International Journal of Advanced Smart Convergence Vol.7 No.3 92-100 (2018)
http://dx.doi.org/10.7236/IJASC.2018.7.3.92

IJASC 18-3-11

Fault-tolerant Scheduling of Real-time Tasks with Energy Efficiency on Lightly Loaded Multicore Processors

Wan Yeon Lee[1*], Yun-Seok Choi[2]

[1] *Dept. of Computer Science, Dongduk Women's University, Seoul 136-714, South Korea, wanlee@dongduk.ac.kr*
[2] *Dept. of Computer Science, Dongduk Women's University, Seoul 136-714, South Korea, cooling@ dongduk.ac.kr*

Abstract

In this paper, we propose a fault-tolerant scheduling scheme with energy efficiency for real-time periodic tasks on DVFS-enabled multicore processors. The scheme provides the tolerance of a permanent fault with the primary-backup task model. Also the scheme reduces the energy consumption of real-time tasks with the fully overlapped execution between each primary task and its backup task, whereas most of previous methods tried to minimize the overlapped execution between the two tasks. In order to the leakage energy loss of idle cores, the scheme activates a part of available cores with rarely used cores powered off. Evaluation results show that the proposed scheme saves up to 82% energy consumption of the previous method.

Keywords: *Real-time Task, Scheduling, Fault Tolerance, Energy-efficient Design, Multicore Processor*

1. Introduction

Energy-efficient design of multicore processors is a critical issue, especially for battery-operated mobile devices. High energy dissipation of processing components results in temperature increase of computing systems. The temperature increase directly impacts the performance and reliability of integrated circuits (ICs) [1]. Thus low energy dissipation of processing components is important for the scheduling problem of real-time tasks. Whereas traditional approaches employed the power down of unused components for energy saving [2], state-of-the-art approaches employ the Dynamic Voltage Frequency Scaling(DVFS) mechanism because the energy saving effect in the DVFS mechanism is much larger than that in the power down of unused processing cores. The DVFS mechanism dynamically changes the voltage and clock frequency supplied to processing cores [3,4].

As well as energy efficiency, fault tolerance is another important factor for mission-critical applications such as smartphone navigation and mobile surveillance system. Faults are generally classified into two

Manuscript Received: August. 5, 2018 / Revised: August. 13, 2018 / Accepted: August. 20, 2018
Corresponding Author: wanlee@dongduk.ac.kr
Tel: +82-2-940-4685, Fax: +82-2-940-4170
Dept. of Computer Science, Dongduk Women's University

groups: permanent fault and transient fault. A permanent fault is tolerated with modular redundancy technique that executes an application in duplicate on several processing units [5]. A transient fault is tolerated with temporal redundancy technique that re-executes a faulty application [6]. Fault-tolerant design for permanent faults is directly applicable to that for transient faults, but fault-tolerant design for transient faults cannot be directly applicable to that for permanent faults. Hence the tolerance for permanent faults is considered in this study.

Recent fault-tolerant scheduling schemes have considered energy efficiency of real-time tasks with the DVFS mechanism. In order to tolerate a permanent fault, a backup copy of each task is ready to run on other cores in case of its primary task failure, which is referred to as primary-backup(PB) model [5,7]. However, most of them have been developed with a premise that backup copy task is activated only when its primary task stops due to a failure. The proposed scheme concurrently executes a primary task and its backup task with the lowest clock frequency meeting their deadlines in order to reduce energy consumption of real-time tasks. Total energy consumption of two cores executing the primary task and the backup task concurrently with low core speed is possibly smaller than that of two cores executing the two tasks sequentially with high core speed, because energy consumption is approximately proportional to the cube of core speed in the DVFS mechanism. Also the proposed scheme turns off the power of rarely used processing cores in order to reduce the leakage energy consumption.

