International Journal of Multimedia and Ubiquitous Engineering
Vol.10, No.7 (2015), pp.171-182
http://dx.doi.org/10.14257/ijmue.2015.10.7.18

On-Line Dynamic Voltage Scaling for EDZL Scheduling on Symmetric Multiprocessor Real-Time Systems

Xuefeng Piao † and Moonju Park[1] ‡

†*School of Computer Science and Technology, Harbin Institute of Technology, Weihai, Shandong, China*
‡ *School of Computer Science and Engineering, Incheon National University, Incheon, Korea*
hbpark@hit.edu.cn, mpark@incheon.ac.kr

Abstract

EDZL (Earliest Deadline Zero Laxity) scheduling is known to be at least as EDF (Earliest Deadline First) in task scheduling on symmetric multiprocessor real-time systems; however, there are few works on energy conversation on the EDZL. This paper proposes an on-line Dynamic Voltage Scaling (DVS) algorithm of the global EDZL to reduce energy consumption of real-time tasks. The proposed algorithm dynamically adjusts processor speed at each scheduling point with re-assigning deadlines of active jobs that reduces power consumption of processors while making all real-time tasks schedulable by EDZL. Extensive simulations show that the proposed algorithm reduces power consumption more than the previous algorithm for EDZL.

Keywords: *Real-time scheduling, Multiprocessor, Dynamic voltage scaling, EDZL*

1. Introduction

As the processing power of embedded processors grows, modern embedded systems can use the computation power of multiprocessors to run complex real-time applications. However, due to limited power supply, minimizing energy consumption while guaranteeing timing constraints is an important design issue of multiprocessor embedded systems. To reduce energy consumption, dynamic voltage scaling techniques have been introduced to exploit the hardware characteristics of processors by lowering supply voltage or speed. A small reduction in processor speed, which is linearly proportional to supply voltage, can result in a significant reduction in energy consumption since power consumption of a processor is in quadratic relation with supply voltage or speed [1]. But lowering supply voltage or speed might affect the schedulability of real-time tasks on the system. Therefore, the main objective of DVS algorithms is to control processor supply voltage or speed without missing any deadlines of real-time tasks.

There have been many DVS algorithms proposed to reduce energy consumption for real-time tasks. Aydin *et al*. [2] proposed energy-aware algorithms for periodic tasks on multiprocessor platforms. Also, polynomial time approximation algorithms were proposed for frame-based tasks in [3,4]. Based on the observation that tasks might complete earlier than their worst-case execution time, a slack time reclamation algorithm was proposed in [5]. Though the global EDZL [6] outperforms EDF [7-10], few works on energy conservation for EDZL other than our previous works in [11,12] could be found. In previous works, we proposed an off-line algorithm and an on-line algorithm for EDZL; the off-line algorithm determines a static processor speed at design time and the on-line algorithm determines a dynamic speed at each scheduling point. This paper is an extension of our previous works: an enhanced on-line DVS algorithm of the global EDZL

[1] *Corresponding author*

ISSN: 1975-0080 IJMUE

scheduling which makes all real-time tasks schedulable while reducing power consumption of processors on symmetric multiprocessor real-time systems.

The rest of this paper is organized as follows. Section 2 defines the system model. Section 3 briefly reviews the EDZL scheduling and Section 4 presents a deadline re-assignment method for scaling voltage with the EDZL scheduling. In Section 5, we present a new on-line DVS algorithm based on the deadline re-assignment method. Section 6 presents the simulation results comparing energy consumption of the proposed method with the previous works and Section 7 concludes our work.

2. System Model

We consider a preemptive hard real-time system in which periodic tasks are scheduled by EDZL on m processors running at an identical speed. A periodic task set $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ consists of n independent periodic tasks. A periodic task τ_i is characterized by (e_i, p_i), where e_i represents the worst case execution time at the maximum processor speed and p_i represents a period. The relative deadline of each task is equal to its period. The utilization of τ_i is defined as

$$U_i = e_i/p_i. \quad (1)$$

The total utilization of a task set τ is defined as

$$U(\tau) = \sum_{i=1}^{n} U_i. \quad (2)$$

The maximum utilization of τ is defined as

$$U_{\max}(\tau) = \max_{\tau_i \in \tau}(U_i). \quad (3)$$

An execution instance of a periodic task τ_i at time t, denoted by $\mathcal{J}_i(t)$, is called as a job which is characterized by $(\mathcal{C}_i(t), \mathcal{D}_i(t))$, where $\mathcal{C}_i(t)$ represents a remaining execution time and $\mathcal{D}_i(t)$ represents an absolute deadline at time t. An active job set at time t is denoted by $\mathcal{J}(t) = \{\mathcal{J}_i(t)\}$. The laxity of $\mathcal{J}_i(t)$ is defined as

$$\mathcal{D}_i(t) - t - \frac{\mathcal{C}_i(t)}{\mathcal{S}(t)}, \quad (4)$$

where $\mathcal{S}(t)$ represents the speed of processors at time t normalized to the maximum speed of the processor. The density of $\mathcal{J}(t)$ is defined as

$$\lambda_i(t) = \frac{\mathcal{C}_i(t)}{\mathcal{D}_i(t) - t}. \quad (5)$$

The density set for $\mathcal{J}(t)$ is defined as

$$\lambda(t) = \{\lambda_i(t)\}. \quad (6)$$

Then the maximum density of $\mathcal{J}(t)$ is defined as

$$\lambda_{\max}(t) = \max_{\lambda_i(t) \in \lambda(t)}(\lambda_i(t)). \quad (7)$$

Now we introduce the power model used in this paper. Power consumption of a processor is dominated by dynamic power dissipation

$$P_D = C_{ef} \cdot V_D^2 \cdot f, \quad (8)$$

where C_{ef} represents effective switched capacitance, V_D represents supply voltage and f represents clock frequency.

