Journal of Engineering and Applied Sciences 14 (21): 8039-8046, 2019
ISSN: 1816-949X

Dynamic Voltage/Frequency Scaling for EDZL Scheduling in Multicore Real-Time Systems

Sangchul Han, Minkyu Park and Woojin Paik
Department of Software Technology, Konkuk University,
268 Chungwondaero, Chungju-si, 27478 Chungcheongbuk-do, Korea
wjpaik@kku.ac.kr, +82-43-840-3598

Abstract: Under heavy workload mobile real-time systems make full use of system resources to meet timing constraints of tasks. Under light workload they need to reduce energy consumption due to limited system resources and a limited power source. EDZL (Earliest Deadline until Zero Laxity), one of multiprocessor real-time scheduling algorithms can enable higher system utilization than EDF (Earliest Deadline First) and its scheduling overhead can be restricted. However, there is little work on reducing energy consumption in EDZL scheduling. This study proposes a DVFS (Dynamic Voltage/Frequency Scaling) technique in EDZL scheduling. The technique includes two algorithms: one that calculates a uniform speed on full-chip DVFS platforms and the other that calculates task's individual speed on per-core DVFS platforms. We present simulation results that show our technique is simple but effective for reducing energy consumption in real-time scheduling.

Key words: Multicore, mobile real-time system, scheduling, EDZL, dynamic voltage/frequency scaling, DVFS

INTRODUCTION

Mobile embedded systems including wearable computers, smartphones and tablet PCs are powered by batteries. Since, batteries have a limited energy capacity, reducing energy consumption is a significant issue in designing mobile embedded systems to provide reliable operation, prolong service time and execute a wide range of applications. In recent decades many techniques have been developed to reduce energy consumption of architectural components such as display, processors and memory. The techniques include DVFS (Dynamic Voltage/Frequency Scaling) and power-aware scheduling-based techniques, DPM (Dynamic Power Management) using low power modes, microarchitectural energy-efficient techniques in components such as RAM, cache and TLB and techniques using unconventional-cores such as DSP or GPU (Mittal, 2014).

DVFS is a technique that reduces energy consumption in processors by altering processor supply voltage and/or frequency depending on the situation. Since, dynamic power dissipation of switching components decreases as the supply voltage decreases, the energy consumption of processors can be reduced by lowering the supply voltage (Chandrakasan *et al*., 1992). Many processor benders have developed DVFS technology for their products. For example, Intel's DVFS technology is EIST (Enhanced Intel SpeedStep). EIST is integrated into single/multi-core processors such as Intel Pentium series and Intel Core M series (IC., 2018). ARM's DVFS technology is IET (Intelligent Energy Management) which is contained in ARM1176JZF-S platform (Khan *et al*., 2012). AMD also, developed a DVFS technology power. Now, It is implemented in multicore processors such as AMD FX processors and Athlon processors (AMD., 2018).

The challenge of applying DVFS to embedded real-time systems is that lowering supply voltage/frequency causes real-time tasks to execute for a longer time. The prolonged execution time causes, in turn, a change in the schedule of real-time tasks. If the supply voltage/frequency is excessively lowered, the timing constraints of real-time tasks may not be satisfied, i.e., they may not complete their execution by their deadline in the changed schedule. Hence, in embedded real-time systems, DVFS techniques should be properly integrated with schedulers and the supply voltage and frequency should be altered prudently.

Nowadays, many mobile embedded systems adopt multicore processors to run high performance real-time applications like computer vision, speech/motion

Corresponding Author: Woojin Paik, Department of Software Technology, Konkuk University, 268 Chungwondaero, Chungju-si, 27478 Chungcheongbuk-do, Korea, wjpaik@kku.ac.kr, +82-43-840-3598

recognition software and so on. Since, these complex real-time apvplications consumes a considerable amount of energy, mobile embedded systems require effective processor scheduling methods for reducing energy consumption as well as satisfying the timing constraints.

There are numerous researches about energy-aware scheduling of real-time tasks on DVFS-enabled multicore platforms. Many of them such as Chen and Kuo (2005), Yang *et al*. (2005), Chen and Kuo (2007) and Aydin and Yang (2003), devise DVFS techniques in partitioning approach. In partitioning approach, tasks are partitioned into subsets and each subset is statically assigned to a processing core. Each subset is scheduled by a uniprocessor scheduling algorithm like Rate Monotonic (RM) (Liu and Layland, 1973), Earliest Deadline First (EDF) (Liu and Layland, 1973) and Deadline Monotonic (DM) (Audsley *et al*., 1991). The merit of partitioning approach is that it can deploy well-studied schedulability analysis and DVFS techniques for uniprocessor scheduling. The demerits are: task partitioning with minimized energy consumption is an NP-hard problem (Aydin and Yang, 2003). Since, the processor utilization in partitioning approach is not as high as global approach, some applications that are schedulable in global approach might not be schedulable in partitioning approach. Adding or removing tasks requires task set re-partitioning. On the other hand, there are some research works on DVFS techniques in global approach (Funaoka *et al*., 2008; Nelis *et al*., 2008; Funk *et al*., 2012; Han *et al*., 2015). In global approach, task migration is allowed, that is preempted tasks can be resumed on any available processing core. Global approaches can provide higher processor utilization than partitioning approach and tasks can be easily added or removed to/from a task set.