Whereas the energy-efficient scheduling problems with two constraints among real-time completion, fault tolerance and multicore processors have been widely studied, the energy-efficient scheduling problem considering all the three constraints is rarely studied. A few studies dealt with the energy-efficient scheduling problem of real-time tasks with fault tolerance on multicore processors [5,7,8,9]. However the previous studies [5,7,8] do not consider the potential energy saving capability of concurrently executing a primary task and its backup task, and the previous study [9] does not consider the potential energy saving capability of turning off the power of rarely used cores. In this paper, we handle a scheduling problem of real-time periodic tasks on multicore processors that minimizes the total energy consumptions of all processing cores while tolerating a single permanent fault and completing the execution of each task within its deadline. Unfortunately, the problem of minimizing total energy consumption is NP-hard. Hence the proposed scheme searches for a near minimum-energy schedule within a polynomial time on multiple processing cores, where the deadline and fault tolerance constraints are satisfied. Evaluation results show that the proposed scheme saves up to 82% energy consumption of the previous method.

The rest of this paper is organized as follows; Section 2 explains the considered system model. Section 3 describes the proposed scheme in detail. Section 4 shows evaluation results. Section 5 provides concluding remarks.

2. System Model

Given $\boldsymbol{N}$ processing cores are homogeneous and support the DVFS(Dynamic Voltage Frequency Scaling) mechanism that dynamically changes the voltage and clock frequency supplied to the processors [3,4]. The n^{th} processing core is denoted as $\boldsymbol{C}_n$. Computation speed of the processors is typically proportional to the clock frequency, and their energy consumption per unit time is approximately proportional to the cube of the clock frequency. Then energy consumption per unit time is proportional to the cube of core speed. The maximum core speed is denoted as $\boldsymbol{S}_{\max}$ and normalized to $\boldsymbol{S}_{\max} = 1.0$. Scaled-down speed is denoted as $\boldsymbol{S}$ such that $0 < \boldsymbol{S} < \boldsymbol{S}_{\max} = 1.0$. The energy consumption rate at a core speed $\boldsymbol{S}$ is denoted as $\boldsymbol{E}(\boldsymbol{S}) = \alpha \cdot \boldsymbol{S}^3 + \boldsymbol{e}_0$, where α is a hardware-dependent constant and $\boldsymbol{e}_0$ is leakage energy consumption. The leakage energy consumption e_0 is the energy consumption in the idle status when a core has no computation to execute, which is strictly positive [3]. That is $E(0) = e_0 > 0$. Therefore turning off the power of unused cores need reduces the energy consumption. An identical core speed is supplied to all activated cores. We do not consider the system model that supplies different core speeds simultaneously to activated cores.

Given M periodic tasks are preemptive and have no interdependency. In the PB(primary-backup) model [5,7], each primary periodic task and its backup copy task should complete their computation within their arrival period, which becomes their deadline. The m^{th} primary periodic task is denoted as T_m and its backup task is denoted as B_m. D_m and W_m respectively denote the deadline of T_m and the worst processing time of T_m at the maximum core speed S_{max}. B_m has the same parameters with T_m. When T_m and B_m are executed with a speed S such that $S < S_{max}$, their execution time is W_m/S such that $W_m/S > W_m/S_{max}$. The proposed scheme is designed for a single permanent hardware fault, which includes a transient fault and results in failure of at most one core. In order to tolerate the failure of Tm, both T_m and B_m should be completed within D_m.

The ratio of the execution time of T_m at the maximum speed S_{max} to the deadline D_m is referred to as *task utilization* and denoted as $U_m = W_m/(D_m \cdot S_{max})$. Because $S_{max} = 1$, $W_m = D_m \cdot U_m$. The backup task B_m has the same U_m with its primary task T_m. Total utilization of all tasks assigned to a core C_n is referred to as *core load* and denoted as $L_n = \sum U$. The lowest constant speed that can execute all the tasks assigned to core C_n is referred to as *optimal speed* and denoted as S_n^{opt}. In other words, the optimal speed is the minimum core speed satisfying the deadlines of all the assigned tasks.

