An Energy-aware Scheduling for Real-time Task Synchronization Using DVS and Leakage-aware Methods

Da-Ren Chen
Department of Information Management, National Taichung University of Science and Technology
Taichung, Taiwan, R.O.C.
danny@nutc.edu.tw

You-Shyang Chen
Department of Information Management,Hwa Hsia Institute of Technology
Taipei, Taiwan, R.O.C.
ys_chen@cc.hwh.edu.tw

***Abstract*-Due to the importance of resource allocation and energy efficiency, this paper considers minimizing priority inversion and energy consumptions in the embedded real-time systems. While dynamic voltage scaling (DVS) is known to reduce dynamic power consumption, it also causes increased blocking time of lower priority tasks and leakage energy consumption due to increased execution. We proposed a concept of latency locking to prevent priority inversion using sleeping mode and define a block-free interval in which both DVS and leakage-aware methods can be applied. In order to compute the optimal sleeping time and its duration and to meet the timing constraints, we also propose a weighted directed graph (WDG) to obtain additional task information. By traversing WDG, task information can be updated online and the scheduling decisions could be done in linear time complexity.**

Keywords*-real-time scheduling; priority ceiling protocol; priority inversion.*

I. INTRODUCTION

The power-aware real-time scheduling problem has been well studied, relatively few work address energy-efficient real-time task synchronization. Most embedded real-time applications have shared resources in the system and mutually exclusive access to shared resources. On such a system, real-time tasks can lead to priority inversion if a task is blocked by a lower priority task due to non-preemptive resource sharing. The recent related work is an extension of Priority Ceiling Protocol (PCP) [7] in frequency inheritance. Zang and Chanson [9] 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 [5] computes two slowdown factor, which can be classified into static slowdown, computed offline based on task properties, and dynamic slowdown, computed using online task execution information. Chen et al. [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. The work which is designed with DVS capability to slowdown or speedup the blocked or blocking tasks in the critical section. These methods may receive additional priority inversion, and thus increase the difficulties of schedulability. Our goal is to propose a energy-aware task synchronization protocol, which can minimize the priority inversion and reduce the energy-consumption of the processor. The basic idea is to postpone the intention to the locking on resources invoked by lower-priority tasks, and to construct a blocking-free interval in which the tasks' speed can be reduced using DVS. Additionally, the extended execution due to DVS does not increase the priority inversion.

II. 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 and indexed in order of decreasing priorities. 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 length of T_i is unique in order to have each task a unique priority index in the rate-monotonic (RM) scheduling. The j^{th} invocation of task τ_i is denoted as $J_{i,j}$ whose actual start and finish times are denoted as $S(J_{i,j})$ and $F(J_{i,j})$, respectively. Notation $s_{i,j}$ denotes the available static slack time for job $J_{i,j}$. Job $J_{i,j}$ could be completed early at time $EC(i, j)$ during its WCET.

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_{dorm}=0 by scaling the static power consumption [1], while the system clock and chipset retain necessary functions to support motoring and waking up processor at right time. To execute jobs, the processor has to be in the active mode with speed S_{active}. The time and power overhead required to switch the processor to the dormant mode can be neglected by treating them as a part of the overhead to turn on the processor. Let E_{sw} and t_{sw} denote the energy and time overhead, respectively, for switching from dormant mode to active mode. When the processor is idle in the active mode, the processor executes NOP instruction at processor speed S_{idle} for low-power consumption. Additionally, when the idle interval is longer than break-even time $\frac{E_{sw}}{P(S_{idle})}$, turning it to the dormant mode is worthwhile. The DVS model is similar to those in lpWDA [6] and can be abridged here.

We assume that semaphores are used for task synchronization. All tasks are assumed to be preemptive, while the access to the shared resources must be serialized. Therefore, 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 [8]. The k^{th} critical section of task τ_i is denoted as $z_{i,k}$ which is properly

nested. Each task specifies the access to the shared resource types and the required WCETs. With the given information in [4], we can compute the maximum blocking time for a task. In the different resource synchronization protocol, such as PCP [7], 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 an energy-aware real-time scheduling with task synchronization based on PCP, which can minimize the priority inversion while reducing the 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 not only postpone the intention of locking on the resources invoked by lower-priority tasks but also construct a blocking-free interval to slowdown the tasks speed using DVS methods.

