Efficient Power Management Schemes for Dual-Processor Fault-Tolerant Systems

Yifeng Guo, Dakai Zhu
University of Texas at San Antonio
{yguo, dzhu}@cs.utsa.edu

Hakan Aydin
George Mason University
aydin@cs.gmu.edu

***Abstract*—To address the increasingly important energy efficiency problem, a Standby-Sparing scheme has been studied previously to reduce the energy consumption in dual-processor fault-tolerant systems. However, the dedicated (spare) processor in the Standby-Sparing scheme needs to execute backup tasks at full speed to preserve system reliability and thus cannot effectively exploit its available slack time. In this paper, based on our prior work of the *Preference-Oriented Earliest Deadline (POED)* scheduling algorithm, we study an *Energy-Efficient Fault-Tolerant (EEFT)* scheme for dual-processor real-time systems, which can effectively exploit the slack time on both processors for better energy savings. Specifically, the primary and backup tasks are first mapped in a mixed manner to both processors. Then, the POED algorithm is used to schedule the mixed tasks on each processor, where primary tasks are executed *as soon as possible* while backup tasks *as late as possible* to reduce the overlapped execution of primary and backup copies of the same task and thus reduce energy consumption. The online scheme that considers the additional slack from the cancellation of backup tasks and early completion of primary tasks is further explored. The proposed EEFT schemes are evaluated through extensive simulations. The results show that, compared to the state-of-the-art Standby-Sparing scheme, significant (more than 40%) energy savings can be obtained under the POED-based EEFT schemes.**

I. Introduction

Reliability and fault tolerance techniques have been the traditional research focus for computing systems [20], where different types of faults may occur at runtime due to various reasons (such as hardware defects, electromagnetic interference and cosmic ray radiations). For example, *permanent faults* may bring a system component (i.e., the processor) to halt and can not be recovered without some form of hardware redundancy while *transient faults* are often triggered by temporary causes and will disappear after a short time interval. With reduced technology size and increased level of integration, it is said that modern computing devices are more susceptible to such faults [16] and it is imperative to incorporate fault tolerance techniques, especially for the mission-critical embedded systems.

Note that, for embedded systems that have been widely adopted in ICU patient monitoring systems, electrical control systems in automobiles (such as ABS systems) or Unmanned Aerial Vehicles (UAVs), they normally have various timing constraints and real-time scheduling theories have also been studied extensively in the past decades. Energy management has been considered another important issue in embedded systems, where the system components can be put into idle/sleep states when they are not in use through the *Dynamic Power Management (DPM)* technique [22]. Moreover, *Dynamic Voltage Scaling* (DVS) can scale the supply voltage and processing frequency of processors and other components simultaneously to save energy [24]. Although both fault tolerance [4], [7], [11], [12] and energy management [1], [17], [18], [24], [28] have been studied extensively (but independently) for real-time embedded systems, the research on the co-management of reliability and energy is rather limitted.

Resent research also shows that the probability of having transient fault increases drastically when systems are operated at lower supply voltages [10], [29]. By incorporating such negative effects of DVS on system reliability, the *Reliability-Aware Power Management (RAPM)* framework has been studied, which exploits time-redundancy for both reliability preservation and energy savings [14], [21], [25], [27]. Here, the main idea is to schedule a separate recovery job for every job that has been selected to scale down to preserve the system's original reliability, which is assumed to be satisfactory. The recovery job is executed at the maximum speed only when its scaled primary job incurs transient faults. Although the RAPM schemes can preserve the system reliability with respect to transient faults, there is no consideration for permanent faults. Moreover, due to the separate allocation of the recovery job on the same processor with its primary job, large tasks with utilization greater than 50% cannot be managed under RAPM, which limits its applicability.