Even though the static version of the proposed scheme tolerates a single permanent fault, its dynamic version can tolerate multiple permanent faults through the dynamic reconfiguration mechanism that reassigns all tasks to non-faulty cores after excluding the faulty core at run-time.

The considered problem is to minimize total energy consumption of N processors while executing M periodic real-time tasks within their respective deadlines. The M independent tasks can be executed on a subset of N processors with unused processing cores powered off. A schedule is called *feasible* if the M tasks are completely executed within their respective deadlines.

3. Proposed Scheduling

It is verified that the minimum energy consumption when *maximizing* the overlapped execution period between the primary task and its backup task is smaller than that when *minimizing* the overlapped execution period between the two tasks under the condition $U > (1 - 1/\sqrt{2}) \approx 0.29$ [9].

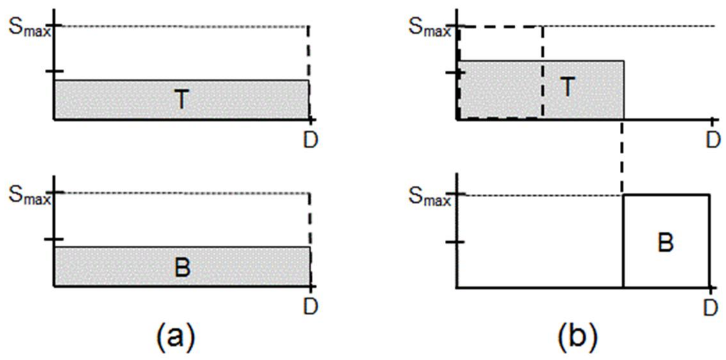

Figure 1. Scheduling comparison of two approaches

Figure 1(a) shows the case where the overlapped execution period between the primary task and its backup task is maximized. In this figure, the primary task and its backup task are concurrently executed with the optimal speed on separate two cores. Figure 1(b) shows the case where the overlapped execution period between the primary task and its backup task is minimized. In this figure, the primary task and its backup tasks are sequentially executed on separate two cores. To minimize the energy consumption in the fault-free status, the backup task is not actually executed if the primary task is successfully completed. The execution time of the backup task is reserved with the maximum core speed in order to minimize its dominantly wasted execution time due to very low probability of a permanent fault. Then the primary task is executed with the lowest speed completing the primary task within the time period remaining after excluding the reserved time of its backup task from their deadline. The energy consumption of the backup task is discarded due to the very low probability of a permanent fault in this comparison.

In Figure 1(a), the optimal speed is equal to the task utilization according to the definition of task utilization in Section 2, i.e., $\boldsymbol{S}^{opt} = \boldsymbol{U}$. Then the energy consumption of two cores when executing the primary task and its backup task concurrently with the optimal speed for the time $\boldsymbol{D}$ is

$$2 \cdot \{\alpha \cdot (S^{opt})^3 \cdot e_0\} \cdot D = 2 \cdot \alpha \cdot (U)^3 \cdot D + 2 \cdot e_0 \cdot D \tag{1}$$