The circuit delay t_d is inversely related to the supply voltage V_D as given by the formula

$$t_d = \frac{kV_D}{(V_D - V_t)^2}, \quad (9)$$

where k is a constant and V_t is a threshold voltage. Since clock frequency is inversely proportional to circuit delay and processor speed is approximately proportional to the supply voltage [13], the power consumption is roughly proportional to the cube of the supply voltage.

In essence, Voltage scaling technique saves energy at the expense of increased latency. We assume that the processor speed can be adjusted to any value from 0% to 100% of the maximum processor speed, and power loss of voltage switching and time overhead to change processor supply voltage/speed are assumed to be negligible as in

[4].

3. EDZL Overview

In multiprocessor real-time systems, scheduling algorithms is classified into a partitioned approach or a global approach by the job-level migration. If no migration is permitted in a scheduling algorithm, it is referred as partitioned approach; otherwise, it is a global approach. Each approach has advantages over the other. Comparing to the global scheduling approach, the partitioned scheduling has the following advantages: (a) there is no penalty in terms of migration cost, (b) when a task overruns its worst-case execution time, it only affects tasks on the same processor, and (c) the overhead of manipulating a separate run-queue per processor is lower than that of manipulating a single global queue. On the other hand, the global scheduling has the following advantages over the partitioned scheduling: (a) there are typically fewer context switches and preemptions, (b) there is no need for re-balancing and task re-allocation even if the task set changes in run time, and (c) if the actual task execution is less than its worst-case execution time, then spare capacity which can be used later for other task is reserved. In this sense, global scheduling is more appropriate for open systems [14] where tasks can join and leave dynamically.

EDF [15], LLF (Least Laxity First) [16], EDF-US [17] and fpEDF [18] are some examples of the scheduling algorithms following the global approach. The global EDF assigns the highest priority to jobs with the earliest deadline and is optimal on uniprocessor, but the utilization bound could be very low in the worst case on multiprocessor. The LLF might utilize processor higher than the global EDF but it can cause a large number of context switches. The EDF-US and the fpEDF are variants of EDF. They divide tasks into higher utilization and lower utilization tasks. The higher utilization tasks are given the highest priority and lower utilization tasks are given priorities according to EDF. Two algorithms differ in the boundary condition that dives tasks and in their schedulable utilization bound.

Combining the advantage of EDF and LLF, EDZL is introduced to provide a high schedulable utilization bound with a small number of preemptions. EDZL considers both deadline and laxity of jobs in priority assignment. Jobs with zero laxity are assigned the highest priority and other jobs are assigned priorities by EDF, and ties between jobs with same priority are broken arbitrarily. Since the global EDF and the global EDZL scheduling are both work conserving [7], the processors are never idle if there is a ready job. A scheduling algorithm A_1 is said to strictly dominate another scheduling algorithm A_2 if any task set schedulable by A_2 is also schedulable by A_1 and there is a task set which is schedulable by A_1 but not schedulable by A_2. The global EDZL strictly dominates the global EDF and it is suboptimal for two processors [9]. It was shown that the EDZL is completion time predictable in [20] in the sense defined by Ha and Liu [19].

For EDZL scheduling, we proposed low-power algorithms for static speed determination and dynamic speed determination in [11,12]. To determine a static speed, we was used the schedulability condition of EDZL and the schedulability condition of EDF. Since EDZL strictly dominates EDF, a speed obtained from EDF schedulability condition can be used as a static speed of EDZL. Therefore, the static speed is determined as a small value between a speed obtained from EDF schedulability condition and a speed obtained from EDZL schedulability condition. To determine a dynamic speed at each scheduling point, we was used a property of EDZL which is if the density of an active job set within an time interval from current time to the minimum upcoming release time, and the maximum density within the interval is less than or equal to one, then the active job set is schedulable by EDZL even though the deadlines of all active jobs are set to the minimum upcoming release time. Therefore, a speed can be obtained whenever an active

job set satisfies the above conditions. Ultimately, the dynamic speed is determines as a small value between the static speed and the speed obtained from the property of EDZL.

