ISSN 2347 - 3983
Volume 8. No. 8, August 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter27882020.pdf
https://doi.org/10.30534/ijeter/2020/27882020

A Simple and Aggressive Dynamic Voltage/Frequency Scaling Technique for EDZL Scheduling on Multiprocessors

Sangchul Han
Dept. of Software Technology, Konkuk University, Korea, schan@kku.ac.kr

ABSTRACT

One of significant energy-saving techniques for processor is DVFS (Dynamic Voltage/Frequency Scaling).In real-time embedded systems, DVFS schemes coupled with task schedulers control the execution speed of tasksas low as possible while satisfying the tasks' timing constraints. This article proposes an aggressive DVFS technique for EDZL (Earliest Deadline until Zero Laxity) scheduling. Taking advantage of the gap between actual execution amount and worst-case execution demand, our technique lowers the execution speed of tasks aggressively. It starts a task at a low speed. The speed is raised later if the actual execution amount of the task exceeds half of the worst-case execution demand in order not to violate its timing constraints. We show through simulation that our technique can reduce the energy consumption.

Key words :DVFS, EDZL Scheduling, Real-Time Systems, Multiprocessor

1. INTRODUCTION

Since the capacity of battery is limited, energy-saving is one of significant concerns in battery-powered embedded systems. Many techniques have been devised to reduce energy consumption of computer components such as processor, memory, display, etc[23]-[25].To reduce energy consumption within processor, recent processors provide the capability of changing the supply voltage/frequency, which is called DVFS (dynamic voltage/frequency scaling). Lowering the supply voltage/frequency causes a processor to execute tasks slower but to consume much less energy. Examples of DVFS technology are EIST(by Intel), IET (by ARM), and PowerNow!(by AMD)[3]-[5].

Theissue of employing DVFS in real-time embedded systems is that if the execution speed of real-time task is excessively low the task may fail to meet its timing constraints. The execution speed of real-time tasks should be controlled carefully.In real-time embedded systems, DVFS schemes need to be tightly coupled with task scheduler and adjust processor speed as low as possible while satisfying the tasks' timing constraints, i.e. their deadline.

There are many researches on DVFS technique for real-time embedded systems equipped with multiprocessors (or multicore). Well-known DVFS techniques in partitioned scheduling are[6]-[9]. In partitioned scheduling, several tasks are mapped onto a processor (or a core) and executed only on that core. Each processor deploys uniprocessor scheduling algorithm such as[10] and [11]. DVFS techniques in global scheduling are [12]-[16]. In global scheduling, tasks can execute on any available core at any time. The advantage of global scheduling over partitioned scheduling is that it can achieve higher processor utilization and it does not need repartitioning.

A global scheduling algorithm for multiprocessor real-time systems is Earliest Deadline until Zero Laxity (EDZL) [17]. In EDZL scheduling, jobs (task instances) whose laxity is zero are executed immediately. The remaining jobs with positive laxity are given a priority following EDF (Earliest Deadline Frist) policy. Any EDF-schedulable task set is also schedulable by EDZL [18], and EDZL's schedulability is better than other EDF variants [19]-[21].Some research work has studied DVFS techniques for EDZL scheduling. [14] and [22] proposed a task-level static scheme for periodic task set where speed of task is determined before execution. [26] proposed a job-level static scheme where speed of job is determined when the job is released, and the speed is not altered during execution.

In this paper, we propose a simple and aggressive job-level dynamic DVFS technique for EDZL scheduling algorithm on multicore platforms. Taking advantage of the gap between actual execution amount and worst-case execution demand, our technique starts a job at a low speed. The speed is raised later if the actual execution amount of the job exceeds half of the worst-case execution demand in order not to violate its timing constraints. If a job uses much less processor time than its worst-case execution demand, the job can complete its execution at the low speed. The job's energy consumption can be reduced safely. We show through simulation that our technique is energy-efficient in scheduling real-time tasks.

2. SYSTEM MODEL

The system is composed of m processing cores which of each have an individual clock. The clock frequency of cores can be controlled independently. The current speed of core isf / f_{max} where f is the current frequency and f_{max} is the maximum frequency. The range of core speed is $[f_{min}/f_{max}, 1]$ where $f_{min}$is the minimum frequency. For an instance, if a job is executed on a core from t_1 to $t_2$atspeed0.7, the amount of execution is $(t_2 - t_1) \times 0.7$.

The power dissipation of core is modelled as $P \propto V^2 f$, where f is the frequency and V is the supply voltage [2]. When we lower the execution speed of core, we usually decrease both frequency and supply voltage. As the execution speed decreases, the power dissipation declines dramatically.