In Figure 1(b), the execution time of the backup task with the maximum speed is $\boldsymbol{W} = \boldsymbol{D} \cdot \boldsymbol{U}$ according to the definition of task utilization $\boldsymbol{U}$ in Section 2. The primary task is executed within the decreased period ($\boldsymbol{D} - \boldsymbol{W}$) = ($\boldsymbol{D} - \boldsymbol{D} \cdot \boldsymbol{U}$) remaining after reserving the execution of the backup task. The lowest speed executing the primary task for the time ($\boldsymbol{D} - \boldsymbol{D} \cdot \boldsymbol{U}$) is $\boldsymbol{S} = \boldsymbol{W} / (\boldsymbol{D} - \boldsymbol{D} \cdot \boldsymbol{U}) = \boldsymbol{D} \cdot \boldsymbol{U} / (\boldsymbol{D} - \boldsymbol{D} \cdot \boldsymbol{U}) = \boldsymbol{U} / (1 - \boldsymbol{U})$ because $\boldsymbol{W} / \boldsymbol{S} = (\boldsymbol{D} - \boldsymbol{D} \cdot \boldsymbol{U})$ and $W = \boldsymbol{D} \cdot \boldsymbol{U} \cdot \boldsymbol{S}_{\max} = \boldsymbol{D} \cdot \boldsymbol{U}$ according to the definitions of the scale-down speed $\boldsymbol{S}$ and the worst processing time $\boldsymbol{W}$ in Section 2. Then the energy consumption of the core executing the primary task for the time $\boldsymbol{D}$ is $\{\alpha \cdot (\boldsymbol{S})^3 + \boldsymbol{e}_0\} \cdot (\boldsymbol{D} - \boldsymbol{D} \cdot \boldsymbol{U}) + \boldsymbol{e}_0 \cdot (\boldsymbol{D} \cdot \boldsymbol{U}) = \alpha \cdot (\boldsymbol{S})^3 \cdot (\boldsymbol{D} - \boldsymbol{D} \cdot \boldsymbol{U}) + \boldsymbol{e}_0 \cdot \boldsymbol{D}$. The energy consumption of the core reserved for the back task for the time $\boldsymbol{D}$ is $\boldsymbol{e}_0 \cdot \boldsymbol{D}$. Hence total energy consumption of the two cores for the time $\boldsymbol{D}$ is

$$\begin{gathered}\alpha \cdot (S^{opt})^3 \cdot (D - D \cdot U) + 2 \cdot e_0 \cdot D = \\ \alpha \cdot (U/(1-U))^3 \cdot (D - D \cdot U) + 2 \cdot e_0 \cdot D = \\ \alpha \cdot U^3/(1-U)^2 \cdot D + 2 \cdot e_0 \cdot D\end{gathered} \tag{2}$$

From the equation (1) and (2), the energy consumption in Figure 1(a) is smaller than that in Figure 1(b) when $\boldsymbol{U} > (1 - 1/2^{1/2}) \approx 0.29$, because $2 \cdot \alpha \cdot (\boldsymbol{U})^3 \cdot \boldsymbol{D} < \alpha \cdot \boldsymbol{U}^3 / (1 - \boldsymbol{U})^2 \cdot \boldsymbol{D}$ when $\boldsymbol{U} > (1 - 1/2^{1/2})$.