Recently, by considering both reliability preservation due to transient faults and tolerating one permanent fault, Ejlali *et al.* proposed a Standby-Sparing technique for applications running on a dual-processor system [9]. Specifically, the primary tasks are executed on the main processor with DVS based energy management, while the spare processor is reserved for executing *backup* tasks, if needed. Upon the successful completion of a primary task, the corresponding backup task is cancelled and the excessive energy consumption is avoided. However, should the primary task fail due to either permanent or transient faults, the backup runs to completion at the maximum speed. Thus, a single permanent fault on any processor can be tolerated and the system's original reliability (in terms of resilience with respect to transient faults) is also preserved [9]. Moreover, by statically allocating backup tasks on the spare processor to minimize the overlap between primary and backup copies of the same task, it can reduce energy consumption as well. Considering the backup tasks will most likely be cancelled without even being executed since real-time tasks typically complete early, more energy savings

This work was supported in part by NSF awards CNS-0855247, CNS-1016855, CNS-1016974, and NSF CAREER Awards CNS-0953005.

can be expected at runtime.

However, the above work considers only *nonpreemptive* scheduling for *aperiodic* tasks. In [15], Haque *et al.* extended the Standby-Sparing technique for *preemptive periodic* real-time applications. Here, the *earliest deadline first (EDF)* scheduling is applied on the primary processor while the backup tasks are executed on the spare processor according to the *earliest deadline latestest (EDL)* scheduling policy [8]. By delaying the backup tasks as much as possible and obtaining idle intervals as early as possible, the joint use of EDF and EDL on the primary and spare processors, respectively, provides better opportunities to reduce the overlapped execution of the primary and backup copies of the same task at runtime and thus reduce the energy consumption. Note that, by dedicating a spare processor for backup tasks that need to be executed at the maximum processing speed, the Standby-Sparing scheme cannot effectively exploit the available slack time on the spare processor for better energy savings.

In this work, instead of dedicating a spare processor for backup tasks, we treat both processors equally and schedule a mixed set of primary and backup copies of different tasks on each of them. Then, based on our prior work of the *Preference-Oriented Earliest Deadline (POED)* scheduling algorithm [13], we study the *energy efficient fault tolerance (EEFT)* schemes. Specifically, on each processor, primary tasks are executed *as soon as possible* while backup tasks *as late as possible* under POED. That is, the slack time on both processors can be efficiently utilized to minimize the overlapped execution of the primary and backup copies of the same task for better energy savings. Simulation results show that, compared to the state-of-the-art Standby-Sparing scheme [15], significant (up to 40%) energy savings can be obtained under the POED-based EEFT schemes.

The remainder of this paper is organized as follows. Section II presents system models and state our assumptions. In Section III, the basic idea of the proposed scheme is illustrated through a concrete example, followed by the brief review of the preference-oriented earliest deadline (POED) scheduling algorithm. The proposed energy-efficient fault-tolerance (EEFT) schemes are discussed in Section IV. Section V presents the evaluation results and Section VI concludes the paper.

II. Models and Assumptions

A. Task Model

We consider a set of n periodic real-time tasks $\Psi = \{T_1, \ldots, T_n\}$ to be executed on a dual-processor system. Each task T_i is represented as a tuple (c_i, p_i), where c_i is its worst-case execution time (WCET) under the maximum available CPU frequency, and p_i is its period. The utilization of a task T_i is defined as $u_i = \frac{c_i}{p_i}$. The system utilization of a given task set is the summation of all task utilizations: $U = \sum_{T_i \in \Psi} u_i$.

Tasks are assumed to have implicit deadlines. That is, the j^{th} task instance (or job) of T_i, denoted as $T_{i,j}$, arrives at time $(j-1) \cdot p_i$ and needs to complete its execution by its deadline at $j \cdot p_i$. Note that, a task has only one active task instance at any time. When there is no ambiguity, we use T_i to represent both the task and its current task instance.

During the operation of a computing system, both *permanent* and *transient* faults may occur due to, for instance, the effects of hardware defects or cosmic ray radiations, which can result in system *errors*. Therefore, for reliability and fault tolerance, each task T_i has a backup task B_i, which has exactly the same timing parameters (i.e., c_i and p_i) as its primary task T_i. For consistency, the j^{th} task instance (or job) of B_i is denoted as $B_{i,j}$.

