Fixed-Priority Allocation and Scheduling for Energy-Efficient Fault Tolerance in Hard Real-Time Multiprocessor Systems

Tongquan Wei, Piyush Mishra, *Member*, *IEEE*, Kaijie Wu, *Member*, *IEEE*, and Han Liang

Abstract—Energy-efficient task allocation and scheduling schemes with deterministic fault-tolerant capabilities are proposed for symmetric multiprocessor systems executing tasks with hard real-time constraints. The proposed heuristic achieves energy savings by optimally balancing the application workload among processors in a system. Based on the observation that a fault-free operation is expected to remain dominant in the near future and the probability of the worst case faults is low, an optimistic fault-tolerant heuristic is then proposed to achieve optimum energy savings in the absence of faults and meet application timing requirements in the worst case faults at the cost of energy inefficiency. Extensive simulation experiment results show that when compared to the state-of-art schemes, the proposed optimistic heuristic achieves average energy savings of up to 70 percent and exhibits higher tolerance to variations in application utilizations and more resilience to fault occurrences beyond system specification.

Index Terms—Energy-aware systems, multiprocessor systems, real-time and embedded systems, scheduling and task partitioning.

1 Introduction

An increasing number of embedded systems are being deployed in mission-critical applications with hard real-time constraints, such as navigation, process control, automated surveillance, and system monitoring. Many of these systems such as satellite-based parallel signal processing, parallel computing in sensor networks, and automated target recognition (ATR) are rich in thread-level parallelism (TLP). Unlike applications rich in instruction-level parallelism (ILP), whose timing requirements are usually satisfied by high-performance uniprocessors, TLP-rich applications are better suited to multiprocessor systems where multiple threads can be executed simultaneously on different processors to provide high computing throughput.

A scheduler in hard real-time multiprocessor systems operates in three phases. In phase 1, tasks are allocated and bounded to processors. In phase 2, tasks allocated to an individual processor are scheduled according to some scheduling scheme, and in phase 3, the scheduler prepares the tasks to be dispatched for execution. Real-time task allocation and scheduling in symmetric multiprocessor (SMP) systems is a well-studied problem, and traditionally, the focus has been on trading off the feasibility and performance with resource requirements. Of late, the number of faults in hardware has been rising continuously due to increasing complexity of design, aggressive technology scaling, and extreme operating conditions. In particular, susceptibility to transient faults has been on the rise due to the increasing level of integration, reducing size of transistor features, lowering voltage levels in integrated circuits, and harsh operating environments [1], [2]. The probability of fault occurrences is even higher in a multiprocessor system as a result of the large number of components and increased design complexity. Since hard real-time systems demand both temporal and logical correctness, it is essential that even in the presence of faults all tasks finishes execution before their respective deadlines. Therefore, real-time embedded systems deployed in mission- or safety-critical applications are typically designed with enough margins to tolerate the worst case expected number of faults by trading off fault coverage and fault detection latency with system performance. Since multiprocessor systems are more amenable to fault-tolerant techniques due to their inherent redundancy, several techniques have been developed with varying the levels of granularity: 1) triple modular redundancy (TMR), 2) primary backup (PB), and 3) checkpointing scheme. The TMR technique tolerates faults by running three copies of a task concurrently and voting on the results from these copies. In the PB approach, two copies of a task are executed serially on two different processors, and an acceptance test is performed to check the result. The backup copy is executed only if the output of the primary version fails the acceptance test. Checkpointing in conjunction with backward error recovery is a fault-tolerant strategy that allows a processor to rollback to its previously known valid state to resume normal execution by exploiting the available slack time. Though the checkpointing strategy incurs overhead even in the absence of faults, it has the advantages of small fault detection latency and recovery costs.