4. Deadline Re-Assignment Method

Let us introduce the following notations for further discussion. At time *t*, the next release time of a task τ_i is given by

$$\mathcal{R}_i(t) = \mathcal{D}_i(t) + p_i. \quad (10)$$

The first k upcoming release times of all tasks sorted by increasing order at time t are denoted by $\mathcal{R}^1(t), \mathcal{R}^2(t), \cdots, \mathcal{R}^k(t)$. We can choose any k such that

$$\mathcal{R}^k(t) \leq \max\{\mathcal{D}_i(t)\}. \quad (11)$$

The *minimum upcoming release time* is defined as

$$\mathcal{R}_{\min}(t) = \min_{\tau_i \in \tau}(\mathcal{R}_i(t)) = \mathcal{R}^1(t). \quad (12)$$

Algorithm 1: Absolute Deadline Assignment

Input: a job set $\mathcal{J}(t)$
Output: a modified job set with absolute deadlines reassigned
forall $\mathcal{J}_i(t) \in \mathcal{J}(t)$ **do**
 for $j = k$ **downto** 1 **do**
  if $\mathcal{D}_i(t) \geq \mathcal{R}^j(t)$ and $\frac{\mathcal{C}_i(t)}{\mathcal{R}^j(t)-t} \leq 1$ **then**
   $\mathcal{D}_i(t) \leftarrow \mathcal{R}^j(t)$
  endif
 endfor
endfor

In previous work of [11], the method of dynamic speed calculation for EDZL was applicable only when all the active jobs can be completed before $\mathcal{R}_{\min}(t)$. Thus the previous method could be applied for very limited cases only. If active jobs are not to be completed before $\mathcal{R}_{\min}(t)$, tasks run at the static speed, so additional energy saving could not be achieved even though actual execution time of a job is shorter than the worst case execution time. However, we found that by considering the farther release times of tasks, the speed of the processors can be more reduced, which may result in less power consumption. This is done by re-assigning deadlines that makes tasks schedulable by EDZL.

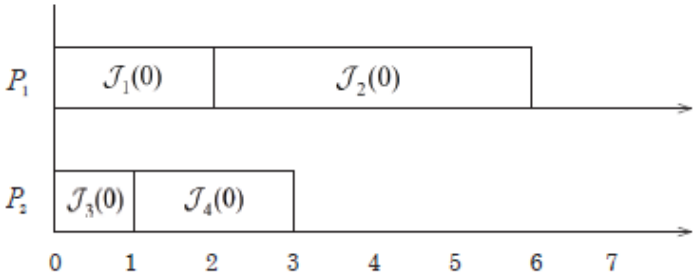
Algorithm 1 shows the pseudo-code for absolute deadline assignment. The algorithm tries to find a deadline assignment with which the jobs can complete before $R^i(t)$ where $i \leq k$. The absolute deadlines are set to the upcoming release times by Algorithm 1 in $O(kn)$ time when there are n tasks. An assigned deadline is shorter than or equal to the existing one. The schedulability of the jobs with re-assigned deadline will be discussed later.

Example: Let us consider a periodic task set τ is $\tau_1 = (2,5), \tau_2 = (4,12), \tau_3 = (1,4), \tau_4 = (2,8)$. Assume that the tasks in τ are released at time zero for the first time simultaneously. When the job set is scheduled by EDZL on two processors P_1 and P_2, the job set is scheduled in the same way as the EDF schedule since no job reaches zero laxity as shown in Figure 1(a).

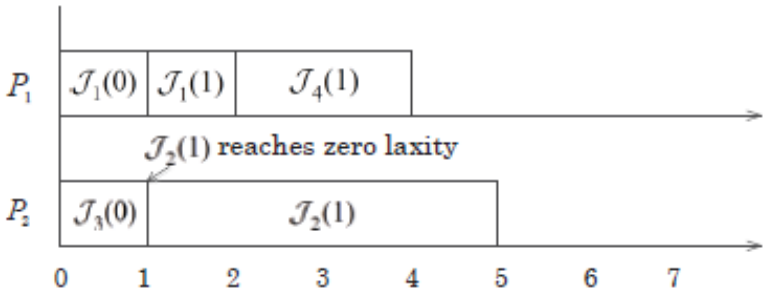
Consider the situation at time 1 when the job of τ_3 finishes. The active job set $\mathcal{J}(1)$ is $\mathcal{J}_1(1) = (1,5), \mathcal{J}_2(1) = (4,12), \mathcal{J}_4(1) = (2,8)$. For this job set, the dynamic speed determination method presented in [11] cannot be applied because $\frac{1+4+2}{4-1} > 2$.

However, if we re-assign the absolute deadlines of jobs as $\mathcal{D}_1(1) = 4, \mathcal{D}_2(1) = 5, \mathcal{D}_4(1) = 4$ at time 1, then the job $\mathcal{J}_2(1)$ reaches zero laxity at time 1 and the job set is scheduled by EDZL as shown in Figure 1(b). As a result, the latest completion time of jobs in $\mathcal{J}(0)$ becomes shorter than Figure 1(a).

In the above example, we found that an active job set can be completed with absolute deadline re-assignment. Moreover, the following conditions must be satisfied for the jobs to meet their deadlines. The re-assigned absolute deadline of $\mathcal{J}_i(t)$ by Algorithm 1 is denoted by $\mathcal{D}_i'(t)$.