To tolerate a permanent fault in such a dual-processor system, the backup task B_i needs to be allocated to the processor that is different from the one of its corresponding primary task T_i. Moreover, to incorporate the negative effects of DVS on the rate of transient faults when executing the task T_i at a scaled frequency, the backup task B_i should be executed at the maximum processing frequency to preserve the task T_i's *original* reliability with respect to transient faults. Here, the original reliability is defined as the probability of finishing T_i's execution successfully when no energy management (i.e., DVS) is applied [26].

In addition, it is assumed that soft errors are detected with *sanity* (or *consistency*) checks at the end of a task's execution [19]. The overhead for fault detection is also assumed to be incorporated into tasks' WCETs.

B. Power Model

Each processor in the dual-processor system is assumed to have DVS capability, which has been a common feature in modern processors. We further assume that the processors can only be operated at L different discrete frequency levels $\{f_1, f_2, \ldots, f_L\}$, where $f_L = f_{max}$ is the maximum frequency. Moreover, the time overhead for frequency (and supply voltage) changes is assumed to be incorporated into the WCETs of tasks [28].

With increasing importance of leakage power and emphasis on the need for considering all system components in power management [1], [17], [18], we adopt a system-level power model in previous RAPM research for each processor in dual-processor systems [27], which can be expressed as:

$$P = P_s + P_{ind} + C_{ef} \cdot f^3 \tag{1}$$

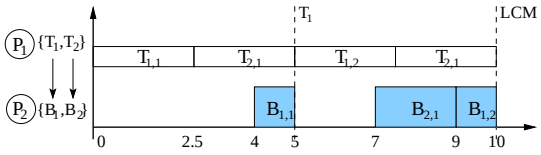
Here, P_s stands for *static power*, which can be removed only by powering off the whole system. Due to the prohibitive overhead of turning off and on the system in periodic real-time execution settings, we assume that the system is in *on* state at all times and that P_s is always consumed. Hence, we will focus on the energy consumption related to active power, which is given by the last two items in the above equation. P_{ind} is the *frequency-independent active power*. The *frequency-dependent active power* depends on the system-dependent constants C_{ef} and m, as well as f, which can be managed through the dynamic voltage scaling (DVS) technique [6]. When no task is active, we assume that both active powers can be efficiently removed by putting the processor (or system) in sleep states with the dynamic power management (DPM) technique.

III. An Example and POED Scheduling

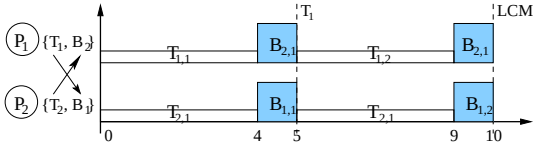
In this section, we first introduce the basic ideas of our *energy efficient fault tolerance (EEFT)* scheme for a dual-processor system through a concrete example. Then, we briefly review the essential principles and basic steps of the underlying *preference-oriented earliest deadline (POED)* scheduling algorithm, which serves as the cornerstone for our proposed EEFT schemes.

A. A Motivational Example

We consider a dual-processor system with two periodic real-time tasks where $T_1 = (1, 5)$ and $T_2 = (2, 10)$. As we discussed in the last section, for reliability and fault tolerance purpose, there are two *backup* tasks B_1 and B_2, which correspond to the *primary* tasks T_1 and T_2, respectively. Here, the primary and backup copies of the same task have to be scheduled on different processors. Once a task's primary copy completes successfully, its backup copy can be cancelled. Therefore, for energy efficiency, we should postpone the execution of a task's backup copy as much as possible and minimize the overlapped execution with its primary task that is running on another processor [23].



a. Schedules for Standby-Sparing scheme



b. Schedules for POED-based scheme

Fig. 1. A dual-processor fault-tolerant system with two tasks: $T_1 = (1, 5)$ and $T_2 = (2, 10)$.

Specifically, for dual-processor fault-tolerant systems, the *Standby-Sparing* scheme has been recently studied [15]. Here, as shown in Figure 1a, all primary tasks are executed on one processor under EDF at a scaled frequency, while all backup tasks are scheduled on the secondary (spare) processor under EDL for energy savings. Recall that, to address the negative effects of voltage scaling on system reliability, the executions of tasks' backup copies are not scaled and the slack time on the secondary processor is wasted [15].