One of global real-time scheduling algorithms for multicore platforms is Earliest Deadline until Zero Laxity (EDZL) (Cho *et al*., 2002). EDZL assigns the highest priority to jobs (instance of tasks) whose laxity is zero. The priority of the remaining jobs is given according to EDF policy. The schedulability of EDZL is superior to many scheduling algorithms while its runtime overhead is acceptable. EDZL can schedule all task sets that are schedulable by global EDF (Park *et al*., 2005) and can schedule far more task sets than other EDF variants (Cirinei and Baker, 2007; Baker *et al*., 2008; Lee and Shin, 2013). In spite of the superiority of EDZL, there is little research work on DVFS techniques for EDZL algorithm.

In this study, we propose a DVFS technique for EDZL scheduling algorithm on multicore platforms. Based on the EDZL schedulability test presented by Lee and Shin (2013), our technique computes the processing core's speed that can guarantee the timing constraints of tasks while reducing their energy consumption. On full-chip DVFS platforms where all cores share one clock our technique computes static uniform speed all cores operate uniformly at the computed speed. On per-core DVFS platforms where each core contains individual clock our technique computes static individual speed for each task. We also, present simulation results that demonstrate that the proposed technique can effectively reduce the energy consumption of tasks in EDZL scheduling on multicore platforms.

System model: Multicore platforms consist of m identical processing cores P_1, P_2, …, P_m. On full-chip DVFS platforms, all processing cores share one clock and the clock frequency of all cores is the same at any time. On per-core DVFS platforms, each core contains an individual clock and the clock frequency of each core may be different from each other. The frequency range of clocks is $[f_{min}, f_{max}]$ and the execution speed of each core is defined as the ratio of the current clock frequency to f_{max}, that is if the current clock frequency of core P_i is f, the execution speed of P_i is $s_i = f / f_{max}$. The maximum speed of cores is $s_{max} = 1$ and the minimum speed is $s_{min} = f_{min}/f_{max}$. For example, if a job executes from time t_1 to time t_2 with a clock frequency $0.5 \times f_{max}$, the speed of the core during (t_1, t_2) is 0.5 and the amount of execution is $(t_2-t_1)\times 0.5$.

In CMOS digital circuits like microprocessor circuits, switching components is one of the main sources of power dissipation. The power dissipation of switching components, P, can be modelled as the following equation (Chandrakasan *et al*., 1992):

$$P = C_{ef} \cdot V_{dd}^{\ 2} \cdot f \quad (1)$$

Where:

C_{ef} = Constant that is determined by the characteristics of circuits
V_{dd} = Denotes the supply Voltage
f = Denotes the clock frequency

The relation between the supply voltage and the clock frequency is as follows:

$$\frac{1}{T_{delay}} = f \propto \frac{(V_{dd} - V_{th})^{\alpha}}{V_{dd}} \quad (2)$$

Where:

T_{delay} = The circuit delay
V_{th} = Threshold voltage which is much $< V_{dd}$
α = Constant dependent upon hardware characteristics and usually $\alpha = 2$ (Chandrakasan *et al*., 1992)

Thus, it is assumed that $f \propto V_{dd}$. From Eq. 1 and 2, we say $P \propto f^3$. Since, the execution speed of processing core, say s is proportional to f, we have $P \propto s^3$.

In this study, we consider scheduling a set of periodic tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task is represented by two parameters and is denoted by $\tau_i = (e_i, p_i)$. e_i is the worst-case execution time which is the amount of time for a processing core to fulfil the worst-case execution requirement of τ_i assuming it executes at speed s_{max}. p_i is a period; τ_i repeatedly generates a job (or an instance) at every period. Suppose τ_i generates job $\tau_{i,j}$ at time a. $\tau_{i,j}$ requires the execution of at most e_i time units (at speed s_{max}) and should be completed by absolute deadline $d_{i,j} = a+p_i$. The utilization of τ_i is $u_i = e_i/p_i$ and the total utilization of τ is $U(\tau) = \Sigma_{\tau i \in \tau} u_i$. We define $U_{max}(\tau) = \max \{u_i | \tau_i \in \tau\}$.

