Vol.3, No.6 pp.173~178 pISSN: 2287-5891 http://dx.doi.org/10.3745/KTCCS.2014.3.6.173

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

Kwanwoo Lee†

ABSTRACT

By exploiting parallel processing, the proposed scheduling scheme enhances energy saving capability of multicore processors for real-time tasks while satisfying deadline and fault tolerance constraints. The scheme searches for a near minimum-energy schedule within a polynomial time, because finding the minimum-energy schedule on multicore processors is a NP-hard problem. The scheme consumes manifestly less energy than the state-of-the-arts method even with low parallel processing speedup as well as with high parallel processing speedup, and saves the energy consumption up to 86%.

Keywords : Fault Tolerance, Real-time Task, Scheduling, Multicore Processor

멀티코어 프로세서 상에서 에너지 효율을 고려한 실시간 병렬 작업들의 결함 포용 스케줄링

이 관 우†

요 약

제시된 스케줄링 기법은 병렬처리 기법을 활용하여 실시간 작업들의 데드라인 제약과 결함 포용 제약을 만족하면서 멀티코어 프로세서의 에너지 소모 효율성을 향상시켰다. 최소 에너지 소모량 스케줄을 찾는 것은 NP-hard 문제이므로, 제시된 기법은 다항식의 시간 내에 최소 에너지 소모량에 근접하는 스케줄을 찾는다. 제시된 기법은 연관된 최신 기법과 비교하여 높은 병렬처리 속도는 물론 낮은 병렬처리 속도에서도 에너지 소모량이 현격하게 낮았으며, 에너지 소모량을 최대 86% 줄였다.

키워드 : 결함 포용, 실시간 작업, 스케줄링, 멀티코어 프로세서

1. Introduction

Mission-critical applications such as smartphone navigation and mobile surveillance system demand satisfactions of fault tolerance, real-time constraint, and energy efficiency. Many fault-tolerant scheduling schemes [1, 2, 3] have considered energy efficiency of real-time tasks with DVFS (Dynamic Voltage Frequency Scaling) mechanism for battery-operated mobile devices. Permanent faults are tolerated with modular redundancy technique that executes an application in duplicate on several processing units [1, 2]. Ttransient faults are tolerated with temporal redundancy technique that re-executes a faulty application [3].

However, most of them have been done with a premise that each task is running on a single processing component. The proposed scheme exploits parallel processing to reduce energy consumption of real-time tasks while meeting their deadlines and tolerating a permanent fault based on PB (primary-backup) model [1, 2]. In PB model, a backup copy of each task is ready to run on other cores in case of its primary task failure. If the processing time of a primary task is larger than half its deadline, a portion of the backup copy must be executed even in the fault-free status in order to complete the backup copy within the deadline. Parallel processing on multiple cores reduces the processing times of the primary task and its backup copy,

※ This research was financially supported by Hansung University
† 정 회 원 : 한성대학교 정보시스템공학과 부교수
Manuscript Received : February 24, 2014
First Revision : June 2, 2014
Accepted : June 2, 2014
* Corresponding Author : Kwanwoo Lee(kwlee@hansung.ac.kr)

which can avoid executing a portion of the backup copy even in the fault-free status. It also decreases the lowest core speed at which a given task is completed exactly at its deadline. Total energy consumption of multiple cores executing the task in parallel with low core speed is generally smaller than that of a single core executing the task alone with high core speed [4], because energy consumption is approximately proportional to the cube of core speed in DVFS mechanism.

Unfortunately, the problem of minimizing total energy consumption of real-time tasks on multicore processors is NP-hard. Hence the proposed scheme searches for a near minimum-energy schedule within a polynomial time, where the deadline and fault tolerance constraints are satisfied and each task can be executed on multiple cores in parallel. To the best of our knowledge, the proposed scheme is the first study to deal with both energy efficiency of real-time tasks and tolerance of permanent faults with the support of parallel processing, whereas state-of-the-arts methods [1, 2, 3] disregarded the energy saving potential of parallel processing. A few recent studies [4, 5] verified that parallel processing can enhance energy efficiency. However, fault tolerance was not considered in [4] and deadline constraint of real-time tasks was not considered in [5]. Another recent study [6] addressed intra-task scheduling of a parallel task with tolerance of transient faults, but did not consider inter-task scheduling of multiple parallel tasks with tolerance of permanent faults.

Section 2 describes the considered system model, and Section 3 describes the proposed scheme in detail. Section 4 shows evaluation results, and Section 5 provides concluding remarks.