In our task model, a task τ_i is denoted by a tuple of a period $_i$ and a worst-case execution time e_i, i.e., $\tau_i = (e_i, p_i)$. e_i is the amount of time to fulfil the worst-case execution demand of τ_i assuming it always executes at speed 1. A task generates jobs periodically. $\tau_{i,j}$ denotes the j-th job generated by τ_i at $j \cdot p_i$ (j=0,1,2,…). $\tau_{i,j}$ demands at most e_i time unit processor time. It should finish its execution by time $d_{i,j} = p_i(j+1)$ which is called absolute deadline. τ denotes a set of n tasks, $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. τ_i's utilization is defined by $u_i = e_i/p_i$ and τ_i's total utilization is defined by $U(\tau) = \Sigma u_i$.

We define $\tau_{i,j}$'s remaining execution at time t as $r_{i,j}(t)$. 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 defined as the maximum 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, its laxity at time t is as the equation below.

$$l_{i,j}(t, s) = d - t - \frac{r_{i,j}(t)}{s} \quad (1)$$

3. DVFS TECHNIQUES FOR EDZL SCHEDULING

3.1 Related Work

EDZL is a simple but effective multiprocessor real-time scheduling algorithm. By adding a simple barrier condition, *zero laxity*, to global EDF scheduling, it overcomes the multiprocessor anomaly of global EDF and achieves a higher utilization. A simple utilization-based schedulability test is presented in [18] and a precise schedulability test is proposed in [19]-[20]. Another schedulability test is proposed by Lee et al [21]. We call this test as Lee's test in this article. [21] showed that Lee's test is tighter than other EDZL schedulability tests.

There are some studies on DVFS technique for EDZL scheduling. [14] proposed a task-level static technique that computes a uniform speed to which the speed of all tasks can be safely altered to reduce energy consumption. [22] proposed another task-level static technique. It computes a uniform speed using Lee's test and calculates an individual speed of task based on the uniform speed. Every job belonging to a task runs at the individual speed, but jobs belonging to different task may execute at different speed .It was shown that the determined individual speed is always lower than or equal to the uniform speed. [26] proposed a job-level static DVFS technique. By combining the individual speed of [22] with MOTE [1], if a job executes for a shorter time than its worst-case execution demand, the speed of its next job on the same core is lowered utilizing the unused processor time. The technique is job-level static in the sense that the execution speed of job is determined at its release time and the speed does not change until the job completes. Different jobs generated by a task may have different speed.

3.2A Job-Level Dynamic DVFS Technique

This section presents a simple and aggressive job-level dynamic DVFS technique for EDZL scheduling. Our technique is based on Han's individual speed [22]. It aggressively lowers the execution speed of *heavy* tasks, specifically whose utilization is greater than 0.5. The execution speed of *light* tasks is always its individual speed S_i. When a heavy job (an instance of heavy task) is released, the job starts execution at a low speed $S_i^L = S_i - (1 - S_i)$. The job keeps this speed while its accumulated processor time is less than half of the worst-case execution demand. We call this time instance as *half time*. At that time, the speed of the job is raised to $S_i + (1 - S_i)$ and kept until the job finishes. The total processor time used by this job does not exceed the amount of time allowed to this job in case of the worst-case execution at speed S_i. Hence it does not hinder other jobs from meeting deadline. If the job finishes earlier than its half time, it is sure that the energy consumption of the job is less than would be otherwise. Our technique is job-level dynamic in the sense that the speed of job may change during its execution. To make sure that heavy jobs do not interfere with other jobs, their priority should be given as if they executed at speed S_i. Thus their laxity should be calculated as $l_{i,j}(t, S_i)$. Algorithm 1 describes the scheduling procedure.

Algorithm 1. Job-level dynamic DVFS for EDZL scheduling

```
release_job(τ_{i,j}):
    if u_i > 0.5 then
        S_i^L = 2S_i − 1
    else
        S_i^L = S_i
    acc_exec = 0    // accumulative execution amount

set_ priority(τ_{i,j}):
    if l_{i,j}(t,S_i)==0 then
τ_{i,j}.priority = 0    // highest priority
    else
        τ_{i,j}.priority = d_{i,j}  // according to EDF

schedule_jobs():
    foreach τ_{i,j} in ready queue
    set_priority(τ_{i,j})
    done
    foreach mhighest priority job
        set_speed(τ_{i,j})
    execute_job(τ_{i,j})
    done

set_ speed(τ_{i,j}):
    if acc_exec>worst_case_exec/2 then
```