The amount of remaining execution of job $\tau_{i,j}$ at time t is denoted by $r_{i,j}(t)$. If $\tau_{i,j}$ executes all the remaining execution at speed s, the remaining execution time is $r_{i,j}(t)/s$. The laxity of a job is the amount of time for which the job can idle (or may not execute) without missing its deadline. This amount of time depends upon the speed at which the job executes during its remaining execution. Suppose $\tau_{i,j}$ executes its remaining execution at speed s, the laxity of $\tau_{i,j}$ at time t is defined as Eq. 3:

$$l_{i,j}(t, s) = d - t - \frac{r_{i,j}(t)}{s} \quad (3)$$

We assume preemption and migration are allowed. That is any job can be preempted at any time by a higher priority job and can be resumed on any available processing core later. A job cannot execute on more than one processing core at the same time and each processing core cannot execute more than one job at the same time.

EDZL scheduling: EDZL Cho *et al.* (2002) is a global scheduling algorithm for multiprocessor real-time systems. EDZL is a variant of EDF. It assigns the highest priority to jobs whose laxity is zero. For jobs with positive laxity, it assigns a higher priority to a job with the earlier deadline. There are some researches on the dominance and schedulability of EDZL. Park *et al.* (2005) proposed a utilization-based schedulability test for periodic task sets and showed that EDZL dominates EDF that is EDZL can successfully schedule any task set schedulable by EDF. Baker *et al.* (2008) extended, Cirinei and Baker (2007) and presented an iterative EDZL schedulability test which is called Baker's test. They demonstrated that the test is superior to existing EDF schedulability tests. Lee and Shin (2013) proposed another EDZL schedulability test (lemma 2) which is called Lee's test. This test utilizes Goossen's EDF schedulability test (lemma 1). They also, demonstrated through simulation that their test is superior to Baker's test in terms of the number of schedulable task sets.

DVFS in EDZL scheduling: Piao *et al.* proposed a DVFS technique for EDZL algorithm. This technique determines a uniform speed that reduces energy consumption on full-chip DVFS platforms. Based on Baker's test, the technique computes the minimum uniform speed such that the number of tasks with zero laxity is less than or equal to m. Our technique presented in this research is different from Piao's technique in the following respects. First, our technique is based on Lee's test. This makes our technique simpler and more effective because Lee's test is simpler but tighter than Baker's test. Second, we propose an algorithm for per-core DVFS platforms. The algorithm can determine the speed of individual task when each core contains individual clock.

Lemma 1: Goossens *et al.* (2003) periodic task system τ can be EDF-schedulable upon m unit-speed identical processors, provided its cumulative utilization is bounded from above as follow:

$$U(\tau) \leq m - U_{max(\tau)} \cdot (m - 1) \quad (4)$$

Lemma 2: Lee and Shin (2013) On m-core platforms, a task set τ is schedulable by EDZL, if, there exists $m^*(= 1, 2, \ldots, m)$ satisfying the following equation:

$$\sum_{\tau i \in T_1} u_i \leq m^* - (m^* - 1) \cdot \max \langle u_j | \tau_j \in T_1 \rangle \quad (5)$$

where, $T_1 = \{\tau_i \in \tau \mid \tau_i \notin (m\text{-}m^*)$ task with the largest $u_i\}$. Note that $T_1 = \tau - T_2$ where, T_2 is a set of $(m\text{-}m^*)$ tasks whose utilization is highest. In other words, T_1 is a set of $(n\text{-}m+m^*)$ tasks whose utilization is lowest.

MATERIALS AND METHODS

Full-chip DVFS algorithm: This section proposes an algorithm that determines a uniform speed for EDZL scheduling on full-chip DVFS platforms where all cores share one clock, so that, the cores operate at the same speed at any time. This algorithm is based on Lee's EDZL schedulability test (lemma 2). The following theorem shows that there exists a uniform speed with which a task set is schedulable by EDZL, if the task set satisfies Eq. 5.

Theorem 1: Suppose a task set τ is scheduled by EDZL on m-core platforms. If there exists m^* that satisfies Eq. 5, then, τ is schedulable with a uniform speed S($0<S\leq 1$) that satisfies the following equation:

$$S \geq \max\left\{U_{max(\tau)}, \frac{1}{m^*}\left(U(T_1)+(m^*-1)\cdot U_{max}(T_1)\right)\right\}$$