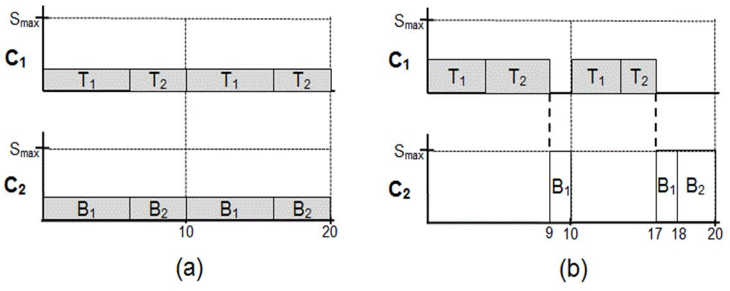

Figure 2. Scheduling of multiple tasks running on a core

When multiple tasks are executed on a single core, it is known that the minimum-energy schedule determines their execution order according to the earliest-deadline first(EDF) rule and its optimal speed is equal to total task utilization of all assigned tasks, i.e., $\boldsymbol{S} = \Sigma \boldsymbol{U}$ [4]. Total task utilization of all tasks assigned to a core $\boldsymbol{C}_n$ is called *core load* and denoted as $\boldsymbol{L}_n = \Sigma \boldsymbol{U}$. Then the energy consumption of is $\boldsymbol{E}(\boldsymbol{S}) = \boldsymbol{E}(\boldsymbol{L})$ because $\boldsymbol{S} = \boldsymbol{L} = \Sigma \boldsymbol{U}$. Figure 2 shows the case when multiple tasks are running on a core. Primary task and its backup tasks are assigned to different cores. For the sake of simplicity, primary tasks or backup tasks are allocated exclusively to at most $\boldsymbol{N}/2$ cores. Because of $\boldsymbol{E}(\boldsymbol{S}) = \boldsymbol{E}(\boldsymbol{L})$, the minimum energy consumption when *maximizing* the overlapped execution period between primary and backup tasks, shown in Figure 2(a), is smaller than that when *minimizing* the overlapped execution period, shown in Figure 2(b), if $\boldsymbol{L} > (1 - 1/2^{1/2})$.

To find the minimum-energy feasible schedule, we first determine the number of activated cores among available $\boldsymbol{N}$ processors. In other works, the number of powered off cores is determined. Our goal is to minimize total mean energy consumption of all activated cores executing given real-time tasks. If the number of activated cores is fixed and given as η, then distributing the total task utilization of all primary and backup tasks evenly to η cores minimizes total power consumption of all activated cores because $\boldsymbol{E}(\boldsymbol{L})$ is a convexly increasing function of $\boldsymbol{L}$ [4].

The number η of activated cores is determined as follows; Define a linear function $\beta \cdot \boldsymbol{L}$ with an input of $\boldsymbol{L}$ where β is a positive constant. If β is selected to be $\beta \cdot \boldsymbol{L} = \boldsymbol{E}(\boldsymbol{L})$ only at a unique point $\boldsymbol{L} = \delta$ as shown in Figure 3, then $\eta = 2 \cdot (\sum_{m=1}^{M} U_m) / \delta$ because $\delta = 2 \cdot (\sum_{m=1}^{M} U_m) / \eta$. When the total utilization of all primary and backup tasks, $2 \cdot (\sum_{m=1}^{M} U_m)$, is evenly distributed to η cores, the minimum energy consumption of η cores is $\eta \cdot \boldsymbol{E}_n(\delta)$ because $\delta = 2 \cdot (\sum_{m=1}^{M} U_m) / \eta$. If $(\eta + \lambda)$ cores are activated for positive λ, then the minimum energy consumption of $(\eta + \lambda)$ cores is $(\eta + \lambda) \cdot \boldsymbol{E}(\delta - \omega)$ where $(\delta - \omega) = (\sum_{m=1}^{M} U_m) / (\eta + \lambda)$. As shown in Figure 3, $(\eta + \lambda) \cdot \boldsymbol{E}(\delta - \omega) > \beta \cdot (\eta + \lambda) \cdot (\delta - \omega) = \beta \cdot \eta \cdot \delta = \eta \cdot \boldsymbol{E}(\delta)$. That is, the minimum energy consumption of $(\eta + \lambda)$ activated cores is larger than that of η activated cores. By a similar reason, the minimum energy consumption of $(\eta - \lambda)$ activated cores is larger than that of η activated cores. Hence, distributing total utilization of all primary and backup tasks to η cores as evenly as possible minimizes total energy consumption of all activated cores. If η is not an integer, then one of two neighboring integers is selected to have less total energy consumption of activated cores.

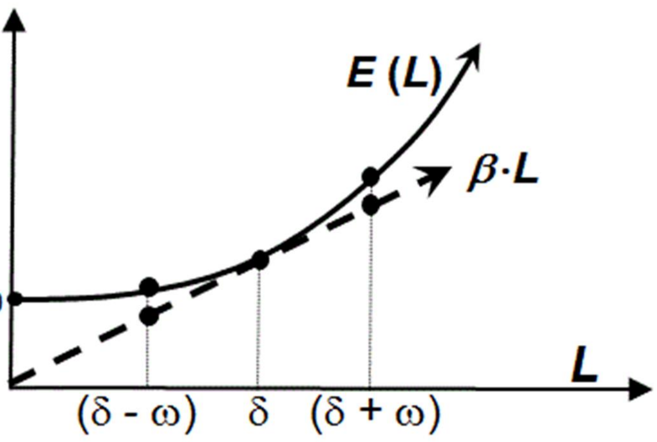

Figure 3. Determination of the number of activated cores

The remaining issue is to determine the core to which each task is assigned. From the load value of each core, we can derive the optimal speed and the minimal long-term energy consumption of each core. The problem of minimizing total energy consumption of all cores can be formulated as follows:

