, and the values of $\beta_{3,1}$ and $\beta_{3,2}$ are, respectively, $3 - (t_4 - t_0')$ and $7 - (t_1 - t_0')$. In accordance with (4), the values of $REW(3,1)$ and $REW(3,2)$ are $(3 - (t_4 - t_0'))/(t_4 - t_0')$ and $(7 - (t_1 - t_0'))/(t_1 - t_0')$. Assuming that available slack for τ_3 is larger than the value of $(t_1 - t_0')$, the processor decides to sleep in the duration of $[t_0', t_1]$.

By updating the information of arcs in WDG during runtime, we can traverse the vertices of WDG by following the current job and make decisions on switching the processor to active or dormant mode. Additionally, the values of ESI can be updated on the fly according to the dynamic slack time due to early completion of a job. Also, the arcs corresponding to the unused resources due to early completion can be removed from this job vertex.

5. Experimental Results

To evaluate the effectiveness of the proposed scheduling method, we implemented the following techniques and derived their energy consumption.

(i) Primitive PCP: all tasks are executed at maximum speed and use PCP for accessing the shared resources [23].

(ii) P-PCP: all tasks are scheduled using procrastination technique under fixed priority [5].

(iii) DVS-PCP: all tasks are scheduled according to the RM-DVS scheduling based on the slowdown factor proposed in [26].

(iv) LL-PCP: all tasks are scheduled according to the proposed method.

We use the processor power model in [5] and consider energy overhead of shutdown required by on-chip cache. With an energy cost of 13 nJ [15] per memory write, the cost of flushing the data cache is computed as 85 uJ. On wakeup, we assume a cost of 98uJ total energy overhead. Additionally, we assume that cache energy overhead to actual charging of circuit logic requires 300 uJ, and the total cost is 85 + 98 + 300 = 483 uJ. In the experiment, we consider 2000 task sets, and each set contains up to 12 randomly generated tasks. Each task was assigned a random period and WCET in the range [10 ms, 125 ms] and [0.5 ms, 10 ms], respectively, to reflect real life real-time applications. A set of shared resources and its mapping to a set of tasks are generated randomly. For simplicity, the number of resources is less than or equal to the number of tasks of the corresponding task set and all resources are mutually exclusive accessed by those tasks.

In Figure 4, the comparison of the energy consumption of the techniques is as a function of the number of task,

Input: a set of task T, a set of resources R.
(Offline part)

(1) Reduce bipartite graph to WDG (T, A);
(2) Compute the value of I (τ_L, τ_H) with respect to each arc a (τ_L, τ_H) in WDG.

(Online part)

On arrival of a job J_i
(3) Identify a set of tasks T_H containing higher priority task τ_H than that of J_i;
(4) Update the value of $REW(\tau_i, \tau_H)$ for each arc $a(\tau_i, \tau_H)$ in WDG and $\tau_H \in T_H$;
(5) Construct a set A_i' of outgoing arcs of τ_i,
$$A_i' = \left\{ a(\tau_i, \tau_H) \mid \alpha_{i,H} \geq \frac{E_{SW}}{P(S_{idle})} \right\}$$
(6) Compute the static available slack s_H for each job in T_H;
(7) Compare each s_H to the corresponding α value of the arcs in A_i';
(8) Construct an arc set $A_i'' \subset A_i'$ where A_i'' $= \{a(\tau_i, \tau_x) \mid s_H \geq \alpha_{x,i}, a(\tau_i, \tau_x) \in A_i' \text{ and } \tau_x \in T_H\}$;
(9) Search for an arc $a(\tau_i, \tau_x)$ in A_i'' with the maximum value of $REW(\tau_i, \tau_x)$ where $\tau_x \in T_H$;
(10) **IF** the processor is in active mode **THEN** switch the processor to dormant mode until time $S(\tau_x)$;
On the beginning of one of the intervals in $\mathbf{ESI_{L,x}}$
(11) Switching the processor to S_{dorm} until the end of the interval;
On turning the processor to the active mode at time t
(12) Schedule the highest priority job in the ready queue; On early-completion of a job at time t;
(13) Compute dynamic slack time due to early completion;

Algorithm 1: LL-PCP algorithm.