(a) $\mathcal{D}_1(1) = 5, \mathcal{D}_2(1) = 12, \mathcal{D}_3 = 4, \mathcal{D}_4(1) = 8$



(b) $\mathcal{D}_1(1) = 4, \mathcal{D}_2(1) = 5, \mathcal{D}_3 = 4, \mathcal{D}_4(1) = 4$

Figure 1. An EDZL Schedule of the Example

• The density of each job with the re-assigned absolute deadline must be less than 1, *i.e.*,

$$\forall \mathcal{J}_i(t) \in \mathcal{J}(t), \lambda_i(t) \leq 1, \tag{13}$$

where

$$\lambda_i(t) = \frac{\mathcal{C}_i(t)}{\mathcal{D}_i'(t) - t}. \tag{14}$$

• The sum of $\lambda_i(t)$'s must be less than the number of processors, *i.e.*,

$$\sum_{\forall \mathcal{J}_i(t) \in \mathcal{J}(t)} \lambda_i(t) \leq m. \tag{15}$$

The following theorem provides a schedulability condition for EDZL with re-assigned absolute deadline.

Theorem 1 *If the absolute deadlines of jobs in an active job set $\mathcal{J}(t)$ are set by Algorithm 1 such that the $\mathcal{J}(t)$ satisfies Equation (13) and Equation (15), then all jobs in $\mathcal{J}(t)$ meet deadlines by the EDZL.*

Proof. We will show that if a job misses its deadline, then

$$\sum_{\forall \mathcal{J}_i(t) \in \mathcal{J}(t)} \lambda_i(t) > m.$$

Suppose that all jobs in $\mathcal{J}(t)$ have re-assigned deadlines by Algorithm 1 but a job $\mathcal{J}_i(t)$ in $\mathcal{J}(t)$ misses its deadline. Let t' be the time at which any job in $\mathcal{J}(t)$ has zero laxity for the first time after or at t, and $\mathcal{D}_i'(t)$ be the re-assigned deadline of $\mathcal{J}_i(t)$. Note that before t', the tasks are scheduled by the EDF. Without loss of generality, we can assume that

$\mathcal{D}_i'(t)$ is the shortest re-assigned deadline among jobs with zero laxity at time t', since the EDZL can break ties arbitrarily.

Let the job set that will have zero laxity at time t' be $\mathcal{J}'(t)$. Since the EDZL is work-conserving, if a job misses its deadline then there is no idle time before the deadline. So $m(t'-t)$ times are executed during the time interval $[t, t']$. Any job $\mathcal{J}_j'(t)$ executed during $[t, t']$ must have non-zero laxity and $\mathcal{D}_j'(t) \leq \mathcal{D}_i'(t)$ because jobs are scheduled by the EDF during the time interval $[t, t']$. Thus we have

$$\sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}(t) - \mathcal{J}'(t)} \mathcal{C}_j(t) \geq m(t'-t). \tag{16}$$

So

$$\sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}(t) - \mathcal{J}'(t)} \frac{\mathcal{C}_j(t)}{\mathcal{D}_j'(t)-t} \geq \sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}(t) - \mathcal{J}'(t)} \frac{\mathcal{C}_j(t)}{\mathcal{D}_i'(t)-t} \geq \frac{m(t'-t)}{\mathcal{D}_i'(t)-t}. \tag{17}$$

Because $\mathcal{J}_i(t)$ misses its deadline, there must be at least $m+1$ jobs with zero laxity at time t' when there are m processors. If the laxity of a job $\mathcal{J}_j(t')$ is zero, then we can see that $\mathcal{C}_j(t') = \mathcal{D}_j'(t) - t'$.

From the above observation, we have

$$\begin{aligned} \sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}(t)} \lambda_j(t) &= \sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}(t) - \mathcal{J}'(t)} \lambda_j(t) + \sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}'(t)} \lambda_j(t) \\ &\geq \frac{m(t'-t)}{\mathcal{D}_i'(t)-t} + \sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}'(t)} \frac{\mathcal{C}_j(t')}{\mathcal{D}_j'(t)-t}. \end{aligned}$$

Since the number of jobs in $\mathcal{J}'(t)$ is larger than m, it becomes

$$\begin{aligned} \sum_{\forall \mathcal{J}_j(t) \in \mathcal{J}(t)} \lambda_j(t) &\geq \sum_{\forall j \neq i, \mathcal{J}_j(t) \in \mathcal{J}'(t)}^{only\ m\ jobs} \left(\frac{t'-t}{\mathcal{D}_i'(t)-t} + \frac{\mathcal{C}_j(t')}{\mathcal{D}_j'(t)-t} \right) + \frac{\mathcal{C}_i(t')}{\mathcal{D}_i'(t)-t} \\ &\geq \sum_{\forall j \neq i, \mathcal{J}_j(t) \in \mathcal{J}'(t)}^{only\ m\ jobs} \left(\frac{t'-t+\mathcal{C}_j(t')}{\mathcal{D}_j'(t)-t} \right) + \frac{\mathcal{C}_i(t')}{\mathcal{D}_i'(t)-t} \\ &= \sum_{\forall j \neq i, \mathcal{J}_j(t) \in \mathcal{J}'(t)}^{only\ m\ jobs} \left(\frac{t'-t+\mathcal{D}_j'(t)-t'}{\mathcal{D}_j'(t)-t} \right) + \frac{\mathcal{C}_i(t')}{\mathcal{D}_i'(t)-t} > m \end{aligned}$$

which proves the theorem.