core_speed = 1
else
core_speed = S_i^L

execute_ job($\tau_{i,j}$):
*acc_exec += used_ time*core_speed*

4. EXPERIMENTS

We evaluate our DVFS technique incorporated with EDZL scheduling through simulation. Periodic task sets are generated as follows. For each m = 4,8,16, the total utilization is incremented from 0.25m to 0.90m with a step of 0.2.For each utilization,we generate 100 tasks sets. When generating taskτ_i, p_i and u_i are randomly chosen from a uniform distribution over (10,1000] and (0.1,1], respectively. Then e_i is calculated by $p_i \times u_i$. Every task set is testedusing Lee's test [21]. If a task set does not pass the test, we discard it and generate another task set. In the simulation, the actual execution time of $\tau_{i,j}$ are randomly chosen from [1, e_i].

The processor model of our experiment is shown in Table 1. When a job is scheduled to execute on a core, the core's supply voltage and frequency are changed together to the lowest level of which the speed is higher than or equal to the job's speed. On a job's completion, we sum up its energy consumption. The amount of energy consumed is modeled as follows.

$$E = P(e/s) \qquad (2)$$

where e is the amount of processor time the job execute, s is the execution speed on that time interval, and P is the power dissipation at the voltage/frequency level.

Table 1: Characteristics of StrongARM SA-1100

Speed	Volt.(V)	Freq.(MHz)	Power(%)
1.000	1.50	206	100
0.947	1.42	195	78.9
0.874	1.30	180	63.2
0.801	1.20	165	50.0
0.728	1.15	150	39.9
0.655	1.10	135	33.6
0.583	1.08	120	33.3
0.510	0.95	105	19.8
0.437	0.90	90	15.0
0.364	0.82	75	11.8
0.291	0.80	60	9.44

For each task set, the energy consumption during a hyper-period is calculated by summing up the energy consumption of all jobs of all tasks in the task set. Then the amount of energy consumption is normalized to the amount of energy consumption without any DVFS technique. We average the normalized energy consumption of task sets with the same total utilization.

Figure 1, 2, 3, and4show the average normalized energy consumption for m = 2, 4,8, 16, respectively. In the figures, *Han* denotes the average normalized energy consumption of the individual speed scheme presented in [22].The figures demonstrate that our technique further reduces energy consumption. Han' scheme cannot reclaim dynamic slack time that occurs when a job actually demand a less amount of execution than the worst-case. By aggressively lower the speed of heavy jobs, our technique reclaims such slack time. For instance, when m = 2 and the total utilization is 1.6, our scheme saves 6.36%of normalized energy in average. For a fixed number of cores, on the whole, more energy can be saved as the total utilization increases. For task sets with high total utilization, it is likely that there exist heavy execution tasks. Such tasks may have much slack time if their jobs actually demand far less execution than the worst-case. Those slack time can be reclaimed by dynamic DVFS techniques like our scheme.

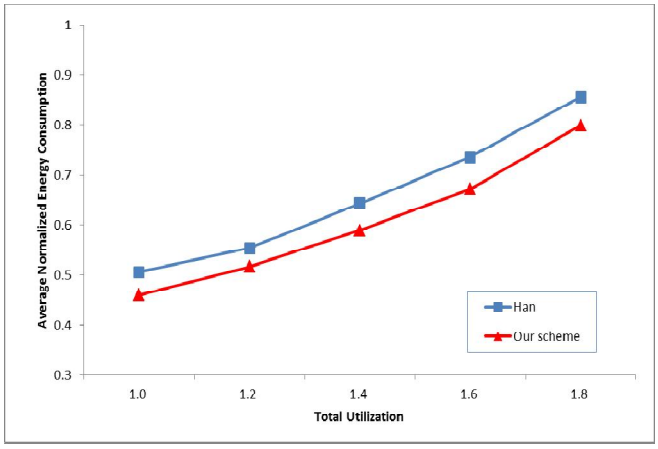

Figure 1: Energy consumption (m=2)

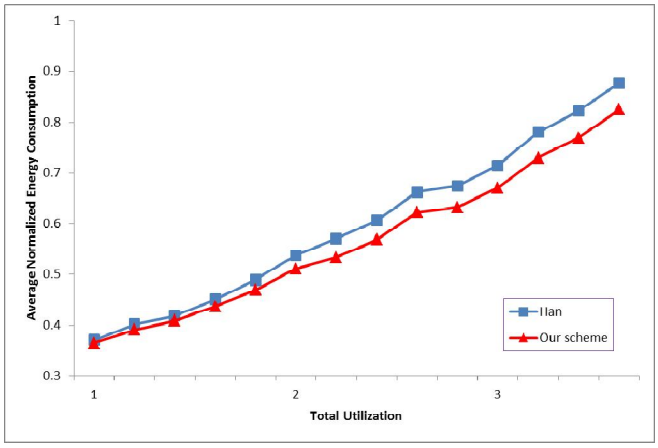

Figure 2: Energy consumption (m=4)