Proof: It is obvious that S should be no $<U_{max}(\tau)$. Let us consider a task set $\tau' = \{\tau_1' = (e_1/S, p_1), \tau_2' = (e_2/S, p_2), ..., \tau_n' = (e_n/S, p_n)\}$ where, the worst-case execution time is scaled by 1/S. Let, u_i' denote the utilization of τ_i'. Then, $u_i' = u_i/S$. Let, T_1' be a subset of τ' such that $T_1' = \{\tau_i' \in \tau' \mid \tau_i' \notin (m-m^*)$ task with the largest $u_i'\}$. Then:

$$\sum_{\tau_i' \in T_1'} u_i' = \sum_{\tau_i \in T_1} \frac{u_i}{S} \leq \frac{\sum_{\tau_i \in T_1} u_i}{\frac{1}{m^*\left(U(T_1)+(m^*-1)U_{max(T_1)}\right)}} \leq \frac{U(T_1)}{\frac{1}{m^*\left(U(T_1)+(m^*-1)U_{max(T_1)}\right)}} = \frac{U(T_1)+(m^*-1)U_{max(T_1)}-(m^*-1)U_{max(T_1)}}{m^*\left(U(T_1)+(m^*-1)U_{max(T_1)}\right)} = m^*-\frac{(m^*-1)U_{max}(T_1)}{\frac{1}{m^*\left(U(T_1)+(m^*-1)U_{max}(T_1)\right)}} \leq m^*-\frac{(m^*-1)\max\{u_j \mid \tau_j \in T_1\}}{S} = m^*-(m^*-1)\max\left\{\frac{u_j}{S} \mid \tau_j \in T_1\right\} = m^*-(m^*-1)\max\{u_j' \mid \tau_j' \in T_1'\}$$

By lemma 2, τ' is schedulable by EDZL on m processing cores with the maximum speed. If the speed of all the processing cores is S, the execution time of τ_i will be e_i/S. The EDZL schedule of τ with speed S is equivalent to the EDZL schedule of τ' with the maximum speed. Since, τ' is schedulable by EDZL with the maximum speed as shown above, τ is also, schedulable by EDZL with speed S, Y'.

Based on theorem 1, algorithm 1 computes the minimum uniform speed for a given task set. For example, consider $\tau = \{(1, 12), (1, 6), (1, 2), (2, 3)\}$ on 2-core full-chip DVFS platform. When $m^* = 1$, $T_1 = \{(1, 12), (1, 6), (1, 2)\}$ and $S = \max\{0.6667, 0.75\} = 0.75$. When $m^* = 2$, $T_1 = \{(1, 12), (1, 6), (1, 2), (2, 3)\}$ and $S = \max\{0.6667, 1.0417\} = 1.0417$ which does not satisfies $0<S\leq 1$. Hence, the minimum uniform speed for τ is $S_{min} = 0.75$ when $m^* = 1$. Since, Algorithm 1 calculates the sum of task utilizations for each m^*(= 1, 2, ..., m), its time complexity is O(mn).

Algorithm 1; Calculate a minimum uniform speed:

```
function_calc uniform_speed(m, τ)
1    S_min = 1
2    for m* from 1 to m do
3        T_1 ← {τ_i ∈ τ | τ_i ∈ m-m* tasks with the largest u_i}
4        s ← max{U_max(τ), 1/m*(U(T_1)+(m*-1)U_max(T_1))}
5        S_min ← min{S_min, s}
6    done
7    return S_min
```

Per-core DVFS algorithm: This study proposes a technique that computes task's individual speed, denoted by S_1, S_2, ..., S_n for EDZL scheduling on per-core DVFS platforms where each core contains individual clock and its speed can be adjusted individually. On this platform, when a job is about to execute on a core, the core's speed is adjusted to the individual speed of the task of the job. Suppose that there exists m^* that satisfies Eq. 5 for task set τ. We assign $(m-m^*)$ cores to $(m-m^*)$ tasks with the largest u_i, i.e., to tasks in $T_2 (= \tau-T_1)$. This is done by setting the individual speed of tasks in T_2 as the same value as their utilization, i.e., $S_i = u_i$ for $\tau_i \in T_2$. Since, the laxity of jobs of these tasks at their arrival is 0 by definition (Eq. 3), the jobs are given the highest priority and executed on $(m-m^*)$ cores exclusively.

The remaining $(n-m+m^*)$ tasks, i.e., T_1's tasks, execute on m^* cores. Since, T_1 satisfies Eq. 4, T_1 is EDF-schedulable on m^* cores by lemma 1. Since, any task set schedulable by EDF is also, schedulable by EDZL (Park *et al*., 2005), T_1 is also, EDZL-schedulable on m^* cores. We set the individual speed of tasks in T_1 as the uniform speed of T_1 on m^* cores. For $\tau \in T_1$, S_i = calc_uniform_speed (m^*, T_1).

Algorithm 2 finds such m^* that minimizes the uniform speed for T_1 and determines the individual speed of each task. Since, it invokes calc_uniform speed() for each m^*, its time complexity is $O(m^2n)$.