III. MOTIVATING EXAMPLE

Suppose that we have three jobs J_1 , J_2 and J_3 , and two shared data structures protected by the binary semaphores z_1 and z_2 in the system. In accordance with PCP, the sequence of events is depicted in Fig. 1(a). A line at a lower level indicates that the corresponding job is in 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 that

- At time t_0, J_3 is initiated and it then locks semaphore z_1.
- At time t'_1, J_2 is initiated and preempts J_3.
- At time t_2, J_2 cannot lock z_1, and J_3 inherits the priority of job J_2 and resumes execution.
- At time t_3, J_1 preempts J_3 in the critical section of z_1 and executes its noncritical section code.
- At time t_4, J_1 attempts to enter its critical section z_1 and is blocked by J_3 due to priority ceiling, and J_3 inherits the priority of job J_1.
- At time t_5, J_3 exits its critical section z_1 and returns to its original priority. J_1 is awakened and locks z_1.

The priority inversions are $[t_2, t_3]$ and $[t_4, t_5]$.

This research work is motivated by the significant priority inversion and power consumption due to unused slack time and context switches. The objective is to minimize the priority inversion while reducing energy-consumption. 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 Fig.1(b) postpones the request of lower-priority task intent to a lock semaphore. At time t_1, J_3 has available slack in interval $[t_9, t_{10}]$ with length longer than break-even time. When a system is conscious that J_3 has intent to lock z_1, it computes the upcoming start time of higher priority tasks that might be blocked by J_3 according to PCP. In the example, J_1 and J_2 could be blocked by J_3 due to z_1, and the lengths of interval $[t_1, t_1']$ are less than the available slack. Therefore, processor switches to dormant mode at time t_1 until the start time of J_2. At time t_1', processor becomes *active* and J_2 preempts J_3 such that J_3 is still unable to lock z_1, and thus J_2 could lock z_2 at time t_2. However, J_1 could be blocked when it intends to lock z_1 if J_2 successfully lock z_2 at t_2. To further reduce priority inversion, job J_2's intent to lock z_2 should be postponed after t_3. Therefore, comparing to the result of Fig. 1(a), the idea eliminates all priority inversions in intervals $[t_3, t_7]$.

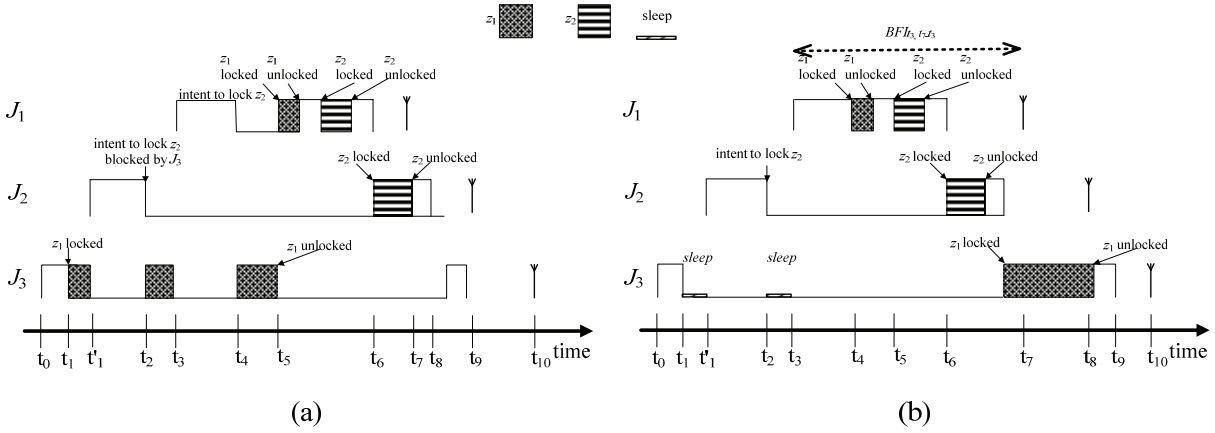

Fig.1. The task synchronization of (a) primitive PCP and (b) latency locking method

IV. LATENCY LOCKING PCP

In this research work, PCP is extended with the concept of latency locking, referred to as LL-PCP. The idea is to do pre-analysis of possible priority inversion and available slack time in the schedule. The objective is to derive the best timing and duration for switching processor to dormant mode, and thus minimize priority inversion. To understand and control the sequence of intent to lock resources, tasks are organized as a WDG reduced from the resource allocation bipartite graph in [4]. Let $G=(U, V, E)$ denote a bipartite graph whose partition of vertices has two subsets U and V. E denotes the set of edges of G, and U denote a task set $\mathfrak{T}$. Let WDG($\mathfrak{T}$, A) denote a weighted directed graph whose vertices in $\mathfrak{T} \subseteq U$ 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

Step1. 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 exists two or more edges $e_{x,v}$ and $e_{y,v}$ in G where $z_v \in V$.

Step2. 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 Step1.

Step3. 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.