With the proliferation of battery-operated embedded real-time systems, the need for an energy-efficient design is

- *T. Wei is with the Department of Electrical and Computer Engineering, Michigan Technological University, EERC 827, 1400 Townsend Drive, Houghton, MI 49931. E-mail: twei@mtu.edu.*
- *P. Mishra is with the Electronics and Energy Conversion Group, GE Global Research, Niskayuna, NY 12309. E-mail: mishrapi@ge.com.*
- *K. Wu and H. Liang are with the Department of Electrical and Computer Engineering, University of Illinois, 1020 SEO, 851 South Morgan St., Chicago, IL 60607. E-mail: {kaijie, hliang}@ece.uic.edu.*

Manuscript received 8 Oct. 2007; revised 1 Apr. 2008; accepted 10 June 2008; published online 7 July 2008.
Recommended for acceptance by I. Ahmad, K.W. Cameron, and R. Melhem.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2007-10-0360.
Digital Object Identifier no. 10.1109/TPDS.2008.127.

increasing to reduce power density and enhance the operational lifetime of battery-powered systems. Numerous techniques have been proposed for energy minimization at the hardware level [3], software level [4], and system level [5]. At the hardware level, energy saving is achieved via various circuit design optimization strategies, while at the software level, energy-efficient modes are selected using a software controller. At the system level, energy efficiency is achieved by dynamically reconfiguring active components of a processor and selectively turning off idle components. One such promising power management technique is Dynamic Voltage Scaling (DVS) that exploits technological advances in power supply circuits to reduce energy consumption. DVS has attracted considerable attention due to its effectiveness and ease of use [6], [7]. It reduces the energy consumption of a processor by dynamically scaling down the processor voltage at the cost of increased execution time. Therefore, in reliable real-time multiprocessor systems, task allocation and scheduling schemes can have a significant impact on system energy consumption, performance, and fault-tolerant capabilities.

This paper focuses on energy-efficient allocation and scheduling schemes that jointly optimize DVS policies and feasibility performance to tolerate transient faults using the checkpointing strategy in hard real-time SMP systems. It is assumed that tasks are periodic, independent, and preemptive. Simulation results show that the proposed technique achieves significant energy savings under varying workloads and fault conditions when compared to state-of-art schemes while maintaining comparable feasibility performance.

1.1 Related Work

Energy-efficient reliability is an active area of research, and recently, much attention has been paid to adapt the techniques for real-time systems. One such scheme proposed by Melhem et al. exploited slacks in task schedules to reduce energy consumption while tolerating faults based on DVS [8]. It made a simplifying assumption that processor frequency can be scaled in a continuous range. Zhang and Chakrabarty proposed energy-efficient techniques based on the offline scheduling algorithm Rate Monotonic (RMA) to tolerate faults in hard real-time systems under the simplifying assumptions that the overhead for checkpointing, rollback recovery, and DVS is negligible [9]. This scheme was improved in [10] to overcome simplistic assumptions but at the cost of increased complexity in the algorithm. Wei et al. proposed low-cost offline algorithms that combine feasibility analysis and voltage scaling for hard real-time systems based on the exact characterization of RMA (ECRMA) [11]. However, most of it focuses on uniprocessor systems.

On the other hand, an increasing number of emerging applications are adopting multiprocessor and multicore real-time embedded systems due to the distributed nature of applications, demands for higher performance and reliability, low power, and overall greater flexibility of operation. Fault tolerance in a real-time multiprocessor system is typically achieved through the PB approach where two copies of a task run on different processors [12]. Oh and Son proposed a scheduling scheme for periodic tasks in a multiprocessor system [14]. In this scheme, a backup schedule is created for each task in a primary schedule such that the two schedules are on different processors, and they do not overlap. Mosse et al. described a fault-tolerant scheduling algorithm based on primary-backup, backup overloading, and backup deallocation to tolerate at most one single transient or permanent fault at any instant of time in the entire system [13]. It schedules more than one backup for a task in the same time slot on the same processor and reclaims the resources reserved for the backup task when the corresponding primary copy completes successfully. In [15], multiple copies of dynamic, aperiodic, and non-preemptive tasks are scheduled on processors, and overloading and deallocation strategies are used to achieve high feasibility performance and resource utilization by adapting to runtime fault occurrences.