To efficiently utilize the slack time on both processors, instead of dedicating one processor for backup tasks, we can partition the primary copies of tasks among the processors and allocate the corresponding backup copies on another processor. That is, as shown in Figure 1b, each processor will have a *mixed* set of primary and backup copies of different tasks. Here, on each processor, we can exploit the *preference-oriented earliest deadline (POED)* scheduling algorithm, which can execute the scaled primary copy *as soon as possible (ASAP)* while the non-scaled backup copy of tasks *as late as possible (ALAP)*.

With such separate considerations of primary and backup copies of tasks in POED, this scheme can efficiently exploit the slack time on *both* processors while significantly reducing the overlapped execution of a task's primary and backup copies, which in turn achieves better energy savings. For this particular example, if we use the same power parameters as in [15], the POED-based EEFT scheme can save 20% more energy to that of the Standby-Sparing scheme.

B. Preference-Oriented Earliest Deadline (POED) Scheduling

As we can see from the above example, in order to improve the energy efficiency on dual-processor fault-tolerant systems, the mixed primary and backup tasks on each processor need to be scheduled according to different *preferences* in addition to meeting all tasks' deadlines. The concept of *preference-oriented execution* was first proposed in [13] for independent periodic tasks executing on a single processor system. The motivation for designing this scheduling algorithm comes from the recent application requirement of real-time systems, such as fault-tolerant real-time systems with low power consumption requirements, and the scheduling of mixed-criticality tasks [3] to provide better service to low-criticality tasks.

There are two basic principles in preference-oriented scheduling. First, if it is possible to do so without causing any deadline miss, the tasks with ASAP preference should be executed before the ones with ALAP preference even if the ASAP task has a later deadline. Second, the execution of tasks with ALAP preference should be delayed as much as possible given that it does not cause deadline miss for both current and future tasks [13].

Following these two principles, the *preference-oriented earliest deadline (POED)* scheduling algorithm tries to execute tasks with ASAP preference in increasing order of their deadlines while ensuring that there is no deadline miss for tasks with ALAP tasks. When there is no active task with ASAP preference, POED will try to let the processor idle before executing tasks with ALAP preference also in the increasing order of their deadlines [13]. We have shown that the POED scheduling algorithm can guarantee that there is no deadline miss while achieving better preference values for all tasks. For the detailed steps and analysis of the POED scheduling algorithm, we refer interested readers to [13].

IV. POED-Based Energy Efficient Fault Tolerance (EEFT) Schemes for Dual-Processor Systems

In this section, based on the POED scheduling algorithm, we discuss the basic steps of the energy-efficient fault-tolerance (EEFT) schemes for dual-processor systems. To effectively exploit the available slack time on both processors, the essential idea of POED-based EEFT schemes is to treat two processors equally. That is, each processor will be allocated a mixed set of primary and backup copies of different tasks, which are scheduled according to the POED scheuding algorithm with

primary tasks having the ASAP preference while backup tasks having the ALAP preference.

Algorithm 1 Basic Steps for the POED-Based EEFT Schemes

1: **Input:** task set $\Psi = \{T_1, \ldots, T_n\}$ with $U \leq 1$;
2: **Step 1:** Partition tasks in Ψ to two processors according to a given (e.g., WFD) heuristic;
3: **Step 2:** For each (primary) task T_i, allocates its backup task B_i to another processor;
4: **Step 3:** On each processor, calculate the scaled frequency for its primary tasks using the available static slack;
5: **Step 4:** Run the mixed set of primary and backup tasks on each processor under POED; when a task completes successfully, cancel its corresponding copy on another processor properly;

The major steps of the POED-based EEFT schemes are summarized in Algorithm 1. The set Ψ of real-time tasks, which consists of only primary tasks under consideration, is first partitioned onto the two processors in the system following any given heuristic (line 2). Note that, the partition with balanced workload on processors is preferred regarding to power management. Since the *Worst-Fit Decreasing (WFD)* heuristic based on tasks' utilizations has been shown to have better performance on generating balanced partitions (Theorem 3 in [2]), we adopt the WFD heuristic in this work.