2. System Model

Given N processing cores are homogeneous and support DVFS mechanism. In DVFS mechanism [3], core speed is dynamically changed and energy consumption is proportional to the cube of core speed. Cores can operate independently with different speeds. The maximum core speed is denoted as S_{max} and normalized to $S_{max}=1.0$. Scaled-down speed is denoted as S where $0 \le S < S_{max}=1.0$. The proposed scheme is designed for a permanent hardware fault, which includes a transient fault and results in failure of at most one core.

Given M periodic tasks are preemptive and have no interdependency. Tasks considered in this study are computation-intensive multimedia applications that are easily split into subtasks and executed concurrently on multiple cores without precedence constraint [4]. In PB model [1, 2], each primary periodic task and its backup copy 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 copy 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}. n_m denotes the number of cores allocated for the execution of T_m. B_m has the same parameters with T_m. T_m and B_m are allocated exclusively to at most $\frac{N}{2}$ cores. Parallel processing speedups of T_m and B_m on n cores are given as a vector $P_m[n]$ for $1 \le n \le \frac{N}{2}$. Then $P_m[n_a] \le P_m[n_b]$ for $n_a < n_b$, $P_m[1]=1$, and $P_m[n] < n$ for $n \ge 2$ due to overhead of parallel processing [4]. The backup copy is always running at the maximum core speed to minimize the time reserved but nearly unused and to preserve the system's original reliability [1, 2]. The proposed scheme focuses on minimizing energy consumption in the fault-free status, because the fault-free status is dominant due to very low probability of faults.

3. Proposed Scheme

The ratio of total processing time of n cores executing T_m at S_{max} to the deadline is referred to as *task utilization* and denoted as $U_m(n)$. Processing time of a single core for T_m running on n cores at S_{max} is denoted as $E_m(n)$. Then $U_m(n)$ and $E_m(n)$ can be formulated as below:

$$U_m(n) = n \cdot \frac{W_m}{P_m[n] \cdot D_m} \text{ and } E_m(n) = \frac{W_m}{P_m[n]}$$

Notice that the definition of task utilization is different from that in the previous studies [1, 2, 3] where task utilization is fixed as $U_m(1) = \frac{W_m}{D_m}$. $U_m(n)$ and $E_m(n)$ values vary according to n and $P_m[n]$ values.

The proposed scheme adopts the EEFT method [1] that minimizes the portion of the backup copy being executed in the fault-free status. In the EEFT method, primary tasks are scheduled as *soon* as possible and backup copies are scheduled as *late* as possible. Then the minimum energy consumption amount of a primary task

and its backup copy activated in the fault-free status can be formulated as below:

$$\psi_m(n)=\begin{cases} n\cdot(E_m(n)+2(E_m(n)-\frac{D_m}{2} & \text{if } E_m(n)>\frac{D_m}{2} \\ \left(\frac{E_m(n)}{D_m-E_m(n)}\right)^2\cdot n\cdot E_m(n) & \text{if } E_m(n)\le\frac{D_m}{2} \end{cases}$$

which is explained with Fig. 1.

Fig. 1(a) shows the sequential execution T^s of a primary task on a core C_1 and the sequential execution B^s of its backup copy on a core C_2 at S_{max}. Processing time activated in the fault-free status is depicted with filled area. In this case, T^s and B^s are overlapped at least for the time $2(E(1)-\frac{D}{2}$. The condition for overlapping is $E(n)>\frac{D}{2}$. The energy consumption in the fault-free status for T^s is $n\cdot(S_{max})^3\cdot E(n)=n\cdot E(n)=E(1)$, and that for B^s is $n\cdot(S_{max})^3\cdot 2\left(E(n)-\frac{D}{2}\right)=n\cdot 2\left(E(n)-\frac{D}{2}\right)=2\left(E(1)-\frac{D}{2}\right)$. Fig. 1(b) shows the parallel execution T^p of the primary task on two cores C_1 and C_2, and the parallel execution B^p of its backup copy on two cores C_3 and C_4 at S_{max}. In this case, T^p and B^p are not overlapped. The condition for non-overlapping is $E(n)\le\frac{D}{2}$. Decreased processing time of parallel processing eliminates the overlapping of T^s and B^s, and also generates slack time between the end of T^p and the start of B^p. Fig. 1(c) shows the minimum energy consumption case of Fig. 1(b) without overlapping, where the core speeds of C_1 and C_2 are scaled down to S at which T^p is completed exactly at the start of B^p. The scaled-down speed is $S=\frac{E(n)}{D-E(n)}=\frac{E(2)}{D-E(2)}$. The energy consumption of T^p for the processing time $\frac{E(n)}{s}$ is $(S)^3\cdot n\cdot\frac{E(n)}{s}=(S)^2\cdot n\cdot E(n)=(S)^2\cdot 2\cdot E(2)$ and that of B^p in the fault-free status is 0.