Energy-efficient multiprocessor allocation and scheduling under hard real-time constraints has also been an active research topic in the past decade. In [16], energy-efficient offline task allocation and scheduling schemes were proposed to minimize the energy consumption of an aperiodic task set. The tasks in a given task set are assigned to processors using a round-robin-based approach and the allocated tasks are optimally scheduled on individual processors. Zhu et al. proposed energy-efficient heuristics for scheduling periodic task sets sharing a common deadline [17]. Energy saving is achieved for dependent and independent tasks by using the shared slack reclamation on DVS-capable processors. In [19] and [20], energy-efficient approximation algorithms were proposed for scheduling a set of independent periodic tasks sharing a common deadline on multiprocessor systems. It was assumed that processors are homogeneous and support continuous voltage levels. The authors of [18], [21], and [22] proposed Earliest Deadline First (EDF)-based online energy-efficient scheduling policies to schedule independent periodic tasks on multiprocessor systems. Although the online EDF algorithm dominates over fixed-priority offline scheduling algorithms such as RMA, in feasibility performance, fixed-priority algorithms are of great practical importance, and most real-time scheduling algorithms, especially hard real-time systems, use fixed-priority assignments due to their low overhead and predictability [23]. Therefore, the work in [22] was extended in [24] to systems using RMA to schedule independent tasks on processors. However, the proposed scheme suffers from significant degradation in feasibility performance under a light system workload.

1.2 Outline

A novel task allocation heuristic combined with an RMA-based scheduling scheme is proposed to jointly optimize energy consumption and fault tolerance in hard real-time multiprocessor systems while preserving the feasibility of tasks. The proposed offline scheme achieves energy efficiency based on the observation that in real life, faults seldom occur, and the traditional approach of designing for corner cases is highly inefficient. To the best of our knowledge, this is the first work that addresses energy efficiency and fault tolerance in hard real-time multiprocessor systems.

The rest of the paper is organized as follows: Section 2 describes the system architecture, the application model, the fault model, and the energy model. Section 3 discusses the state-of-art in energy management strategies for hard real-time multiprocessor systems. Section 4 proposes an energy-efficient fault-tolerant allocation scheme that achieves optimum energy savings in the absence of faults and sacrifices

energy efficiency to meet application timing requirements in the presence of faults. Section 5 presents the experimental results and discusses the key observations. Section 6 concludes this paper with future research direction.

2 System Definitions

The focus of the study is on SMP systems that are found in homogeneous multicore embedded processor systems and tightly coupled multiple-processor systems. In such systems, processing units have identical capabilities and characteristics, and interunit communication is often achieved via shared memories such that the communication cost can be assumed to be negligible. Each processor unit is DVS-capable and is equipped with an individual cache. Tasks are assumed to be preemptive, such that a processor may selectively suspend or reassign the current task and switch context to a new task according to some heuristic. It is assumed that a central scheduler, implemented on an independent processor, runs the proposed allocation and scheduling algorithms. It accepts as input the task set to be executed, processor voltage settings, and fault characteristics to produce an energy-efficient voltage schedule as output. This strategy to deploy a scheduler achieves high system performance but introduces low overhead. An example of such a system is the Spring Kernel system [25].

2.1 Architecture and Application Model

Consider a multiprocessor system consisting of n homogeneous processors $\mathrm{P} = \{P_1, P_2, \ldots, P_n\}$ and a task set Γ consisting of m independent periodic tasks $\{\tau_1, \tau_2, \ldots, \tau_m\}$ with hard real-time constraints. The timing characteristics of task τ_i are defined as a tuple $\tau_i = \{T_i, D_i, C_i\}$, where T_i is the period, D_i is the deadline, and C_i is the worst case execution time in cycles. The hyperperiod of the task set, denoted by T, is the lowest common multiple of all task periods $\{T_1, T_2, \ldots, T_m\}$. It is assumed that the voltage, and hence speed, of each processor can be scaled independently, and all tasks on a processor are executed at the same speed. Denoting the minimum and maximum speed supported by a processor as $S_{\min}$ and $S_{\max}$, respectively, speed S_i of processor P_i normalized with respect to $S_{\max}$ satisfies $0 \leq S_{\min} \leq S_i \leq (S_{\max} = 1.0)$. The worst-case execution time and utilization of task τ_i executing on P_i is given by C_i/S_i and $u_i = C_i/(T_i S_i)$, respectively. It is assumed that in the absence of faults, the task set scheduled on each processing unit is schedulable at $S_{\max}$.