$$\text{Minimize } \sum_{n=1}^{N} \boldsymbol{E_n}(\boldsymbol{L_n}), \text{ subject to } L_n \leq 1 \quad \text{for each } n$$

where unused cores with no assigned tasks are powered off (i.e., $\boldsymbol{E}(0) = 0$ if $\boldsymbol{L}_n = 0$, instead of $\boldsymbol{E}(0) = e_0$).

Although the above problem has a lower complexity than the original task scheduling problem, it is still NP-hard for a general task set because the problem of distributing total task utilization of given multiple tasks evenly to given multiple cores is NP-hard [4]. Because this computational overhead is too heavy to run even offline for a large number of available cores and tasks, we propose a scheduling scheme that finds a near minimum-energy feasible schedule within a polynomial time. The following pseudo-code describes the proposed scheduling scheme.

```
Step 1. Calculate task utilization values $U_m$ for each task $T_m$ and determine the number $\eta$ of
        activated cores such that $\eta = 2 \cdot (\sum_{m=1}^{M} U_m) / \delta$.
Step 2. If the evenly distributed core load $L$ (= $\delta$) of $\eta$ activated cores is no larger than ($\delta < 1 - 1/2^{1/2}$),
        then use the previous method [5]. Otherwise, go to Step 3.
Step 3. Assign given primary $M$ tasks to $\eta/2$ cores and backup $M$ tasks to the remaining $\eta/2$ cores.
    3.1: Sort all primary tasks in the decreasing order of their task utilization.
    3.2: Assign each primary task one by one to the core with the lowest core load among $\eta/2$ cores.
    3.3: Sort all backup tasks in the decreasing order of their task utilization.
    3.4: Assign each backup task one by one to the core with the lowest core load among $\eta/2$ cores.
Step 4. Determine the schedule of each activated core.
    4.1: Sort the execution order of the tasks assigned to each core based on the
         earliest-deadline-first(EDF) rule.
    4.2: Apply the optimal speed $S^{opt} = L = \delta$ (= $2 \cdot (\sum_{m=1}^{M} U_m) / \eta$) to all activated cores.
```

The computational complexity of the proposed algorithm is $O(M \cdot \log M \cdot N)$. The complexity of Step 1 is $O(M \cdot N)$. The complexity to find the value of β and δ is $O(M)$. The complexity to calculate the number η is $O(M)$. The complexity of Step 2 is $O(1)$. The complexity of Step 3.1 and Step 3.3 is $O(M \cdot \log \mathrm{M})$. The time complexity of Step 3.2 and Step 3.4 is $O(M \cdot N)$. The complexity of Step 4.1 is $O(M \cdot \log \mathrm{M} \cdot N)$ and that of Step 4.2 is $O(1)$.

4. Evaluation

The proposed scheme is compared with the previous method [9] that assigns given tasks to all available cores. The previous method [9] did not consider turning off the power of rarely used cores, whereas the proposed scheme turns off the power of rarely used cores after migrating their assigned tasks to other activated cores. As a performance metric, we define the ratio of total energy consumption in the proposed scheme to that in the previous method as *Normalized Energy Consumption* (NEC). For performance evaluation, we employ simulation experiments with MATLAB tool on Windows 7 operating system.