Then, for each task T_i in Ψ, its backup task B_i is allocated to another processor, which is different from T_i's processor (line 3). As we can see that, after this step, each processor will have a mixed set of primary and backup tasks. When the system is not fully loaded with the system utilization of the task set Ψ being $U < 1$, the remaining processor capacity (i.e., static slack) on both processors can be exploited to scale down the processing frequency of the primary tasks on each processor respectively (line 4).

Finally, the mixed primary and backup tasks on each processor are scheduled under the POED algorithm, where primary tasks have ASAP preference and backup tasks have ALAP preferene (line 5). Note that, with the scaled processing frequency of primary tasks, the inflated system utilization of tasks on each processor is no more than 1. From [13], we know that all (primary and backup) tasks on both processors can meet their deadlines under the POED scheduling algorithm.

Note that, not all (especially backup) tasks need to run to completion. When a task (either primary or backup) passes its sanity check and completes successfully at runtime, it should notify another processor to cancel the execution of its corresponding (backup or primary) task for energy efficiency.

A. Online Extension of POED-Based EEFT

The cancellation of the execution of backup tasks can generate significant amount of dynamic slack at runtime. Moreover, real-time tasks typically take much less time than their WCETs and complete early, which can contribute dynamic slack as well. Such dynamic slack can be exploited to further scale down the processing frequency of primary tasks and/or delay the execution of future backup tasks for better energy savings. As shown in Section V, the online version of the POED-based dynamic power management can drastically reduce the unnecessary overlapped execution of the two copies of the same task and thus lead to significantly more energy savings, when compared to that of the Standby-Sparing scheme [15].

V. Simulations and Evaluations

In this section, we evaluate the performance of the proposed POED-based EEFT schemes through extensive simulations. For such purpose, we developed a discrete event simulator using C++. We have implemented both the static (offline) and dynamic (online) versions of the POED-based EEFT schemes, which are denoted as *POED-SPM* and *POED-DPM*, respectively. For comparison, we also implemented the Standby-Sparing scheme with offline calculated scaled frequency for primary tasks (denoted as *SS-SPM*) and its online version of the ASSPT scheme that exploits online slack to further reduce the processing frequency of primary tasks (denoted as *SS-DPM*) [15]. Moreover, the *basic Standby-Sparing (Basic-SS)* scheme that does not scale down any task is used as the baseline during the evaluation. Here, all schemes cancel the execution of backup tasks once the corresponding primary tasks complete successfully.

The task sets under consideration have either 10 or 20 tasks, where the utilization of each task is generated using the *UUniFast* scheme proposed in [5]. The period of each task is uniformly distributed in the range of $[10, 100]$ and the WCETs of tasks are set accordingly. Since most modern processors have a few frequency levels, in the evaluations, we assume that there are four normalized frequency levels (which are $\{0.4, 0.6, 0.8, 1.0\}$). Moreover, the same as in [15], we assume that $C_{ef} = 1$ and $m = 3$.

Note that, all schemes under consideration can tolerate a permanent fault. Moreover, by enforcing the backup tasks to be executed at the maximum processing frequency, the original system reliability with respect to transient faults can be preserved as well under all schemes. Therefore, we focus on evaluating the energy efficiency of the schemes. We vary the system load (i.e., utilization U) and dynamic load (i.e., the average-to-worst case execution time ratio of tasks) during the simulations. The actual execution time of each task is generated based on the average-to-worst case execution time ratio. Each data point in the figures corresponds to the average result of 100 task sets. All experiments were conducted on a Linux box with an Intel Xeon E5507 ($2.27GHz$) processor and 32 GB of memory.