2.2 Fault Model

Transient faults can be modeled using a Poisson distribution such that the probability of k faults in a time interval T is given by $\mathrm{e}^{-\lambda T}(\lambda T)^k/k!$, where λ is the fault arrival rate. In general, λ increases as supply voltages and frequency decrease [26]. However, for the current fault rates, fault occurrences are expected to remain highly infrequent in the foreseeable future, and the fault-free operation will continue to dominate [2], [27]. Therefore, it is a common practice to design a system to tolerate up to L faults in each task instance with an upper bound of up to K $(K > L)$ faults in the system during the hyperperiod T, where L is often set to one [1].

Fault tolerance comprises fault detection followed by fault recovery. In general, fault detection mechanisms are classified into four categories:

1. a fail-signal processor to notify other processors of an immediate fault occurrence,
2. watchdog processors for concurrent control flow checking,
3. signatures that can be used for detection of hardware and software faults, and
4. an acceptance test that checks the test results for hardware or software faults [15].

It is assumed that faults are detected as soon as they occur. Upon detecting faults, the system can recover via TMR, the reexecution of the affected task such as PB or backward recovery. Of these, TMR is energy intensive, with more than 100 percent energy overhead, while PB leads to an energy-inefficient offline schedule and requires complex and efficient online schemes for slack reclamation. In a backward recovery system, each task has a set of checkpoints at which computation correctness is verified. If no fault is detected, the system state is saved, and the computation continues onward; if one or more faults are detected, the task rolls back to the previously saved states to resume computation. In order to simplify the analysis, it is often assumed that the time overhead to save the system state, Cs, and the time overhead to retrieve the stored system state, Cr, are constant, and checkpointing intervals for a given task are equal. Zhang and Chakrabarty showed that under these assumptions, the optimal number of checkpoints for task τ_i that minimizes its worst-case response time is given by $X_i = \mathrm{Max}(\|\sqrt{LC_i/Cs} - 1\|, 0)$ [10]. The term $\|\sqrt{LC_i/Cs} - 1\|$ denotes the value from the pair $\{\lceil\sqrt{LC_i/Cs} - 1\rceil, \lfloor\sqrt{LC_i/Cs} - 1\rfloor\}$ that minimizes the worst-case response time of task τ_i. Therefore, the total fault-free execution time of an instance of task τ_i is the sum of its execution time and its checkpointing overhead, that is, $C_i + X_i Cs$, and the total worst case cost of an instance of task τ_i including L fault overhead is given by $C_i + X_i Cs + LC_i/(X_i + 1) + L(Cs + Cr)$.

2.3 Energy Model

DVS-capable processors support discrete voltage levels, and power-aware memories deploy power management strategies to support one or more low-power operation modes. It has been shown that the energy consumption of aggressive power-aware memories is much lower than the energy consumption of processors and, therefore, the DVS characteristics of systems closely follow the DVS characteristics of processors. In other words, the energy consumption characteristics of overall systems can be modeled based on the energy consumption characteristics of processors [28].

The power consumption of a processor running at the normalized speed S is given by a strictly increasing and convex function $g(S) = S^3$ [22], [36], [37]. Therefore, the energy consumed by a task τ_i during time interval $[t_1, t_2]$ is given by $E_i = \int_{t_1}^{t_2} g(S)dt$. For a periodic task τ_i executing at speed S, the total energy consumption during a hyperperiod T is given by $g(S)(C_i/S)\ (T/T_i)$. Let $u_k = C_k/(S_{\max} \times T_k) = C_k/T_k$ denote the utilization of task τ_k at the highest processor speed and let U_j denote the total utilization of processor P_j with m_j tasks running at speed $S_{\max}$. The energy consumption of processor P_j is given by

$$\begin{aligned} E_j &= \sum_{(1\le k\le mj)} g(S_j)\times (C_k/S_j)\times (T/T_k) \\ &= \big(g(S_j)/S_j\big)\times T\times \sum_{(1\le k\le mj)} u_k \\ &= \big(g(S_j)/S_j\big)\times T\times U_j. \end{aligned} \tag{2.1}$$

Therefore, the total energy consumption E_{tot} of an n-processor system is given by

$$\begin{aligned} E_{tot} &= \sum_{(1\le j\le n)} E_j = \sum_{(1\le j\le n)} \big(g(S_j)/S_j\big)\times T\times U_j \\ &= T\times \sum_{(1\le j\le n)} \big(g(S_j)/S_j\big)\times U_j. \end{aligned} \tag{2.2}$$