5. On-line DVS Algorithm: Determining Dynamic Speed

In this Section, we present dynamic speed calculation method based on Theorem 1. We claim that if

$$\sum_{\forall \mathcal{J}_i(t) \in \mathcal{J}(t)} \lambda_i(t) \leq m,$$

the dynamic processor speed at time t, denoted by $S_D(t)$, can be set to

$$\mathcal{S}_D(t) = \max\left(\frac{1}{m}\sum_{\forall \mathcal{J}_i(t) \in \mathcal{J}(t)} \lambda_i(t), \lambda_{\max}(t)\right) \tag{18}$$

for jobs with $\lambda_{\max}(t) \leq 1$, where

$$\lambda_i(t) = \frac{\mathcal{C}_i(t)}{\mathcal{D}_i'(t)-t}, \tag{19}$$

$\mathcal{D}_i'(t)$ is the absolute deadline re-assigned by Algorithm 1, and m is the number of processors. The following theorem shows that the processor speed given by Equation (18) guarantees to meet deadlines.

Theorem 2 *If absolute deadlines of jobs in $\mathcal{J}(t)$ are set by Algorithm 1 such that $\lambda_{max}(t) \leq 1$, then all jobs meet deadlines by EDZL with a dynamic speed $S_D(t)$ given by Equation (18) if*

$$\sum_{\forall \mathcal{J}_i(t) \in \mathcal{J}(t)} \lambda_i(t) \leq m \tag{20}$$

Proof. Let $\mathcal{J}'(t)$ be a modified job set of $\mathcal{J}(t)$ such that $\mathcal{J}_i(t)' \in \mathcal{J}'(t)$ is given by $(\mathcal{C}_i(t)', \mathcal{D}_i'(t))$ where $\mathcal{C}_i'(t) = \frac{\mathcal{C}_i(t)}{\mathcal{S}_i(t)}$. First, if

$$\frac{1}{m}\sum_{\forall \mathcal{J}_j \in \mathcal{J}(t)} \lambda_j(t) > \lambda_{\max}(t) \tag{21}$$

then

$$\lambda_i'(t) = \frac{\mathcal{C}_i'(t)}{\mathcal{D}_i'(t)\frac{1}{m}\sum_{\forall \mathcal{J}_j \in \mathcal{J}(t)} \lambda_j(t)} = \frac{m}{\sum_{\forall \mathcal{J}_j \in \mathcal{J}(t)} \lambda_j(t)} \cdot \frac{\mathcal{C}_i(t)}{\mathcal{D}_i'(t)}. \tag{22}$$

Otherwise,

$$\lambda_i'(t) = \frac{\mathcal{C}_i'(t)}{\lambda_{\max}(t)\mathcal{D}_i'(t)} \leq \frac{\mathcal{C}_i'(t)}{\mathcal{D}_i'\frac{1}{m}\sum_{\forall \mathcal{J}_j \in \mathcal{J}(t)} \lambda_j(t)} = \frac{m}{\sum_{\forall \mathcal{J}_i \in \mathcal{J}(t)} \lambda_j(t)} \frac{\mathcal{C}_i(t)}{\mathcal{D}_i'(t)}. \tag{23}$$

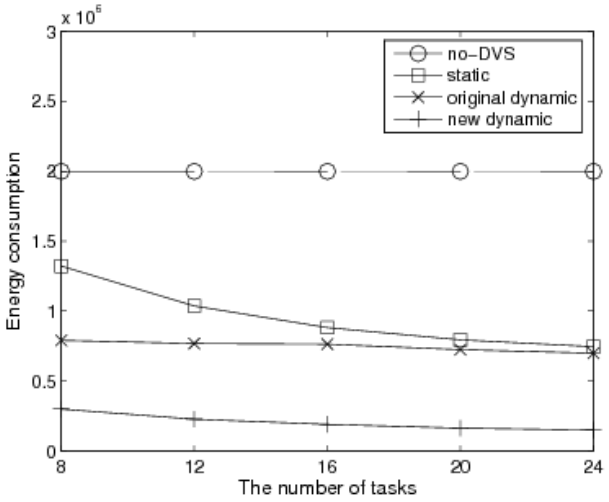
Therefore, we have

$$\begin{aligned} \sum_{\forall \mathcal{J}_i \in \mathcal{J}(t)} \lambda_i'(t) &\leq \frac{m}{\sum_{\forall \mathcal{J}_i \in \mathcal{J}(t)} \lambda_i(t)} \sum_{\forall \mathcal{J}_i \in \mathcal{J}(t)} \frac{\mathcal{C}_i(t)}{\mathcal{D}_i'(t)} \\ &\leq \frac{m}{\sum_{\forall \mathcal{J}_i \in \mathcal{J}(t)} \lambda_i(t)} \sum_{\forall \mathcal{J}_i \in \mathcal{J}(t)} \lambda_i(t) = m. \end{aligned}$$

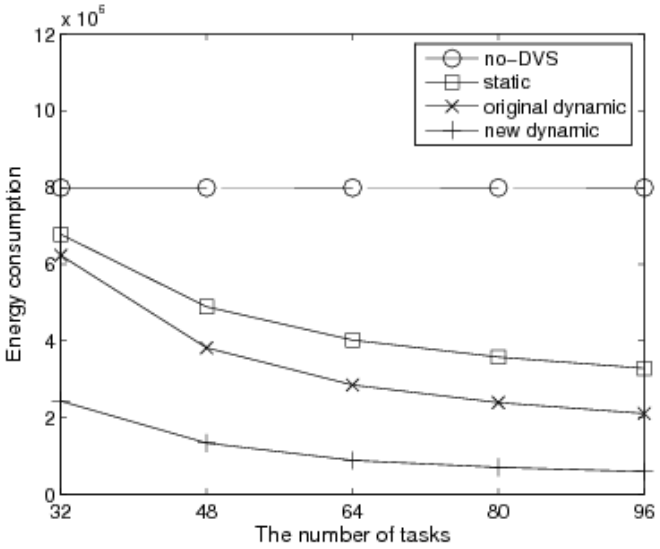
So by Theorem 1, the jobs in $\mathcal{J}'(t)$ meet their deadlines. ■