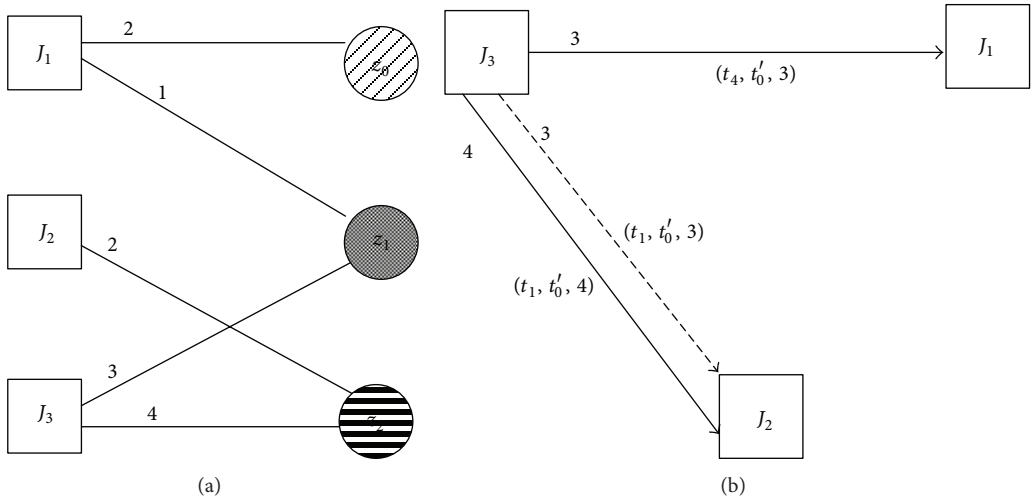


Figure 3: Graph reduction from (a) bipartite graph to (b) WDG.

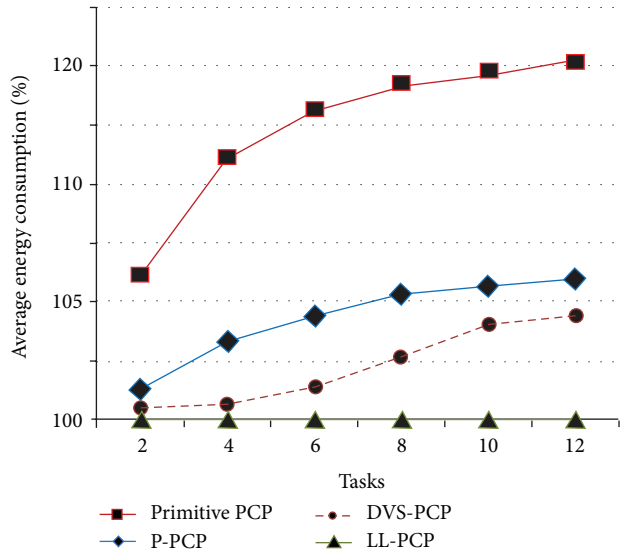


Figure 4: Comparison of energy consumption in the different number of tasks, utilization = 50%.

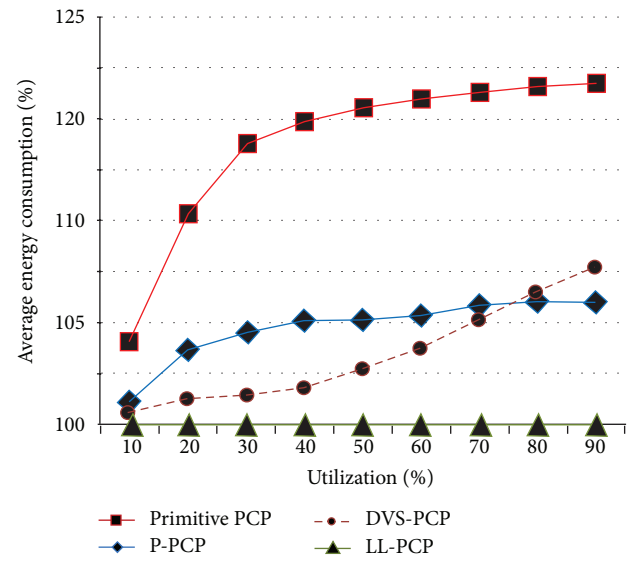


Figure 5: Comparison of energy consumption in the different processor utilization, 6 tasks.

and the energy consumption of all techniques is normalized to LL-PCP. As the number of task increases, primitive PCP starts consuming more energy than other methods. When the number of tasks is greater than four, DVS-PCP starts consuming more energy due to the dominance of leakage and speed switching. The leakage is derived from frequent priority inversion due to increased resource contentions. We see that the LL-PCP results in an up to an additional 4.5% gains over DVS-PCP when the number of tasks is not less than 10.