Lemma 1: If $\frac{p[\eta_2]}{\eta_1}>\left(\frac{\eta_2}{\eta_1}\right)^{\frac{1}{3}}$ for $1\le\eta_1<\eta_2\le\frac{N}{2}$, increment of n_m decreases energy consumption of T_m with the same processing time.

proof: When $n_m=\eta_1$, the processing time of T_m at core speed S_1 is $\frac{W}{S_1\cdot P[\eta_1]}$ and its energy consumption per unit time is $(S_1)^3\cdot\eta_1$. If $n_m=\eta_1$ is replaced with $n_m=\eta_2(>\eta_1)$, the processing time becomes $\frac{W}{S_1\cdot P[\eta_2]}\left(\le\frac{W}{S_1\cdot P[\eta_1]}\right)$. For the same processing time of T_m, scaled-down speed S_2 is selected to be $S_2=S_1\cdot\frac{P[\eta_1]}{P[\eta_2]}\le S_1$. Its energy consumption per unit time becomes $(S_2)^3\cdot\eta_2=\left(S_1\cdot\frac{P[\eta_1]}{P[\eta_2]}\right)^3\cdot\eta_2$. Because $\left(S_1\cdot\frac{P[\eta_1]}{P[\eta_2]}\right)^3\cdot\eta_2<(S_1)^3\cdot\eta_1$ where $\frac{P[\eta_2]}{P[\eta_1]}>\left(\frac{\eta_2}{\eta_1}\right)^{\frac{1}{3}}$ for

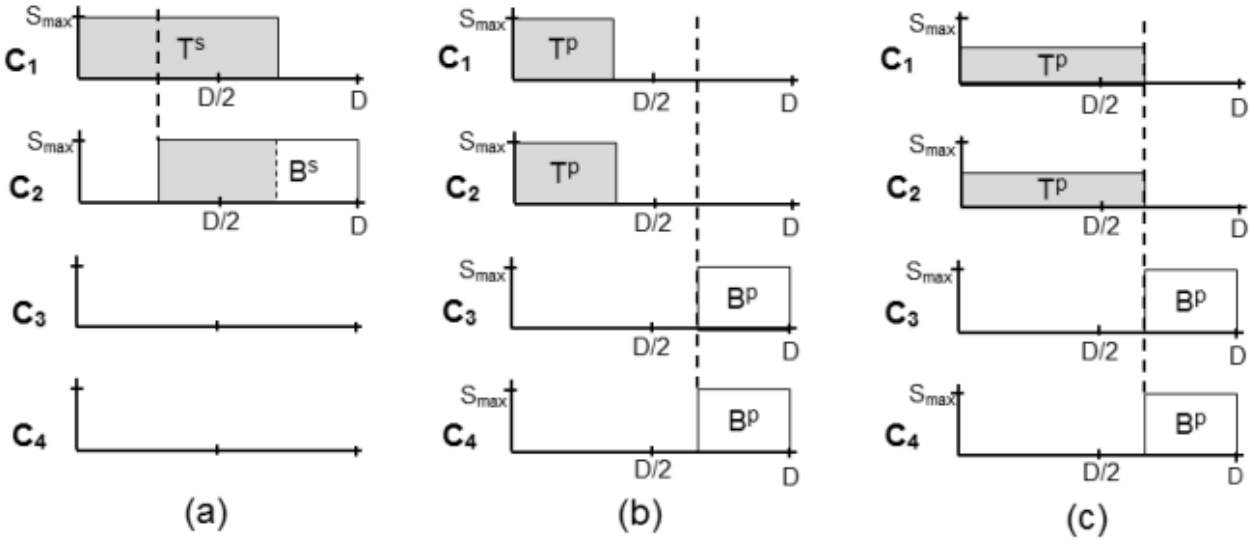

Fig. 1. Energy saving capability of parallel processing

$\eta_1 < \eta_2$, replacing $n_m = \eta_1$ with $n_m = \eta_2$ results in energy consumption decrement of T_m with the same processing time.