During scheduling, for an active job set set $\mathcal{J}(t)$ satisfying Equation (13) and Equation (15), our on-line algorithm computes a dynamic speed at each scheduling point, and determines dynamic speeds of the current jobs. Otherwise, the dynamic speeds of the jobs is set as a static speed which is obtained by the off-line algorithm proposed in [12].

6. Performance Evaluation



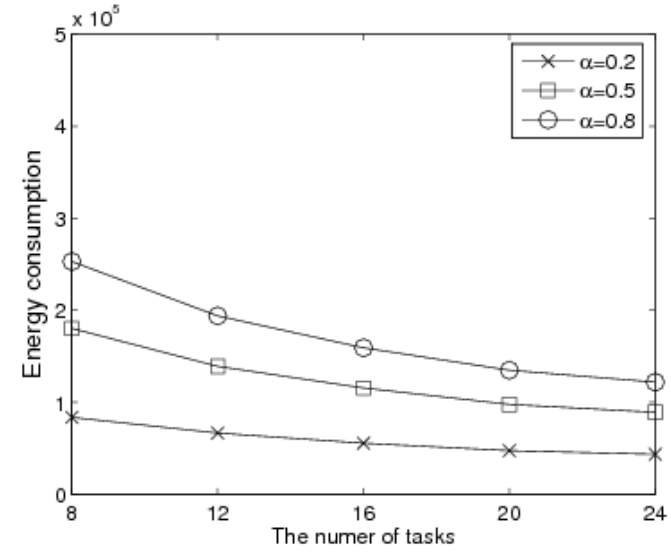
$(a)\ m = 4, U(\tau) = 2$



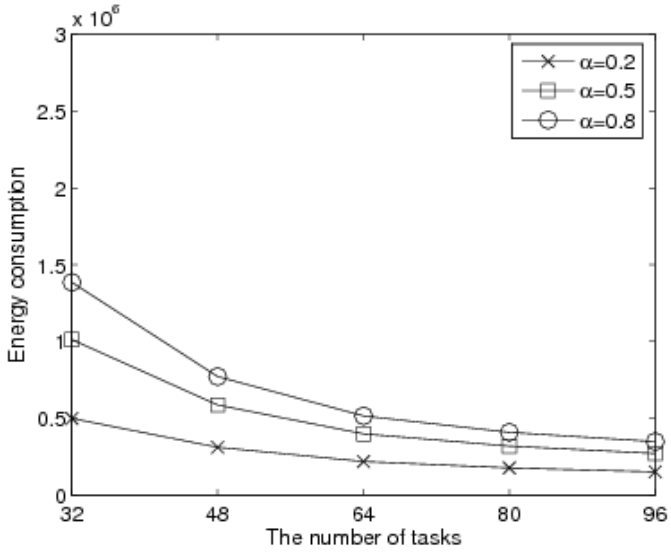
(b) $m = 16, U(\tau) = 8$

Figure 2. Simulation Results for Task Group G1 when Tasks Run with the Worst Case Execution Time

We evaluated the proposed on-line DVS algorithm by simulation. In the simulation, the period p_i of a task was given as a random number uniformly distributed in the range [100,1000], and the utilization U_i was uniformly distributed in the range [0.1,1). Then the worst-case execution time is given by $e_i = U_i p_i$. The energy consumption function is proportional to the cube of the processor speed for each time unit as modeled in [5]. The simulations were conducted on 4 and 16 processors for the following two cases: (G1) increasing the number of tasks with $U(\tau) = 2$ on $m = 4$, and $U(\tau) = 8$ on $m = 16$; (G2) increasing the total utilization with $n = 12$ on $m = 4$, and $n = 48$ on $m = 16$.



$(a)\ m = 4, U(\tau) = 2$



(b) $m = 16, U(\tau) = 8$

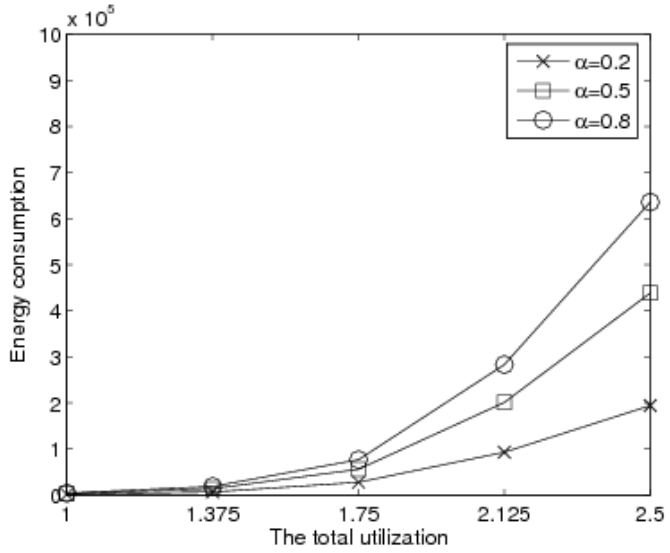
Figure 4. Simulation Results for task Group G1 when Tasks Run with the Actual Execution Times