In Figure 5, the energy consumption of the techniques is as a function of the processor utilization at maximum speed. When the processor utilization increases, DVS technique starts consuming more energy. As the processor utilization is low, DVS-PCP presents significant energy savings. There are idle intervals in these techniques and could be utilized by applying dynamic reclamation methods [3, 8, 12] for more energy gains. These idle intervals can be used for processor shutdown to compare the benefits of our procrastination scheme. Both P-PCP and LL-PCP are procrastination scheme and their energy consumptions outperform that of preemptive PCP through shutdown. Additionally, P-PCP and LL-PCP are not sensitive to the change in processor utilization because they are leakage-aware methods. We see that LL-PCP leads to up to 6% energy gains over P-PCP and an average of 5% energy savings compared to DVS-PCP.

Figure 6 shows the comparison of the length of priority inversion normalized to LL-PCP, as a function of the number of tasks. In consideration of task synchronization, priority inversions directly affect the feasibility of real-time scheduling because they usually postpone actual finish time of higher-priority tasks and would result in deadline misses. When the number of tasks increases, primitive PCP, P-PCP, and DVS-PCP result in increasing priority inversion longer than that of LL-PCP. The main reason is that the number of increases of nonpreemptive resources and access conflictions among the tasks would produce additional short idle periods and switching time that also hamper timely completion of the tasks. LL-PCP reduces an additional 4.8% priority inversion over DVS-PCP at 12 tasks in each task set.

The comparison of the length of priority inversion of the techniques is shown in Figure 7, as a function of the processor utilization at maximum speed. As the processor utilization increases, preemptive PCP, P-PCP, and DVS-PCP result in less increasing priority inversion than that of LL-PCP. The reason is that, in accordance with (4), when the value of $\alpha_{L,H}$ increases and $\beta_{L,H}$ remains unchanged, the returned values of their *REW* function also increases. Obviously, increasing the WCET of each task does not affect the length of priority inversion (i.e., the value of $\beta_{L,H}$) but increases the value of $\alpha_{L,H}$, and therefore the value of $REW(\cdot,\cdot)$ is increased. In accordance with our algorithm, $REW(\cdot,\cdot)$ affects LL-PCP not to switch lower-priority task to dormant mode and therefore produces some priority inversion. However, as the processor utilization is low, there are idle intervals in the schedules and they can be used by LL-PCP to reduce priority inversion. Nevertheless, LL-PCP still outperforms other techniques at any utilization.

The average performance of the abovementioned methods is presented in Table 2 as the setting to the range of processor utilization and the number of tasks in each task set is [10%~90%] and 6, respectively. The proposed method still outperforms primitive PCP, P-PCP, and DVS-PCP with regard to both energy consumption and priority inversion.

The proposed method may have additional offline processing time comparing to other methods. With respect to time complexity, the proposed method takes $O(n^2)$ time during the offline processing because of the required data structures of WDG where n denotes the number of tasks. However, other methods that required analytical function to predict blocking time have to construct a blocking time table similar to Table 1 which also takes $O(n^2)$ time. When using

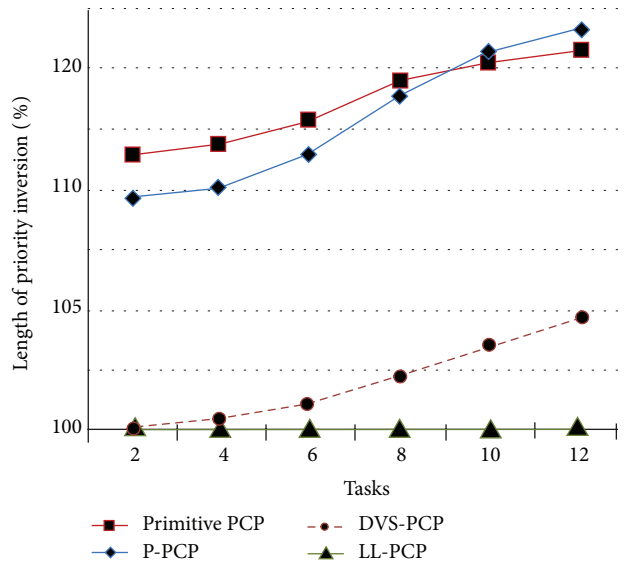


Figure 6: Comparison of priority inversion in the different number of tasks, utilization = 50%.