In the evaluation, we synthetically generates periodic tasks and set $\alpha = 1.55\times10^{-6}$ and e_0 = 60mW obtained by applying curve fitting of $(\alpha{\cdot}S^3 + e_0)$ to the clock frequencies of an Intel XScale processor and their energy consumption value [3]. Then $\delta = 0.268$. 16 processing cores are given (N=16). A primary task set consists of 16, 24 or 32 periodic tasks. The deadline of each primary task is randomly selected between 10 milliseconds and 1 second. The utilization of each primary task is synthetically generated between 0.01 and 1.0 from a normal distribution. We run 10,000 task sets and display their average values.

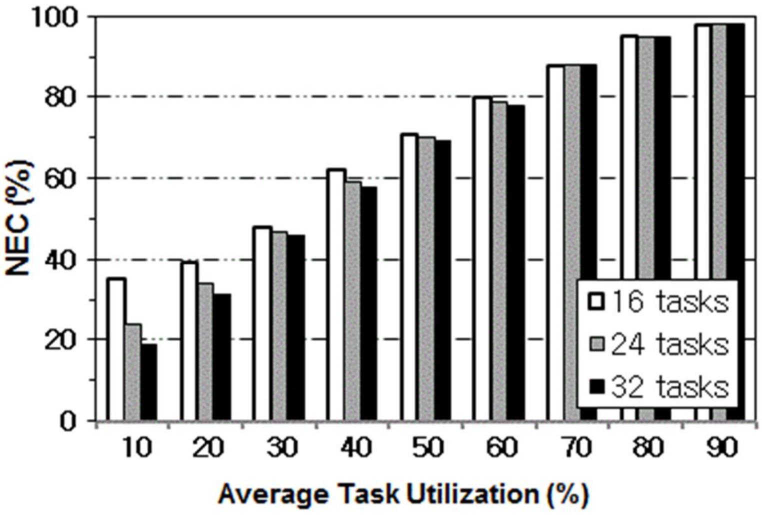

Figure 5. NEC values against average Task Load

In the first set of comparisons, we examine the performance of the relative task utilization to their deadline. To measure the relative task utilization to the deadline, we define the average value of all tasks' utilization as *Average Task Utilization*, i.e., $100 \times \Sigma U / M$. Figure 5 shows NEC values against Average Task Utilization. As the value of Average Task Utilization decreases, the energy saving effect of the proposed scheme increases. Also, as the number of given tasks increases, the energy saving effect increases. When the number of given tasks is 32 and the Average Task Utilization is 10%, the proposed scheme saves about 82% energy consumption of the previous method.

In the second set of comparisons, we examine the number of activated cores among all available cores in the proposed scheme. Figure 6 shows the number of activated cores for all primary and backup tasks. As the value of Average Task Utilization decreases or the number of tasks increases, the proposed scheme activates fewer cores while the previous method activates all available cores. When Average Task Utilization is 10% and the number of tasks is 32, the proposed scheme reduces the number of activated cores by about 87%, compared with the previous method. From Figure 5 and Figure 6, it is verified that the proposed scheme saves more energy as the proposed scheme activates fewer cores among given available cores, compared to the previous method.