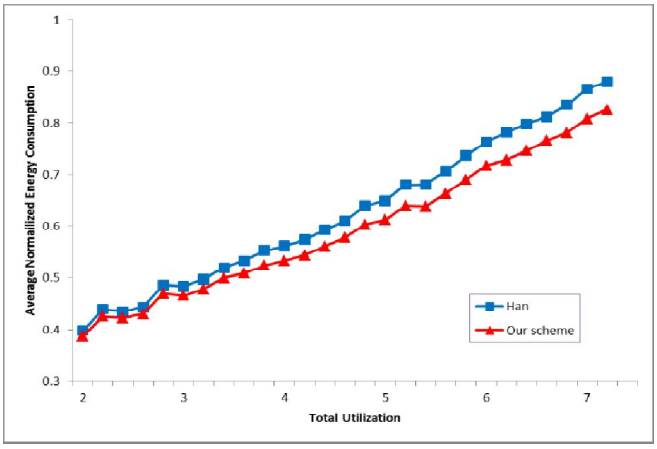

Figure 3: Energy consumption (m=8)

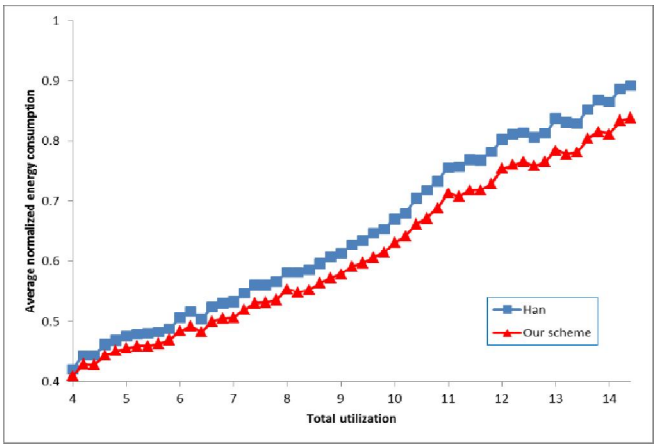

Figure 4: Energy consmption (m=16)

5. CONCLUSION

This paper proposes a simple and aggressive voltage/frequency scaling scheme for EDZL scheduling algorithm. Our scheme aggressively lowers the execution speed of heavy jobs to reclaim slack time. If a job uses much less processor time than its worst-case execution demand, we can achieve a considerable energy reduction. The proposed technique can lessen energy consumption compared with Han's scheme by at most 6.36%.