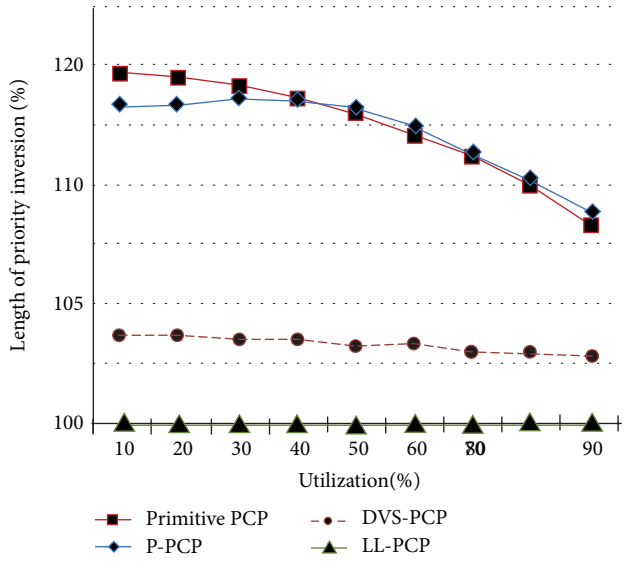


Figure 7: Comparison of priority inversion in the different processor utilization, 6 tasks.

the existing slack-reclamation methods such as the algorithms in [3, 8, 22], the time complexities of the online parts of primitive PCP, P-PCP, DVS-PCP, and LL-PCP require $O(n \log n)$ time. In LL-PCP, the time complexities of each step are discussed as follows. Step (3) in the algorithm searching for the tasks with respect to job J_i takes $O(n)$ time. Step (4) updating the *REW* values of the arcs incident from τ_i takes $O(n)$ time. Similarly, step (5) constructing an outgoing arc set with respect to τ_i requires $O(n)$. In steps (6) and (7), the time complexity depends on slack-reclamation methods applied in current schedule and takes maximum $O(n \log n)$ [7–9, 18, 22]. Steps (7) and (8) comparing the values of the arcs in A_i' and constructing a new arc set A_i'' with respect to τ_i require $O(n)$ time. Finally, in step (9), searching an arc $a(\tau_i, \tau_x)$ in A_i'' with the maximum value of $REW(\tau_i, \tau_x)$ requires $O(n)$ time, and step 10 takes constant time to switch the processor mode. Therefore, the time complexity of the online part of LL-PCP is $O(n \log n)$.

Table 2: Comparisons of average energy consumption and priority inversion.

	Primitive PCP	P-PCP	DVS-PCP	LL-PCP
Energy consumption	100%	87.7%	86%	82%
Priority inversion	100%	97.2%	86.8%	84%

6. Conclusions

This paper reduces priority inversion of real-time task synchronization with speed/voltage switching overhead consideration. The objective is to minimize the priority inversion of a given task set and reduce the leakage energy, provided that the schedulability of tasks is guaranteed. By traversing the vertex in WDG, the scheduling decisions can be done efficiently during the runtime.

For further study, we shall explore the minimization issues of energy consumption on dynamic priority assignment, for example, the Earliest Deadline First Scheduling with Stack Resource Policy [28]. Future research and experiments in these areas may benefit several mobile system designs.

Conflict of Interests

The authors declare that there is no conflict of interests with any financial organization regarding the experimental process and discussion in the paper.

Acknowledgment

The authors would like to thank the National Science Council of the Republic of China, Taiwan, for financially supporting this research under NSC 102-2221-E-025-003, NSC 102-2622-E-025-001-CC3, and NSC-101-2622-E-025-002-CC3.

References

[1] D. Duarte, N. Vijaykrishnan, M. J. Irwin, and Y. F. Tsai, "Impact of technology scaling and packaging on dynamic voltage scaling techniques," in *Proceedings of the 15th Annual IEEE International ASIC/SOC Conference*, September 2002.

[2] J.-J. Chen and T.-W. Kuo, "Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '06)*, pp. 153–162, Ottawa, Canada, June 2006.

[3] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," in *Proceedings of the International Symposium on Low Electronics and Design (ISLPED '01)*, pp. 46–51, Huntington Beach, Calif, USA, August 2001.

[4] S. Irani, S. Shukla, and R. Gupta, "Algorithms for power savings," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 37–46, Baltimore, Md, USA, 2003.

[5] R. Jejurikar and R. Gupta, "Procrastination scheduling in fixed priority real-time systems," in *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '04)*, pp. 57–65, June 2004.

[6] R. Jejurikar, C. Pereira, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," in *Proceedings of the 41st Design Automation Conference*, pp. 275–280, June 2004.