3 Energy Characterization of Fault-Tolerant Multiprocessor Systems

The energy consumption of a multiprocessor system depends significantly on the task-to-processor allocation strategies. These strategies can be broadly classified into two categories: global allocation and partitioning allocation [29]. Under global allocation, a task instance can execute on any processor, and when required, tasks can migrate to other processors. On the other hand, partitioning allocation assigns tasks to processors permanently, and migration among processors is prohibited. The global allocation strategy suffers from its large overhead due to task migration among processors and has an adverse impact on the feasibility performance of optimal uniprocessor scheduling algorithms such as RMA and EDF [29]. Therefore, RMA-based hard real-time systems usually use partitioning allocation due to its simplicity, ease of implementation, and good synergy with energy-efficient scheduling schemes for uniprocessor systems.

Besides allocation, task scheduling also has a significant impact on the energy and feasibility characteristics of a multiprocessor system. A task schedule is guaranteed to be feasible if the total utilization of tasks assigned to each processor is less than or equal to its utilization bound, $Ub \le 1$, and the total utilization of the system, U_{tot}, is less than or equal to the system utilization bound, $n \times Ub$. It has been shown that under the assumption that all tasks on processor P_j run at a common speed, setting the normalized speed of processor P_j equal to its utilization normalized with respect to Ub_j, that is, $S_j = U_j/Ub_j$, achieves an energy-optimum schedule [22], [24]. Therefore, setting $S_j = U_j/Ub_j$ and $g(S) = S^3$, (2.2) becomes

$$E_{tot} = T\times \sum_{(1\le j\le n)} \big(g(S_j)/S_j\big)\times U_j = T\times \sum_{(1\le j\le n)} U_j^3/Ub_j^2. \tag{3.1}$$

Since T is constant for a given task set, the energy optimization problem is equivalent to minimizing the objective function $E_{tot}/T = \sum_{(1\le j\le n)} U_j^3/Ub_j^2$ given by (3.1) subject to

$$\sum_{(1\le j\le n)} U_j = U_{tot}, \tag{3.2}$$

$$U_j \le Ub_j \le 1. \tag{3.3}$$

The utilization bound of fixed-priority offline scheduling algorithms, commonly used in hard real-time systems due to their low overhead and predictability, is variable and depends upon task characteristics. For example, the conservative utilization bound of RMA is given by $Ub_j = m_j\,(2^{1/mj} - 1) \ge 0.69$, where m_j is the number of tasks assigned to processor P_j [30]. Substituting it in (3.1) gives the energy optimization problem:

$$\text{Minimize } E_{tot}/T = \sum_{(1\le j\le n)} U_j^3/m_j^2(2^{1/mj} - 1)^2 \text{ subject to}$$

$$\sum_{(1\le j\le n)} U_j = U_{tot}, \tag{3.4}$$

$$U_j \le m_j(2^{1/mj} - 1), \tag{3.5}$$

$$\sum_{(1\le j\le n)} m_j = m. \tag{3.6}$$

For a given task set, the problem of finding an energy-optimum feasible schedule is essentially a constrained optimization problem. This problem is well known to be NP-hard, and an optimal solution can only be found via exhaustive search, which is computationally intractable. Further, in a fault-prone system, the workload of a processor changes every time a fault occurs. Therefore, the current optimal solution may become suboptimal, and a new task allocation and scheduling needs to be performed during runtime based on fault characteristics.