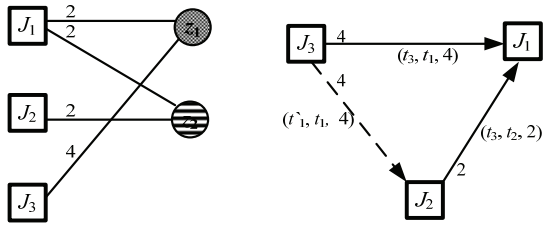

Fig.2 Graph reduction from (a)bipartite graph to (b)WDG

Fig.(2) illustrates the graph reduction from bipartite to WDG. In the bipartite graph, each indirect edge is labeled with the time required to access the resources. Different from the bipartite graph, WDG has a vertex for each task but resources. A task τ_L directly blocks a higher-priority task τ_H is represented by an solid arc $a(\tau_L, \tau_H)$ from task vertex τ_L to τ_H, while an indirect block is represented by an dotted arc. In Fig. 2(a), the bipartite graph is derived from Fig.1(a) and can be reduced to the WDG in Fig.2(b). The maximum priority inversion of J_2 is an indirect blocking incurred by J_3. We may label each arc by a 3-tuple, the first element of each 3-tuple give the actual starting time of higher-priority tasks and defined as $S(\tau_H)$, while the second element gives the locking time of τ_L on semaphore z and denoted as $L(\tau_L, z)$. The last element specifies the duration of the maximum priority inversion and denoted as $I(\tau_L, \tau_H)$. The first two elements are updated during runtime while the third element is derived directly from the arcs in WDG. The example of the 3-tuple labels is illustrated in Figure 2(b), we have the following definitions.

Algorithm LL-PCP

Input: a set of task $\mathfrak{T}$, a set of resources $\mathcal{R}$

(Offline part)

1. Reduce bipartite graph to WDG($\mathfrak{T}$, A);
2. Compute the value of $I(\tau_L, \tau_H)$ with respect to each arc $a(\tau_L, \tau_H)$ in WDG;

(Online part)

On arrival of a job J_i

3. Identify a set of tasks $\mathfrak{T}_H$ containing higher priority task τ_H than that of J_i;
4. Compute the value of $REW(\tau_i, \tau_H)$ for each arc $a(\tau_i, \tau_H)$ in WDG and $\tau_H \in \mathfrak{T}_H$;
5. Construct a set $A_i'$of outgoing arcs of τ_i, $A_i'=\{ a(\tau_i, \tau_H) | \alpha_{i,H} \geq \frac{E_{sw}}{P(S_{idle})}\}$
6. Compute the static available slack s_H for each job in $\mathfrak{T}_H$;
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 $A_i''=\{a(\tau_i, \tau_x) | s_H \geq \alpha_{x,i}$, $a(\tau_i, \tau_x) \in A_i'$ and $\tau_x \in \mathfrak{T}_H\}$;
9. Search for an arc $a(\tau_i, \tau_x)$ in A_i'' with the maximum value of $REW(\tau_i, \tau_x)$ where $\tau_x \in \mathfrak{T}_H$;

On beginning of one of the intervals in $ESI_{L,x}$

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;

On completing or beginning a job J_i at time t

14. **IF** completes early **THEN**
    obtain dynamic slack s_i^d from $F(J_i)$–$EC(J_i)$;
15. Set BFI=[S(J_x), $F(J_x)$] according to the recently carried out $ESI_{L,x}$;
16. Slowdown the speed of tasks whose deadlines are earlier than $F(J_x)$ using lpWDA[6];

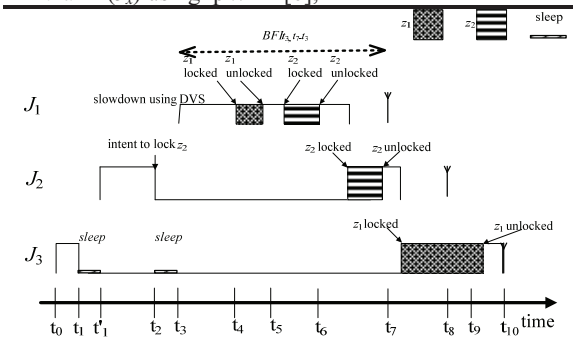

Fig.3 An example using DVS scheduling

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

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

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 $ESI_{L,H}$ denotes 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. defines the expected reduction of priority inversion (*RPI*) due to the processor sleeping in the $ESI_{L,H}$. The value of *RPI* is derived from