(b) $m = 16, n = 48$

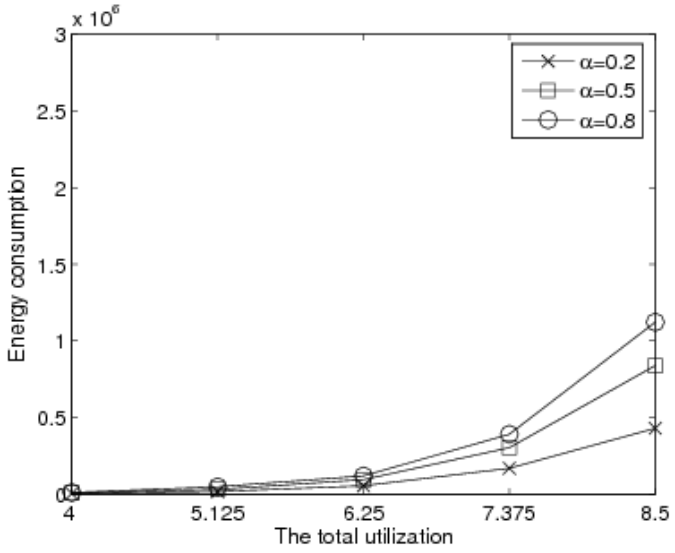
Figure 3. Simulation Results for Task Group G2 when Tasks Run with the Worst Case Execution Time

Firstly, we tested the performance when the tasks run with the worst case execution. We compared the energy consumption of the proposed algorithm with the algorithm proposed in [11]. Figure 2 and Figure 3 shows simulation results for G1 and G2 when the tasks run with the worst case execution. In the figures, *no-DVS* represents the energy consumption when all tasks run without DVS, *static* represents the energy consumption only with a static speed, *original dynamic* represents the energy consumption by the on-line algorithm proposed in [11], and *new dynamic* represents the energy consumption by the new on-line algorithm proposed in this paper. The DVS algorithms reduce energy more efficiently as the number of tasks is increased in Figure 2 where m and $U(\tau)$ are fixed. The reason is

that $U_{\max}(\tau)$ is decreased as the number of tasks increases when m and $U(\tau)$ are fixed. In this case, the original on-line algorithm (*original dynamic*) shows little enhancement than the off-line algorithm. On the other hand, the DVS algorithms become less efficient as the total utilization increases in Figure 3 where m and n are fixed. The reason is that $U_{\max}(\tau)$ is increased as the total utilization increases. However in both experiments of G1 and G2, the new on-line algorithm proposed in this paper outperforms other algorithms. At the best case, the new on-line algorithm reduces more power consumption than the original on-line algorithm 30% approximately.



$(a)\ m = 4, n = 12$



(b) $m = 16, n = 48$

Figure 5. Simulation results for task group G2 when Tasks Run with the Actual Execution Times

Secondly, the simulation was built on realistic assumption that the actual execution time of tasks is commonly less than its worst case execution time. In this simulation, the actual execution time of a task τ_i is given by $e_i\beta_i$, where β_i is normally distributed in the range of (0,1) with a mean value α and a standard

deviation 0.05. Figure 4 and Figure 5 shows simulation results for G1 and G2 when tasks run with the actual execution. Intuitively, the new on-line algorithm reduces more power consumption as the actual execution time is less than the worst case execution time in both experiments of G1 and G2. The influence of α on the power consumption decreases as the number of tasks increases as shown in Figure 4 when the total utilization is fixed. On the hand, the influence of α on power consumption increases as the total utilization increases as shown in Figure 5 when the number of tasks is fixed. We can guess it is because if the number of tasks is increased then the active job sets satisfying the Theorem 2 is increased (many jobs with small execution time), and as the total utilization is increased the number active job sets satisfying the Theorem 2 is decreased.

7. Conclusion

We have presented an on-line DVS algorithm for the EDZL scheduling to reduce power consumption of real-time tasks on multiprocessor platforms in which all processors run at an identical speed. The proposed algorithm determines a dynamic speed of processors at each scheduling point by re-assigning deadlines that makes tasks schedulable by the EDZL. Simulation results show that the proposed algorithm further reduce the power consumption than previous algorithms for the EDZL.

Acknowledgements

This work was supported by the Incheon National University Research Grant in 2014.