On other hand, a more elegant solution exists for the energy-optimization problem given by the objective function in (3.1) for scheduling schemes with constant utilization bounds. Note that for some constants $a_1, a_2, \ldots, a_n > 0$ and $s < t$, the power mean inequality

$$\big((a_1^s + a_2^s + \ldots + a_n^s)/n\big)^{1/s} \le \big((a_1^t + a_2^t + \ldots + a_n^t)/n\big)^{1/t} \tag{3.7}$$

holds, where the equality holds iff $a_1 = a_2 = \ldots = a_n$ [31]. Setting $a_j = U_j$, $s = 1$, and $t = 3$, (3.7) becomes

$$(U_1 + U_2 + \ldots + U_n)^3 \times n^{-2} \le U_1^3 + U_2^3 + \ldots + U_n^3. \tag{3.8}$$

Since $\sum_{(1\le j\le n)} U_j = U_{tot}$ and n is constant, the right-hand side of (3.8) is minimized when $U_1 = U_2 = \ldots = U_n = U_{tot}/n$ and the equality holds. Therefore, the energy consumption of such a multiprocessor system is minimized to $(T \times U_{tot}^3)/(n^2 \times Ub^2)$ when

$$\begin{cases} Ub_i = Ub_j = Ub \\ U_i = U_{tot}/n \end{cases} \quad 1 \le i \ne j \le n \tag{3.9}$$

is satisfied. In other words, for a given hard real-time system with known T, U_{tot}, n and a scheduling strategy with fixed utilization bound Ub, the energy consumption of the system is optimized when the workload is balanced uniformly among all processors. A similar conclusion was reached by Aydin and Yang in [22] for the EDF scheduling algorithm. The EDF algorithm is based on dynamic task priorities and has a constant utilization bound of $Ub = 1.0$.

Due to the lack of fixed-priority scheduling schemes with constant utilization bound for hard real-time systems, an alternate approach is proposed, which operates in two phases. It is based on two critical observations from the design of practical fault-tolerant real-time systems. In phase 1, the asymptotic bound of the RMA scheduling strategy, given by $m_j(2^{1/mj} - 1) \to 0.69$ as $m_j \to \infty$, is used

as the utilization bound for allocating tasks to processors. Since the asymptotic bound is a constant value, a novel partitioning scheme is proposed to uniformly distribute the system workload among all processors in order to achieve energy efficiency. In this phase, the task workload does not include the associated fault overhead, since according to the current fault rates, a fault-free execution is expected to be dominant in the foreseeable future. In phase 2, the tasks on individual processors are scheduled using RMA after accounting for the fault overhead, and an efficient exact characterization of the RMA-based scheme is proposed to test the feasibility of the schedule. In practice, RMA performs much better than the conservative estimate of the asymptotic bound, with processor utilization often reaching as high as 0.90 [23]. This allows for task set schedules to tolerate the fault overhead in spite of the use of the asymptotic bound in phase 1.

4 Joint Optimization of Energy, Reliability, and Feasibility in Multiprocessor Systems

The main contribution of this work is a novel task partitioning heuristics with proven load-balancing properties under varying workload conditions and its integration with an optimistic fault-tolerant scheme to achieve energy efficiency in a reliable hard real-time multiprocessor system. Since a fault-free operation is expected to remain dominant in the near future and the probability of the worst case faults is low, the proposed task allocation scheme does not account for the cost of fault tolerance. Combining it with an efficient scheduling scheme, which jointly optimizes feasibility performance and DVS policies and dynamically adapts to fault characteristics, significantly reduces the energy consumption without compromising the feasibility or reliability of the system. The proposed scheduling scheme assumes the worst case fault overhead and improves its feasibility performance using ECRMA.

4.1 Energy-Efficient Task Allocation Based on Optimistic Fault Tolerance

Task allocation heuristics can be classified into two categories based on their objectives: 1) minimizing the number of processors used to feasibly schedule a given task set and 2) minimizing the finish time of a task set on a given number of processors via workload balancing. Resource-constrained task allocation in multiprocessor systems falls in the same class as the well-known bin packing problems. Several classic heuristics, including Next Fit (NF), First Fit (FF), Best Fit (BF), and Worst Fit (WF), have been adapted for task-to-processor allocation [33]. Typically, these allocation heuristics suffer from poor feasibility performance due to their unaccounted workload properties such as individual deadlines of tasks and various task dependencies (data, resource, temporal, etc.). It has been shown that the performance of these heuristics could be improved by sorting tasks in nonincreasing order of utilization before assigning them to processors [34], [35]. NF Decreasing (NFD), FF Decreasing (FFD), BF Decreasing (BFD), and WF Decreasing (WFD) belong to this new class of heuristics and are counterparts of NF, FF, BF, and WF, respectively [33], [34]. The goal of FFD, NFD, and BFD is to minimize the number of processors, and they often result in a highly unbalanced workload distribution. On other hand, the goal of WFD is to balance the workload in order to minimize the system response time [22], [24], [34]. However, the load-balancing property of WFD starts to degrade under a light-workload condition since 1) it is assumed that not all processors are open for task assignment at the beginning of allocation process and 2) processors are selected in the order of maximum remaining capacity. For example, for a task set whose workload can be accommodated on a single processor, WFD will yield a highly unbalanced task partition in which all tasks are assigned to the first processor and all other processors are idling.