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

According to equations (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

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

The *reward* for an arc is referred to the reduction of priority inversion time if the processor is switched to sleep during the interval $ESI_{L,H}$. 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 will be avoided. For example, in Fig.2(b), the values of $\alpha_{3,1}$ and $\alpha_{3,2}$ are set respectively $x= t_3 - t_1 - (t_2 - t'_1)$ and $y= t'_1 - t_1$, and the values of $\beta_{3,1}$ and $\beta_{3,2}$ are respectively $4 - x$ and $4 - y$. In accordance with equation 4, the values of *REW*(3,1) and *REW*(3,2) are $\frac{4-x}{x}$ and $\frac{4-y}{y}$, and obviously *REW*(3,1) < *REW*(3,2). Assuming that available slack for τ_3 is larger than the values of x and y, the proposed algorithm switchs the processor to sleep in the duration of [t_1, t'_1], and traverses from vertex J_3 to J_2. In the vertex J_2, we can traverse from J_2 to J_1 by switching the processor to sleep mode in interval [t_2, t_3].

Definition 4. (Blocking-free interval, *BFI*)

A time interval $BFI_{t,\ell}$ is said to be blocking free in the real-time task synchronization if the interval [t, $t+\ell$] does not have any priority inversion.

Lemma 1. When the time at which J_L has intent to lock z is later than the $S(J_H)$, they do not give rise to priority inversion during interval [$S(J_H)$, $F(J_H)$].

***Proof sketch*:** In accordance with WDG, J_H has higher priority than J_L, as soon as J_L begins after $S(J_H)$, J_L cannot lock z until J_H completes. Therefore, J_H is not preempted by J_L until the completion time of J_H.

Obviously, jobs J_H and J_L do not give rise to the priority inversion in the interval [$F(J_H)$, D_H] when J_H completes before F(JH). Therefore, when the processor sleeps in interval $ESI_{L,H}$, the value of *BFI* can be derived from

$$BFI_{t,\ell}=[S(J_H), D_H] \quad (5)$$

for all jobs J_H are successors of J_L in WDG.

The purpose of BFI is to identify the jobs for saving more energy using DVS. The jobs whose deadlines are in the BFI interval can compute their available slack time to decrease their speed and satisfy their timing constraints. The slack computation such as lpWDA [6] can be applied without modification to our method. An example is presented in Fig.3. By updating the information of arcs in WDG during runtime, we can traverse the WDG by following the current job and make decisions on switching the processor to active or dormant mode.

V. CONCLUSIONS

This work-in-progress continuously improves energy-efficiency of real-time task synchronization with speed switching overhead consideration. By using DVS and leakage-aware techniques, we decrease not only the priority inversion but also energy consumption in the real-time systems. The objective is to minimize the priority inversion and reduce both dynamic and leakage energy, provided that the schedulability of tasks is guaranteed. By traversing the vertex of WDG, the scheduling decisions can be done efficiently during the runtime. Another characteristic is that, in the proposed concept, DVS does not worsen the situation of inevitable priority inversion.

For further study, we shall explore an evaluation function that provides suggestions on how to use DVS or leakage-aware technique during runtime. Future research and experiments in these areas may benefit several mobile system designs.

VI. REFERENCES

[1] Butts, J. Adam, and Sohi, G. S. 2000. A Static Power Model for Architects. In *Proceedings. of the 33rd Annual International Symposium on Microarchitecture* (In Monterey, California from Dec. 10 - 13). MICRO-33, 191-201.

[2] Chen, J.-J., and Kuo, T.-W. 2006. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems* (Ottawa, June 14-16, 2006). *LCTES* 06. ACM, 153–162.

[3] Irani, S., Shukla, S., and Gupta, R. 2003. Algorithms for power savings. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (On the Inner Harbour Baltimore, MD, USA Jan. 12-14, 2003). DA 03. ACM, New York, NY, 37–46.

[4] Jane W. S. Liu. Real-time systems. Prentice Hall PTR Upper Saddle River, NJ, USA, (2000), ISBN:0130996513.

[5] Jejurikar, R., and Gupta, R. K. 2005. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the 42nd Design Automation Conference* (San Diego, CA, USA, June 13-17). *DAC* 05. ACM, New York, NY, 111–116.

[6] Kim, W., Kim,J., and Min, S. L. 2003. 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* (ISPLED'03), ACM Press, New York, NY, 2003, pp. 396–401.

[7] Sha, L., Rajkumar, R., and Lehoczky,J. P. 1990. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. on Computers*, 39 (Sept. 1990), no.9, 1175-1185.

[8] Silberschatz, A. P., Galvin, B., and G. Gagne. 2011. Operating System Concepts. John Willey and Sons, Inc., (2011).

[9] Zhang, F., and Chanson, S. T. 2002. Processor voltage scheduling for real-time tasks with non-preemptible sections. In *23rd Proceedings IEEE Real-Time Systems Symp.*, (Austin, TX, Dec. 2002). RTSS 02. IEEE, 235–245.