Example: Consider a task set $\tau = \{\tau_1 = (6, 10), \tau_2 = (2, 4), \tau_3 = (1, 5), \tau_4 = (2, 20)\}$ on a per-core DVFS platform with 3 cores. Equation 5 is satisfied when $m^* = 1$. Algorithm 2 determines the individual speed of tasks as follows: $T_1 = \{\tau_3, \tau_4\}$ and the uniform speed of T_1 on 1 core is 0.3. The speed of τ_1 and τ_2 is set to the same value as their utilization, respectively. Thus, the individual speed is $S_1 = 0.6$, $S_2 = 0.5$ and $S_3 = S_4 = 0.3$.

Algorithm 2; Calculate individual speeds:

```
function_calc_individual speed(m, τ)
1    S_min = 1, m_min* = m
2    for m* from 1 to m do
3        T_1 ← {τ ∈ τ |τ ∉ m-m* tasks with the largest u_i}
4        if Σ_τi∈T1 u_i = m*-(m*-1)•U_max(T_1) then
5            s←calc_uniform_speed(m ,T_1)
6            if s < S_min then
7                S_min- s, m_min* ← m*
8            fi
9        fi
10   done
11   T_1 ← {τ_i∈τ |τ_i∉m-m_min* tasks with the largest u_i}
12   for τ_i∈T_1 do S_i←S_min done
13   for τ_i ∉τ- T_1 do S_i u_i done
14   return S_1, S_2, ..., S_n
```

Algorithm 1 vs. 2: Now, we compare the energy efficiency of the proposed algorithms in terms of task execution speed. We show that, the execution speed of each task determined by Algorithm 2 is always lower than that of Algorithm 1.

Let us define $T_1(M) = \{\tau_i \in \tau | \tau_i \notin m\text{-}M$ tasks with the largest $u_i\}$ and $S^u(M)$ = calc_uniform_speed(M, $T_1(M)$). It is trivial that $T_1(m) = \tau$ and $S^u(m)$ is the uniform speed of τ on m cores determined by Algorithm 1.

Theorem 2: Suppose a task set τ is EDZL-schedulable on m-core platform. The individual speed determined by Algorithms 2 is lower than the uniform speed determined by Algorithm 1 for each task.

Proof: Suppose $S^u(M)$ is minimized when $M = m^*_{min}$ in Algorithm 2. Then, the individual speed S_i can be denoted as follows:

$$S_i \begin{cases} S^u\left(m^*_{min}\right) & \text{for } \tau_i \in T_1\left(m^*_{min}\right) \\ u_i & \text{for } \tau_i \notin T_1\left(m^*_{min}\right) \end{cases}$$

In case $\tau_i \in T_1(m^*_{min})$, $S^u(m^*_{min})$ is the minimum among $S^u(M)$ for M = 1, 2, ..., m. Hence, $Si = S^u(m^*_{min}) \leq S^u(m)$. In case $\tau_i \notin T_1(m^*_{min})$, from Algorithm 1, it is obvious that $S^u(m) \geq U_{max}(\tau)$. Since, $u_i \leq U_{max}(\tau)$ for all $\tau_i \in \tau$, $S_i = u_i = U_{max}(\tau) \leq S^u(m)$. Therefore, for every $\tau_i \in \tau$, the individual speed S_i is always lower than or equal to the uniform speed $S^u(m)$ ÿ.

RESULTS AND DISCUSSION

We evaluate the proposed DVFS algorithms through periodic task scheduling simulations. We generate periodic task sets as follows. For each m (= 4, 8, 16) that denotes the number of cores, we vary the total utilization of task sets in the range of 0.25-0.90 m with a step of 0.2. Then, for each total utilization we generate 100 task sets. The period (p_i) and utilization (u_i) of tasks are randomly chosen from a uniform distribution over (10, 1000] and (0.1, 1], respectively. The worst-case execution time (e_i) of a task is given by $p_i \times u_i$. The total number of task sets, we generated is 9,400. The schedulability of every task set is tested using both Baker's test (Baker *et al*., 2008) and Lee's (Lee and Shin, 2013). If a task set does not pass both tests or the number of tasks in a task set is <m, we reject the task set.