Figure 2 first shows the normalized energy consumption under different schemes with varying system loads, where the one under *Basic-SS* is used as the baseline. Since similar results are obtained for task sets with 10 and 20 tasks, we will only show results for the case of 20 tasks in this paper. Here, all tasks take their WCETs (that is, the average-to-worst case execution time ratio is 1) at runtime. In general, the normalized energy consumption increases with the system

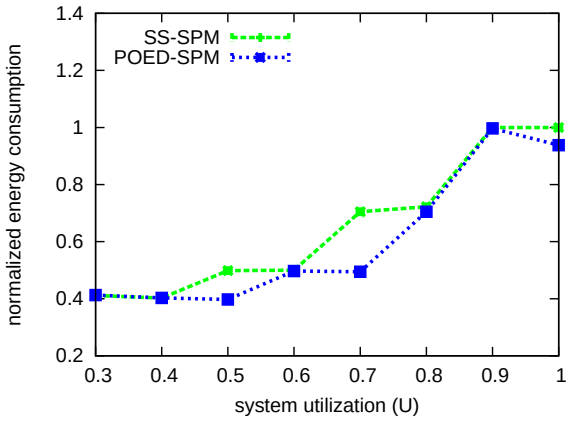


Fig. 2. Normalized energy consumption *vs.* static load; 20 tasks per set

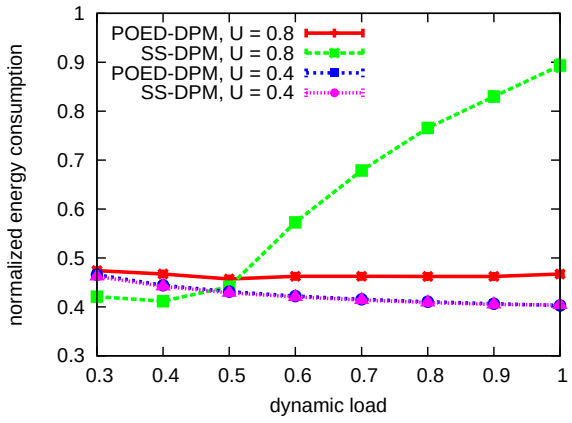


Fig. 3. Normalized energy consumption *vs.* dynamic load; 20 tasks per set

utilization, since the scaled frequency for primary tasks gets higher. However, the *POED-SPM* scheme acheives better energy savings as it can better exploit the slack on both processors and delay the execution of backup tasks. When system utilization $U = 0.9$, the static scaled frequency for primary tasks will be 1.0 under all schemes due to discrete frequency limitation. Therefore, the additional energy savings are limited under *POED-SPM*.

The energy savings under the schemes for various dynamic loads are also evaluated and the results are shown in Figure 3. Here, we can see that, when the dynamic load is low (i.e., in the range of 30% to 50%), the energy savings under all schemes are close to each other. The reasons are: a) most primary tasks are executed at the lowest frequency; and b) most backup tasks will be cancelled.

However, for the case of high system load $U = 80\%$, the normalized energy consumption under *SS-DPM* increases quickly with the increased dynamic load. This comes from the the fact that the greedy slack utilization strategy applied in *SS-DPM* delays the finish time for primary tasks in the primary processor, which directly causes more overlapped execution of backup tasks on the spare processor. In comparison, the normalized energy consumption under our proposed *POED-DPM* scheme is relatively stable since it can maximally delay the execution of backup tasks on both processors and thus reduced the overlappped execution of primary and backup of the same task. Therefore, if online slack is utilized to further scale down the processing frequency of primary tasks, significantly more energy (up to 40%) can be saved under *POED-DPM* when compared to that of the SS-DPM.

VI. Conclusions and Future Works

For real-time tasks running on a dual-processor system, we study the *energy-efficient fault-tolerance (EEFT)* schemes based on our prior work of the *preference-oriented earliest deadline (POED)* scheduling algorithm. Different from the existing Standby-Sparing scheme, which dedicates one processor for backup tasks, the objective is to effectively exploit the slack time on both processors for better energy savings. We illustrate the basic ideas through a concrete example and discuss the major steps of the POED-based EEFT schemes. By executing the primary tasks *as soon as possible* and backup tasks *as late as possible* under the POED scheduling algrithm, the schemes can siginficantly reduce the overlapped execution of primary and backup copies of the same task and thus lead to more energy savings. Our simulation results show that, compared to that of the state-of-the-art Standby-Sparing scheme, the POED-based EEFT schemes can obtain significantly more (up to 40%) energy savings.

For our future work, we will evaluate the effects of additional DVS transition under the POED-based EEFT schemes and extend such schemes to systems with more than two processors.

References

[1] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proc. of The 27^{th} IEEE Real-Time Systems Symposium (RTSS)*, pages 313–322, 2006.