References

[1] D. Zhu, M. Rami and R. C. Bruce, “Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems,” IEEE Transactions on Parallel and Distributed System, vol. 14, **(2003)**, pp. 7.
[2] H. Aydin and Y. Qi, “Energy-Aware Partitioning for Multiprocessor Real-Time Systems,” Proceedings of the 17th International Parallel and Distributed Processing Symposium, Nice, France, April 22-26, **(2003)**.
[3] J. Chen and T. Kuo, “Multiprocessor Energy-Efficient Scheduling for Real-Time Tasks with Different Power Characteristics,” Proceedings of the International Conference on Parallel Processing, Olso, Norway, June 14-17, **(2005)**.
[4] C. Yang, J. Chen and T. Kuo, “An Approximation Algorithm for Energy-Efficient Scheduling on a Chip Multiprocessor,” Proceedings of the Conference on Design, Automation and Test, Munich, Germany, March 7-11, **(2005)**.
[5] J. Chen, C. Yang and T. Kuo, “Slack Reclamation for Real-Time Task Scheduling over Dynamic Voltage Scaling Multiprocessors,” Processing of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing Taichung, Taiwan, June 5-7, **(2006)**.
[6] S. Cho, S. Lee, S. Ahn and K. J. Lin, “Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems,” IEICE Transactions on Communications, vol. 85, **(2002)**, pp. 12.
[7] M. Cirinei and T. P. Baker, “EDZL Scheduling Analysis,” Proceedings of the 19th Euromicro Conference on Real-Time Systems, Pisa, Italy, July 4-6, **(2007)**.
[8] T. P. Baker, M. Cirinein and M. Bertogna, “EDZL Scheduling Analysis,” Real-Time Systems, vol. 40, **(2008)**, pp. 3.
[9] M. Park, S. Han, H. Kim, S. Cho and Y. Cho, “Comparison of Deadline-Based Scheduling Algorithms for Periodic Real-Time Tasks on Multiprocessor,” IEICE Transactions on Information and Systems, vol. 88, **(2005)**, pp. 3.
[10] H. Park, S. Han, H. Kim, S. Cho and Y. Cho, “ZL Scheme: Generalization of EDZL Scheduling Algorithm for Real-Time Multiprocessor Systems,” INFORMATION: An International Interdisciplinary Journal, vol. 8, **(2005)**, pp. 5.
[11] X. Piao, H. Kim, Y. Cho, S. Han, M. Park, M. Park and S. Cho, “Power-Aware EDZL Scheduling Upon Identical Multiprocessor Platforms,” Proceedings of International Conference on Reliable and Autonomous Computational Science, Atlanta, GA, USA, October 27-30, **(2010)**.
[12] X. Piao, H. Kim, Y. Cho, S. Han, M. Park, M. Park and S. Cho, “Low-Power Agorithm for EDZL Scheduling on Multicore Processors,” INFORMATION: An International Interdisciplinary Journal, vol. 14, **(2011)**, pp. 5.

[13] F. Zhang and S. T. Chanson, "Processor Voltage Scheduling for Real-Time Tasks with Non-Preemptible Sections," Proceedings of the Real-Time Systems Symposium Austin, TX, USA, December 3-5, **(2002)**.
[14] R. Davis and A. Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems," ACM Computing Surveys, vol. 43, **(2011)**, pp.4.
[15] C. L. Liu and J. W. Layland, "Scheduling Alorithms for Multiprogramming in A Hard Real-Time Environment," ACM, vol. 20, **(1973)**, pp. 1.
[16] J. Leung, "A New Algorithm for Scheduling Periodic Real-Time Tasks," Algorithmica, vol. 4, **(1989)**, pp. 1.
[17] A. Srinivasan and S. Baruah, "Deadline-Based Scheduling of Periodic Task Systems on Multiprocessors," Information Processing Letters, vol. 84, **(2002)**, pp. 2.
[18] S. Baruah, "Optimal Utilization Bounds for The Fixed-Priority Scheduling of Periodic Task Systems on Identical Multiprocessors," IEEE Transactions on Computers, vol. 53, **(2004)**, pp. 6.
[19] R. Ha and J. Liu, "Validating Timing Constraints in Multiprocessor and Distributed Real-Time Systems," Proceedings of the 14th IEEE International Conference on Distributed Computing Systems, Poznan, Poland, June 21-24, **(1994)**.
[20] X. Piao, S. Han, H. Kim, M. Park and Y. Cho, "Predictability of Earliest Deadline Zero Laxity Algorithm for Multiprocessor Real-Time Systems," Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing Gyeongju, Korea, April 24-26, **(2006)**.

Authors

Xuefeng Piao, He received the B.E. and M.E. degrees in Computer Science and Technology from Hoseo University in 2002 and 2004, respectively. He received Ph.D. degree in Computer Science and Technology from Seoul National University in 2011. He was a visiting researcher of Institute of Computer Technology in Seoul National University from 2011 to 2012. He is an Assistant Professor in School of Computer Science and Technology, Harbin Institute of Technology from 2011. His current research interests include operating systems, real-time systems, and embedded systems.

Moonju Park, He received the B.E. in Naval Architecture and Ocean Engineering, and the M.E. and the Ph.D. in Computer Engineering from Seoul National University in 1995, 1998, and 2002 respectively. From 2000 to 2001, he was a visiting scholar in University of Illinois at Urbana-Champaign. He was with LG Electronics as a chief research engineer from 2002 to 2006, and with IBM Ubiquitous Computing Laboratory as an advisory software engineer from 2006 to 2007. He is currently an Associate Professor in School of Computer Science and Engineering, Incheon National University. He served as a chairperson of embedded software group in ICT Standardization Committee of Korea from 2009 to 2013. His current research interests include operating systems, real-time scheduling, and embedded systems.