Table 1: Processor characteristics

Frequency (MHz)	Voltage (V)	Speed	Power (mW)
1000	1.80	1.00	1600
800	1.60	0.80	900
600	1.30	0.60	400
400	1.00	0.40	170
150	0.75	0.15	80

Table 2: Nomenclatures

Symbols	Discription
τ	Periodic task set
τ_i	ith task
$\tau_{i,j}$	jth job of ith task
e_i	Worst-case execution time of task τ_i
m	Number of processing cores
n	Number of tasks
p_i	Period of task τ_i
S	Uniform speed for periodic task set τ
$S^u(M)$	Minimum uniform speed for (n-m+M) tasks whose utilization is lowest
S_{min}	Minimum uniform speed for periodic task set τ
S_i	Individual speed of task τ_i
s_i	Execution speed of core P_i, $(0<s_i\leq 1)$
T_1	Set of (n-m+m*) tasks whose utilization is lowest. m is a parameter
T_2	Set of (m-m*) tasks whose utilization is highest. $T_2 = \tau - T_1$
$U(\tau)$	Total utilization of periodic task set τ
$U_{max}(\tau)$	Maximum utilization of tasks in periodic task set τ
u_i	Utilization of periodic task $\tau_{i,j}$, $(= e_i/p_i)$

The processor model of this simulation is Intel Xscale processor. Table 1 shows its characteristics. In this model the processor's frequency and supply voltage is altered to one of levels listed in Table 1. The table also, shows the speed and power of each voltage level. In our simulation when we alter the speed of a core to S, the frequency and supply voltage of the core are adjusted to the lowest level whose speed is higher than or equal to S. For example, when a job with individual speed 0.73 is about to execute on a core, the core's speed is adjusted to 0.8 (1.6 V-800 MHz-level). When the job finishes or is preempted, the energy consumption is calculated using the speed and power value of the level. If a job executes for C time units and the speed and power value of the voltage/frequency level are S and P, respectively, the amount of energy consumed is calculated by (C/S)P Table 2.

For each task set, we estimate the energy consumption of all tasks in the task set for hyper-period (the least common multiple of task periods) and normalize it to the energy consumption without any DVFS technique. Then, we average the normalized energy consumption of task sets of a total utilization. Figure 1-3