[2] H. Aydin and Q. Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, page 9 pp., april 2003.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Trans. on Computers*, 2011.

[4] A. A. Bertossi, L. V. Mancini, and F. Rossini. Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems. *Parallel and Distributed Systems, IEEE Transactions on*, 10(9):934 –945, sep 1999.

[5] E. Bini and G.C. Buttazzo. Biasing effects in schedulability measures. In *Proc. of the Euromicro Conf. on Real-Time Systems*, 2004.

[6] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of HICSS Conference*, 1995.

[7] J.-J. Chen, C.-Y. Yang, T.-W. Kuo, and S.-Y. Tseng. Real-time task replication for fault tolerance in identical multiprocessor systems. In *Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 249–258, Washington, DC, USA, 2007. IEEE Computer Society.

[8] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Trans. Softw. Eng.*, 15:1261–1269, 1989.

[9] A. Ejlali, B. M. Al-Hashimi, and P. Eles. A standby-sparing technique with low energy-overhead for fault-tolerant hard real-time systems. In *Proc. of the IEEE/ACM Int'l Conf. on Hardware/Software Codesign and System Synthesis*, pages 193–202, New York, NY, USA, 2009.

[10] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *Micro, IEEE*, 24(6):10 –20, nov.-dec. 2004.

[11] S. Ghosh, R. Melhem, and D. Mossé. Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, 8(3):272 –284, March 1997.

[12] S. Gopalakrishnan and M. Caccamo. Task partitioning with replication upon heterogeneous multiprocessor systems. In *Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 199 – 207, april 2006.

[13] Y. Guo, H. Su, D. Zhu, and H. Aydin. Preference-oriented scheduling framework and its application in fault-tolerant real-time systems (extended version). Technical report, CS-TR-2012-009, Dept. of Computer Science, Univ. of Texas at San Antonio, 2012. available at http://www.cs.utsa.edu/˜dzhu/poed-tr.pdf.

[14] Y. Guo, D. Zhu, and H. Aydin. Reliability-aware power management for parallel real-time applications with precedence constraints. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1 –8, july 2011.

[15] M. Haque, H. Aydin, and D. Zhu. Energy-aware standby-sparing technique for periodic real-time applications. In *Proc. of the IEEE International Conference on Computer Design (ICCD)*, 2011.

[16] P. Hazucha and C. Svensson. Impact of cmos technology scaling on the atmospheric neutron soft error rate. *IEEE Trans. on Nuclear Science*, 47(6):2586–2594, 2000.

[17] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. In *Proc. of The* 14^{th} *Symposium on Discrete Algorithms*, 2003.

[18] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system wide energy minimization in real-time embedded systems. In *Proc. of the Int'l Symposium on Low Power Electronics and Design (ISLPED)*, pages 78–81, 2004.

[19] D. K. Pradhan. *Fault Tolerance Computing: Theory and Techniques*. Prentice Hall, 1986.

[20] D. K. Pradhan, editor. *Fault-tolerant computer system design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[21] X. Qi, D. Zhu, and H. Aydin. Global scheduling based reliability-aware power management for multiprocessor real-time systems. *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, 47(2):109–142, 2011.

[22] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. *System-Level Design Techniques for Energy-Efficient Embedded Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[23] O. S. Unsal, I. Koren, and C. M. Krishna. Towards energy-aware software-based fault tolerance in real-time systems. In *Proc. of the Int'l Symp. on Low Power Electronics and Design*, pages 124–129, 2002.

[24] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[25] B. Zhao, D. Zhu, and H. Aydin. Enhanced reliability-aware power management through shared recovery technique. In *Proc. of the IEEE/ACM Int'l Conference on Computer Aided Design (ICCAD)*, 2009.

[26] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 397 – 407, 2006.

[27] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. *IEEE Trans. on Computers*, 58(10):1382–1397, 2009.

[28] D. Zhu, R. Melhem, and B. R. Childers. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. *IEEE Trans. on Parallel and Distributed Systems*, 14(7):686–700, 2003.

[29] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the Int'l Conf. on Computer Aidded Design*, pages 35–40, 2004.