If $P_m[n_m] > (n_m)^{\frac{1}{3}}$, parallel processing reduces energy consumption of the primary task T_m by Lemma 1. It also reduces the activated time of the backup copy B_m in the fault-free status as shown in Fig. 1(b). On the other hand, it provokes increment of task utilization $U_m(n_m)$ for both T_m and B_m because $P_m[n_m] < n_m$, which may result in failing to satisfy the deadline constraint of other tasks due to lacking of available cores even at the maximum core speed. Total task utilization taken in charge by the $i^{[th]}$ core is called *core utilization* and denoted as $CU_i = \Sigma \frac{U_x(n_x)}{n_x} = \Sigma \frac{E_x(n_x)}{D_x}$ for all T_xs assigned to the i^{th}core. Then $\Sigma_{i=1}^{N} CU_i = \Sigma_{m=1}^{M} 2 \cdot U - M(n_m)$ because each task utilization value is distributed to some of core utilization values. A schedule fails to satisfy the deadline constraint if $CU_i > 1$ for any i or if $\Sigma CU_i = \Sigma\, 2 \cdot U_m(n_m) > N$.

Our study seeks to determine each n_m so that total energy consumption of all tasks would be minimized while satisfying the deadline and fault tolerance constraints of all tasks. Exhaustive assignments of n_m values for all cases require $\left(\frac{N}{2}\right)^M$ trials, which is too heavy to run for large M and N even at offline time. Moreover, minimizing total energy consumption of all tasks with fixed n_m values on multicore processors is known as a NP-hard problem [3]. Hence, the proposed scheme searches for a near minimum-energy schedule in a heuristic manner. In the first phase, the scheme calculates the ratio of the energy decrement quantity obtained when increasing n_m to the corresponding increment quantity of task utilization for $1 \le n_m < \frac{N}{2}$ and each T_m. The ratio can be formulated as below:

$$\Psi_m(n_m) = \frac{\psi_m(n_m) - \psi_m(n_m + 1)}{2 \cdot U_m(n_m + 1) - 2 \cdot U_m(n_m)}$$

In the second phase, the scheme searches for a near minimum-energy schedule using the above $\Psi_m(n_m)$ values. Initially, each n_m is set to 1. The scheme increases n_m of the task with the highest $\Psi_m(n_m)$ among $\Psi_1(n_1), \cdots, \Psi_m(n_M)$ by one, generates a schedule of all tasks using the EEFT method [1], and calculates its minimum energy consumption with constraint check. This procedure is repeated until $\Sigma\, 2 \cdot U_m(n_m) \ge N$ or $n_m = \frac{N}{2}$ for each m. In the final phase, the scheme selects a schedule with the least energy consumption among the schedules satisfying the deadline and fault tolerance constraints.

In the second phase, the scheme triggers the EEFT method at most $M \cdot \frac{N}{2}$ times, whereas exhaustive trials for all cases trigger it $\left(\frac{N}{2}\right)^M$ times. Because the EEFT method is designed to assign each task to only one core among dual cores, we modify it slightly to assign each task to one or more cores among multiple cores. The following pseudo-code describes the modified EEFT method operated with fixed n_ms on multiple cores.

1. Sort all T_ms and all B_ms in descending order of task utilization taken in charge by a single core (i.e., $\frac{U_m(n_m)}{n_m} = \frac{E_m(n_m)}{D_m}$).
2. Assign T_m to n_m cores which have the least core utilization among N cores. Assign B_m to n_m cores which have the least core utilization among $(N - n_m)$ cores except the n_m cores executing T_m.
3. Generate a schedule of each core with the maximum core speed, in which primary tasks are placed as soon as possible according to EDF (Earliest Deadline First) rule and backup copies are placed as late as possible according to EDL (Earliest Deadline Latest) rule.
4. Scale down the processing speed of primary tasks as much as possible using available slack time for each core.

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 [1] that reassigns all tasks to non-faulty cores after excluding the faulty core at run-time.

4. Evaluation

The proposed scheme is compared with the EEFT method [1] that executes each task on a single core. The

parallel processing speedup on n cores is initially set to $S[n]=(n-1)\times 0.8+1$, referred to as *Linear speedup*, because the speedups of many multimedia applications with massive computations are very close to the number of allocated cores [4]. To examine effects of lower parallel processing speedup, we also use two other speedup models: $S[n]=(n-1)\times 0.5+1$, called *Sublinear speedup*, and $S[n]=\sqrt{n}$, called *Square-root speedup*. The number of given periodic tasks is 16, where the number of primary tasks and their backup copies is 32. The deadline of each task is uniformly selected between 10 milliseconds and 1 second. The processing time of each task at the maximum core speed is synthetically generated between one millisecond and its deadline from a normal distribution. Average values of 1,000,000 task sets are displayed.