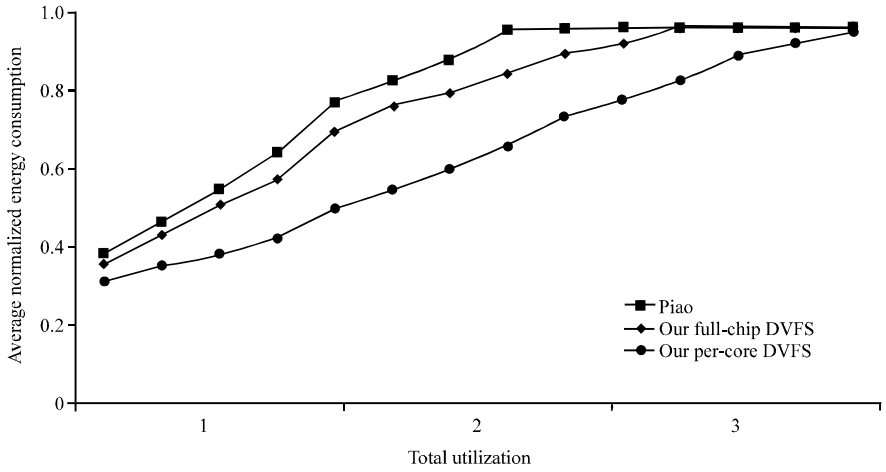

Fig. 1: Average normalized energy consumption when m = 4

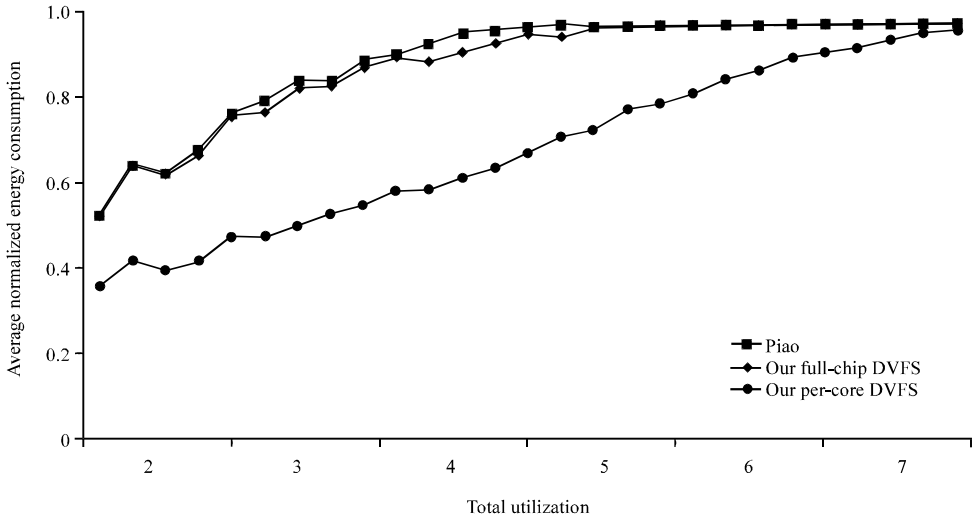

Fig. 2: Average normalized energy consumption when m = 8

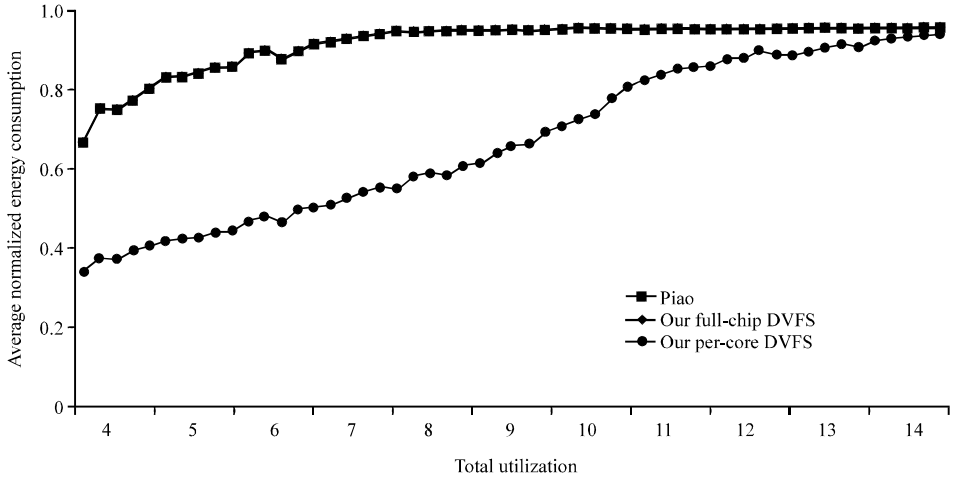

Fig. 3: Average normalized energy consumption when m = 16

show the average normalized energy consumption of our algorithms and Piao's for m = 4, 8, 16, respectively. Our algorithms are denoted by our full-chip DVFS (Algorithm 1) and our per-core DVFS (Algorithm 2). To the best of our knowledge, there is no research work on DVFS techniques in EDZL scheduling other than Piao's research. Thus, we compare our technique with Piao's, which is a full-chip DVFS technique.

As shown in Fig. 1-3, our per-core DVFS algorithm can save a significant amount of energy on per-core DVFS platform. And our full-chip DVFS algorithm can save a little more energy than Piao's. For instance, when m = 4 and the total utilization is 2.0, our per-core DVFS algorithm saves 41.5%, our full-chip DVFS one saves 20.1% and Piao's saves 13.5% of energy in average.

For a fixed number of cores, on the whole, more energy can be saved as the total utilization of a task set gets lower. When m is 4 and the total utilization is 1 (= 0.25 m), the average normalized energy consumption of the three algorithms is 35~42% which implies that 58~65% of energy can be saved in average. For task sets with high total utilization, the algorithms can save less energy because there is less room to disperse task execution.

The normalized energy consumption of our per-core DVFS algorithm increases almost proportionally to task set's total utilization. Meanwhile, when the total utilization becomes higher than a certain value, the normalized energy consumption of our full-chip DVFS algorithm and Piao's one approach to 1, i.e., there is little energy saving. As the total utilization increases, it is more likely that there will be a high-utilization task in random task sets. Since, $U_{max}(\tau)$ is a dominant factor in determining a uniform speed (note that a uniform speed should be greater than or equal to $U_{max}(\tau)$), the performance of uniform speed techniques may degrade owing to one high-utilization task.

CONCLUSION

In this study, we propose a dynamic voltage/frequency scaling technique for EDZL, a global real-time scheduling algorithm. We present two algorithms based on an existing EDZL schedulability test. Our full-chip DVFS algorithm computes a uniform speed all tasks in a task set execute at the computed uniform speed on full-chip DVFS platforms. Our per-core DVFS algorithm calculates individual speed of each task, each task executes at its own individual speed on per-core DVFS platforms.

The simulation results show that full-chip DVFS techniques including our algorithm and Piao's can save significant energy only, if there is no high-utilization task. We can say that full-chip DVFS platforms suffer from high-utilization task. Since, a uniform speed should be greater than or equal to $U_{max}(\tau)$, the uniform speed will be high, if a high-utilization task exists in a task set, resulting in little energy saving. Per-core DVFS platforms do not have such a problem. On per-core DVFS platforms, if scheduling algorithms and DVFS algorithms are properly coupled much more energy can be saved and the energy consumption increases moderately as the total utilization increases.

In our experiment we assume that all tasks require worst-case execution. In most cases, however, the actual execution time of real-time tasks is less than their worst-case execution time. Many techniques, called online DVFS techniques, make good use of this feature. And they show that significant further energy saving is possible. We plan to extend our research to devise online DVFS algorithms for EDZL scheduling.