REFERENCES

1. V. Nelis, J. Goossens, R. Devillers, and N. Navet, **Power-aware real-time scheduling upon identical multiprocessor platforms**, in *Proc. IEEE International conference on sensor networks, ubiquitous and trustworthy computing (SUTC'08)*, pp. 209–216, 2008.
2. A. Chandrakasan, S. Sheng, and R. Brodersen. **Low-Power CMOS Digital Design**, *IEEE Journal of Solid-State Circuit*, vol. 27, no. 4, pp. 473–484, 1992. https://doi.org/10.1109/4.126534
3. Intel Product Specifications. Retrieved November 29, 2018, from: https://ark.intel.com/.
4. J. Khan, S. Bilavarn, and C. Belleudy. **Energy Analysis of a DVFS based power strategy on ARM platforms**, in *Proc. IEEE Faible Tension Faible Consommation (FTFC)*, Paris, France, pp. 1–4, 2012.
5. AMD Products. Retrieved November 29, 2018, from: http://www.amd.com/en-us/products.
6. H. Aydin, and Q. Yang, Q. **Energy-aware partitioning for multiprocessor real-time systems**,in *Proc. 17th International Symposium on Parallel and Distributed Processing (IPDPS'03)*, Nice, France, 2013.
7. J.J.Chen, and T.W. Kuo. **Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics**,in *Proc. the 2005 International Conference on Parallel Processing (ICPP'05)*, Oslo, Norway, pp. 13–20, 2005.
8. C.Y. Yang, J.J. Chen, and T.W Kuo. **An approximation algorithm for energy-efficient scheduling on a chip multiprocessor**. in *Proc. Conference on Design, Automation and Test in Europe (DATE'05)*, Munich, Germany, pp. 468–473, 2005
9. J.J. Chen, and C.F. Kuo. **Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms**,in *Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'07)*, Daegu, Korea, pp. 28–38, 2007. https://doi.org/10.1109/RTCSA.2007.37
10. C.L. Liu, and J.W. Layland. **Scheduling Algorithms for Multiprogramming in a HardReal-Time Environment** ,*Journal of the ACM*. vol. 20, no. 1, pp. 46–61, 1973.
11. N.C. Audsley, A. Burns, M.F. Richardson, and A.J. Wellings. **Hard real-Time scheduling: the deadline-monotonic approach**, in *Proc. IFAC/IFIP Workshop on Real Time Programming*, Atlanta, USA, pp. 127-132, 1991.
12. K. Funaoka, A. Takeda, S. Kato, and N. Yamasaki. **Dynamic Voltage and Frequency Scaling for Optimal Real-Time Scheduling on Multiprocessors**,in *Proc. 3rd IEEE International Symposium on Industrial Embedded Systems (SIES'08)*, Le Grande Motte, France, pp. 27–33, 2008. https://doi.org/10.1109/SIES.2008.4577677
13. V. Nelis, J. Goossens, R. Devillers, and N. Navet. **Power-Aware Real-Time Scheduling upon Identical Multiprocessor Platforms**,in *Proc. IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC'08)*, Taichung, Taiwan, pp. 209–216, 2008.
14. X. Piao, H. Kim, Y. Cho, S. Han, M. Park, and M. Park. **Power-Aware EDZL Scheduling upon Identical Multiprocessor Platforms**,in. *Proc. International Conference on Reliable and Autonomous Computational Science (RACS 2010)*, Atlanta, USA, pp. 61–80, 2010.
15. S. Funk, V. Berten, C. Ho, and J. Goossens. **A global optimal scheduling algorithm for multiprocessor low-power platforms**,in *Proc. 20th International Conference on Real-Time and Network Systems*, Pont à Mousson, France,pp. 71–80, 2012.

16. S. Han, M. Park, X. Piao, and M. Park. **A dual speed scheme for dynamic voltage scaling on real-time multiprocessor systems**,*The Journal of Supercomputing*,vol. 71, no. 2, pp. 574–590, 2015.
17. S. Cho, S.K. Lee, A. Han, and K.J. Lin. **Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems**,*IEICE Trans on Communications*, vol. E85-B, no. 12, pp. 2859–2867, 2002.
18. 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 Trans on Information and Systems*,vol. E88-D, no. 3, pp. 658–661, 2005. https://doi.org/10.1093/ietisy/e88-d.3.658
19. M. Cirinei, and T.P. Baker. **EDZL Scheduling Analysis**,in *Proc. 19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, Pisa, Italy, pp. 9–18, 2007.
20. T.P. Baker, M. Cirinei, and M. Bertogna. **EDZL scheduling analysis**,*Real-Time Systems*,vol. 40, no. 3, pp. 264–289, 2008.
21. J. Lee, and I. Shin. **EDZL Schedulability Analysis in Real-Time Multicore Scheduling**,*IEEE Transactions on Software Engineering*, vol. 39, no. 7, pp. 910–916, 2013.
22. S. Han, M. Park, and W. Paik.**Dynamic Voltage/Frequency Scaling for EDZL Scheduling in Multicore Real-Time Systems**, *Journal of Engineering and Applied Sciences*, vol. 14, no. 21, pp. 8039-8046, 2019.
23. M S. Kumar, F. Noorbasha, S. Inthiyaz, M. Jameela, A. Sandhya, Md. Imran, and S. K. Tulasi. **Low Power Carry Look-Ahead Adder using Transmission Gate Multiplexer**, *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 1, 2020. https://doi.org/10.30534/ijeter/2020/03812020
24. B. Lakshmi, and B. Navyasri. **Energy Efficient Routing Mechanism for Harsh Environment in Wireless Sensor Networks**, *International Journal of Emerging Trends in Engineering Research*, vol. 7, no. 9, 2019. https://doi.org/10.30534/ijeter/2019/04792019
25. P.S. Akram, G.V. Ganesh, A. S. Kumar, K.S. Chand, and M.R. Varma. **Non-Volatile 7T1R SRAM cell design for low voltage applications**, *International Journal of Emerging Trends in Engineering Research*, vol. 7, no. 11, 2019. https://doi.org/10.30534/ijeter/2019/487112019
26. S. Han. **Energy-aware EDZL Scheduling of Periodic Tasks on Multicore Systems**, *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 4, 2020. https://doi.org/10.30534/ijeter/2020/18842020