Fig. 2 shows the ratio of energy consumption in the proposed scheme to that in the EEFT method, called *Relative Energy Consumption*. In Fig. 2(a), N is fixed to 8. *System Load* denotes the ratio of total task utilization when $n_m=1$ to the number of all cores (i.e., $\frac{\sum 2\cdot U_m(1)}{N}$). The proposed scheme shows higher energy saving performance with higher parallel processing speedup. As System Load decreases, energy saving effect increases. When Linear speedup model is used and System Load = 10%, the proposed scheme saves about 82% energy consumption of the previous method. Fig. 2(b) shows energy saving effects of the number of available cores, where total task utilization when $n_m=1$ is fixed to $\sum 2\cdot U_m(1)\simeq 4$. The proposed scheme shows higher energy saving performance with higher parallel processing speedup. As N value increases, energy saving effect increases. When Linear speedup model is used and N = 32, the proposed scheme saves about 86% energy consumption of the previous method.

5. Conclusions

The proposed scheme enhances energy saving capability for real-time tasks while meeting their deadlines and tolerating permanent faults by exploiting parallel processing, whereas the related existing methods disregarded the energy saving potential of parallel processing. The scheme finds a near minimum-energy schedule within a polynomial time, because finding the minimum-energy schedule on multicore processors is a NP-hard problem. Evaluation results verify that the scheme consumes manifestly less energy than the state-of-the-arts method even with low parallel processing speedup as well as with high parallel processing speedup.

Reference

[1] 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.

[2] 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.

[3] 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, Vol.19, No.1, pp.1511-1525.

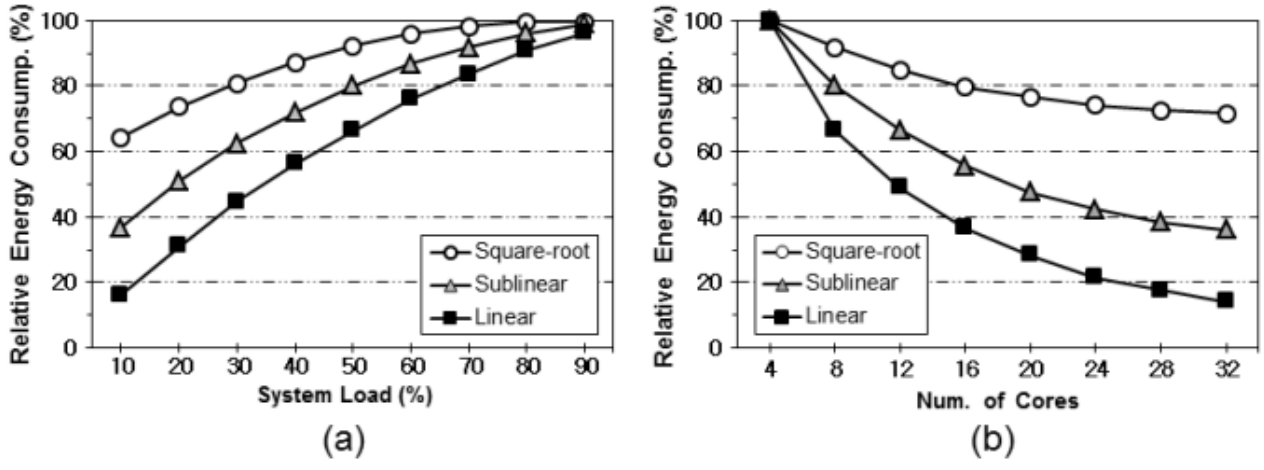

Fig. 2. Performance comparison against (a) System Load and (b) number of available cores

[4] W. Y. Lee, “Energy-efficient scheduling of periodic real-time tasks on lightly loaded multi-core processors”, *IEEE Trans. Parallel Distrib. Syst.*, 2012, Vol.23, No.3, pp.530-537.
[5] E. Meneses, O. Sarood and L. V. Kalé, “Assessing energy efficiency of fault tolerance protocols for HPC systems”, in Proceedings of Int’l Symp. Computer Archi. High Performance Computing, 2012, pp.35-42.
[6] Y. Guo, D. Zhu and H. Aydin, “Reliability-aware power management for parallel real-time applications with precedence constraints”, in Proceedings of Int’l Green Computing Conf., 2011, pp.1-8.

이 관 우

e-mail : kwlee@hansung.ac.kr
1994년 포항공과대학교 전자계산학과 (학사)
1996년 포항공과대학교 컴퓨터공학과 (공학석사)
2003년 포항공과대학교 컴퓨터공학과 (공학박사)
2003년~현 재 한성대학교 정보시스템공학과 부교수
관심분야 : Realtime Embedded System, Software Product Line, Aspect-Oriented Programming, IoT(Internet of Things)