ACKNOWLEDGEMENT

This study was supported by Konkuk University in 2018.

REFERENCES

AMD., 2018. AMD Radeon VII the worlds first 7nm gaming GPU. Advanced Micro Devices, Inc., Santa Clara, California, USA. http://www.amd.com/en

Audsley, N.C., A. Burns, M.F. Richardson and A.J. Wellings, 1991. Hard real-time scheduling: The deadline-monotonic approach 1. IFAC. Proceed. Vol., 24: 127-132.

Aydin, H. and Q. Yang, 2003. Energy-aware partitioning for multiprocessor real-time systems. Proceedings of the 2003 17th International Symposium on Parallel and Distributed Processing, April 22-26, 2003, IEEE, Nice, France, pp: 1-9.

Baker, T.P., M. Cirinei and M. Bertogna, 2008. EDZL scheduling analysis. Real Time Syst., 40: 264-289.

Chandrakasan, A.P., S. Sheng and R.W. Brodersen, 1992. Low-power CMOS digital design. IEEE. J. Solid State Circuit, 27: 473-484.

Chen, J.J. and C.F. Kuo, 2007. Energy-efficient scheduling for real-time systems on Dynamic Voltage Scaling (DVS) platforms. Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), August 21-24, 2007, IEEE, Daegu, South Korea, ISBN:978-0-7695-2975-2, pp: 28-38.

Chen, J.J. and T.W. Kuo, 2005. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. Proceedings of the 2005 International Conference on Parallel Processing (ICPP'05), June 14-17, 2005, IEEE, Oslo, Norway, pp: 13-20.

Cho, S., S.K. Lee, S. Ahn and K.J. Lin, 2002. Efficient real-time scheduling algorithms for multiprocessor systems. IEICE. Trans. Commun., 85: 2859-2867.

Cirinei, M. and T.P. Baker, 2007. EDZL scheduling analysis. Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS'07), July 4-6, 2007, IEEE, Pisa, Italy, ISBN:0-7695-2914-3, pp: 9-18.

Funaoka, K., A. Takeda, S. Kato and N. Yamasaki, 2008. Dynamic voltage and frequency scaling for optimal real-time scheduling on multiprocessors. Proceedings of the 2008 International Symposium on Industrial Embedded Systems, June 11-13, 2008, IEEE, Le Grande Motte, France, ISBN:978-1-4244-1994-4, pp: 27-33.

Funk, S., H.O. Chiahsun, V. Berten and J. Goossens, 2012. A global optimal scheduling algorithm for multiprocessor low-power platforms. Proceedings of the 20th International Conference on Real-Time and Network Systems, November 8-9, 2012, ACM, New York, USA., ISBN:978-1-4503-1409-1, pp: 71-80.

Goossens, J., S. Funk and S. Baruah, 2003. Priority-driven scheduling of periodic task systems on multiprocessor. Real-Time Syst., 25: 187-205.

Han, S., M. Park, X. Piao and M. Park, 2015. A dual speed scheme for dynamic voltage scaling on real-time multiprocessor systems. J. Supercomputing, 71: 574-590.

IC., 2018. Intel product specifications. Intel Corporation, Santa Clara, California, USA. https://ark.intel.com/content/www/us/en/ark.html

Khan, J., S. Bilavarn and C. Belleudy, 2012. Energy analysis of a DVFS based power strategy on arm platforms. Proceedings of the 2012 IEEE International Conference on Faible Tension and Consommation, June 6-8, 2012, IEEE, Paris, France, ISBN: 978-1-4673-0822-9, pp: 1-4.

Lee, J. and I. Shin, 2013. Edzl schedulability analysis in real-time multicore scheduling. IEEE. Trans. Software Eng., 39: 910-916.

Liu, C.L. and J.W. Layland, 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM., 20: 46-61.

Mittal, S., 2014. A survey of techniques for improving energy efficiency in embedded computing systems. Intl. J. Comput. Aided Eng. Technol., 6: 440-459.

Nelis, V., J. Goossens, R. Devillers, D. Milojevic and N. Navet, 2008. Power-aware real-time scheduling upon identical multiprocessor platforms. Proceedings of the 2008 IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (sutc 2008), June 11-13, 2008, IEEE, Taichung, Taiwan, pp: 209-216.

Park, M., S. Han, H. Kim, S. Cho and Y. Cho, 2005. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. IEICE. Trans. Inf. Syst., 88: 658-661.

Yang, C.Y., J.J. Chen and T.W. Kuo, 2005. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. Proceedings of the 2005 International Conference on Design, Automation and Test in Europe, March 7-11, 2005, IEEE, Munich, Germany, pp: 468-473.