[7] W. Kim, J. Kim, and S. L. Min, "A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis," in *Proceedings of the Design Automation and Test in Europe (DATE '02)*, pp. 788–797, August 2002.

[8] W. Kim, J. Kim, and S. L. Min, "Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED '03)*, pp. 396–401, ACM Press, New York, NY, USA, August 2003.

[9] W. Kim, J. Kim, and S. L. Min, "Preemption-aware dynamic voltage scaling in hard real-time systems," in *Proceedings of the 2004 International Symposium on Lower Power Electronics and Design (ISLPED '04)*, pp. 393–398, August 2004.

[10] W. Y. Liang, P. T. Lai, and C. W. Chiou, "An energy conservation DVFS algorithm for the android operating system," *Journal of Convergence*, vol. 1, no. 1, pp. 93–100, 2010.

[11] B. Mochocki, X. S. Hu, and G. Quan, "Transition-overhead-aware voltage scheduling for fixed-priority real-time systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 12, no. 2, Article ID 1230803, 2007.

[12] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low power embedded operating systems," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pp. 89–102, ACM Press, New York, NY, USA, 2001.

[13] G. Quan, L. Niu, X. S. Hu, and B. Mochocki, "Fixed priority scheduling for reducing overall energy on variable voltage processors," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS '04)*, pp. 309–318, December 2004.

[14] Y. H. Lee, K. P. Reddy, and C. M. Krishna, "Scheduling techniques for reducing leakage power in hard real-time systems," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS' 03)*, pp. 105–112, 2003.

[15] R. Jejurikar and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real-time embedded systems," in *Proceedings of the 42nd Design Automation Conference (DAC '05)*, pp. 111–116, New York, NY, USA, June 2005.

[16] L. Niu and G. Quan, "Reducing both dynamic and energy consumption for hard real-time systems," in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pp. 140–148, September 2004.

[17] P. Baptiste, "Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management," in *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*, pp. 364–367, January 2006.

[18] R. Jejurikar and R. Gupta, "Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," in *Proceedings of the International Symposium on Lower Power Electronics and Design (ISLPED '04)*, pp. 78–81, August 2004.

[19] "International Technology Roadmap for Semiconductors," 2002, http://public.itrs.net.

[20] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, 1999.

[21] H. Huang, M. Fan, and G. Quan, "On-line leakage-aware energy minimization scheduling for hard real-time systems," in *Proceedings of the 17th Asia and South Pacific Design Automation Conference (ASP-DAC '12)*, pp. 677–682, Sydney, Australia, February 2012.

[22] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 5, pp. 584–600, 2004.

[23] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

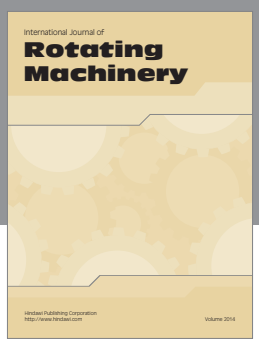
[24] A. P. Silberschatz, B. Galvin, and G. Gagne, *Operating System Concepts*, John Willey and Sons,, 2011.

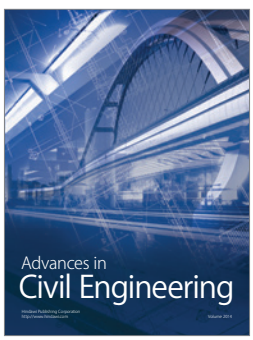
[25] F. Zhang and S. T. Chanson, "Processor voltage scheduling for real-time tasks with non-preemptible sections," in *Proceedings of the 23rd Proceedings IEEE Real-Time Systems Symposium (RTSS '02)*, pp. 235–245, Austin, Tex, USA, December 2002.

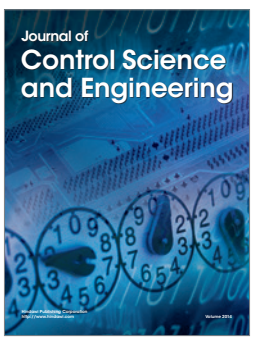
[26] R. Jejurikar and R. Gupta, "Energy-aware task scheduling with task synchronization for embedded real-time systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1024–1036, 2006.

[27] J. A. Butts and G. S. Sohi, "Static power model for architects," in *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 191–201, Monterey, Calif, USA, December 2000.

[28] J. W. S. Liu, *Real-Time Systems*, Prentice Hall, Upper Saddle River, NJ, USA, 2000.







Hindawi
Submit your manuscripts at
http://www.hindawi.com