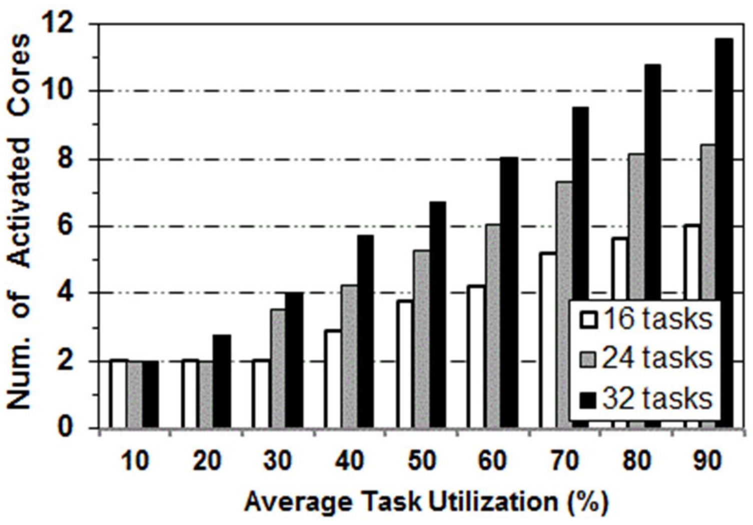

Figure 6. Number of Activated Processors

5. Conclusions

In this paper, we handle the problem of minimizing the energy consumption of real-time periodic tasks while meeting their deadlines and tolerating a permanent fault on lightly loaded DVFS-enabled multicore processors. The proposed scheme reduces the energy consumption of real-time tasks by utilizing the primary-backup model that fully overlaps the execution of each primary task and its backup task, whereas most of previous primary-backup methods tried to minimize the overlapped execution between a primary and its backup tasks. Also the scheme activates a part of available processing cores with rarely used cores powered off, in order to save the leakage energy consumption of idle cores. The proposed scheme is designed to find a near minimum-energy feasible schedule within a polynomial time, because the problem of minimizing the energy consumption of real-time tasks while meeting their deadline is NP-hard. Evaluation shows that the proposed scheme saves up to 82% energy consumption of the previous method.

Acknowledgement

This research was supported by the Dongduk Women's University Grant, 2017.

References

[1] X. Zhou, J. Yang, M. Chrobak and Y. Zhang, "Performance-Aware Thermal Management via Task Scheduling," ACM Transactions on Architecture and Code Optimization (Apr. 2010), vol. 7, no. 1, pp. 1-31.

[2] L. Benini, A. Bogliolo, and G. Micheli, "A survey of design techniques for system-level dynamic power management," IEEE Trans. VLSI Syst., Vol. 8, No. 3, pp. 299-316, 2000.
DOI: http://doi.ieeecomputersociety.org/10.1109/92.845896

[3] R. Xu, C. Xi, R. Melhem and D. Mosse, "Practical PACE for Embedded Systems," International Conference on Embedded Software (2004), pp. 54-63.

[4] Wan Yeon Lee**,** Yun-Seok Choi, "Energy-efficient Scheduling of Periodic Real-time Tasks on Heterogeneous Grid Computing Systems," International Journal of Internet, Broadcasting and Communication, vol. 9, no. 2, pp. 78-86, May 2017.

DOI: http://dx.doi.org/10.7236/IJIBC.2017.9.2.78

[5] Y. Guo, D. Zhu and H. Aydin, “Efficient power management schemes for dual-processor fault-tolerant systems”, in Proceedings of Int'l Workshp Highly-Reliable Power-Efficient Embedded Designs, 2013, pp. 23-27

[6] T. Wei, P. Mishra, K. Wu and H. Liang, “Fixed-priority allocation and scheduling for energy-efficient fault tolerance in hard real-time multiprocessor systems”, IEEE Trans. Parallel Distrib. Syst., 2008, 19 (1), pp. 1511-1525
DOI: http://doi.ieeecomputersociety.org/10.1109/TPDS.2008.127

[7] M. K. Tavana, M. Salehi and A. Ejlali, “Feedback-based energy management in a standby-sparing scheme for hard real-time systems”, in Proceedings of IEEE Real-Time Systems Symp., 2011, pp. 349-356

[8] Y. Liu, H. Liang and K. Wu, “Scheduling for energy efficiency and fault tolerance in hard real-time systems,” Design, Automation and Test in Europe Conference and Exhibition, pp. 1444-1449, 2010.

[9] K. Lee, “Energy-efficient fault-tolerant scheduling based on duplicated executions for real-time tasks on multicore processors,” Journal of the Korea Society of Computer and Information, vol. 19, no. 5, pp. 1-10, 2014.
DOI: http://dx.doi.org/10.9708/jksci.2014.19.5.001