Therefore, a new task partitioning scheme based on the modified WFD heuristic, referred to as MWFD, is proposed for designing energy-efficient multiprocessor systems. MWFD features a proven load-balancing property and hence achieves energy efficiency under varying workload conditions. The feasibility performance of MWFD is comparable to FFD, as shown by the simulation results in Section 5. MWFD has two fundamental differences from WFD: 1) all n processors are open at the beginning, and 2) instead of allocating the next task to the processor with maximum remaining capacity, MWFD selects the processor with minimum workload and breaks ties in favor of the processor with a smaller index. As shown next, MWFD achieves optimal load balancing independent of the system workload.

Proposition. *For a given periodic task set* $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_m\}$ *with corresponding nonincreasing utilizations* $u_1 \geq u_2 \geq \ldots \geq u_m$ *and an* n-processor *system, the MWFD allocation algorithm produces an optimally balanced workload partition among the* n *processors.*

Proof. For an aperiodic independent task set $\Gamma' = \{\tau'_1, \tau'_2, \ldots, \tau'_m\}$, sorting tasks according to their workload, defined as the number of processor cycles required for task execution, and sequentially assigning them to n processors, produces an optimally balanced partition [32]. Denoting the minimum possible finish time (that is, response time) of all possible partitions by ω'_0 and the finish time of a specific partition by ω', a task allocation scheme that yields ω'/ω'_0 relatively closest to one balances the workload and minimizes the finish time of the task set Γ'. It was shown that by ordering the first $(h \leq m)$ longest tasks in the task set from the largest to the smallest task length (task length is equivalent to workload for aperiodic tasks) with no order constraints on the remaining $(m - h)$ tasks, the bound for ω'/ω'_0 is given by

$$\omega'/\omega'_0 \leq 1 + \frac{(1 - 1/n)}{(1 + \lfloor h/n \rfloor)}. \tag{4.1}$$

This bound is optimal for $h \equiv 0 (\mathrm{mod}\ n)$. Since the right-hand-side expression in (4.1) is a strictly decreasing function of h, the bound on ω'/ω'_0 approaches its minimum value $1 + (1 - 1/n)/(1 + \lfloor m/n \rfloor)$ when all m tasks in the task set are sorted according to their workloads. In other words, allocating aperiodic tasks sequentially in the order

of workload results in an optimally balanced partition such that the bound $1 + (1 - 1/n)/(1 + \lfloor m/n \rfloor)$ cannot be replaced by any smaller function of the same variables.

This conclusion can be extended to periodic tasks as follows: Consider a periodic task set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_m\}$ with timing parameter $\{T_i, D_i, C_i\}$ for $1 \leq i \leq m$. The total workload contributed by a task τ_i during hyperperiod T is given by $C_i \times (T/T_i) = T \times (C_i/T_i) = T \times u_i$. In other words, for a periodic task set, sorting tasks in the order of utilization is equivalent to sorting them according to their workloads. Since MWFD assigns tasks to processors in the order of nonincreasing utilization, the bound given by (4.1) holds and $\omega/\omega_0 \leq 1 + (1 - 1/n)/(1 + \lfloor m/n \rfloor)$ when all m tasks in the periodic task set are sorted according to their utilizations. This completes the proof for the optimal load-balancing property of the proposed MWFD heuristic. □

4.2 Runtime Adaptive Fixed-Priority Scheduling Based on the Exact Characterization of RMA

After allocating all tasks to processors using MWFD, an optimal fixed-priority offline scheduling scheme for uniprocessor systems, RMA, is used to independently schedule the task subset on each processor by assigning higher priorities to tasks with shorter periods [30]. The worst case behavior of RMA occurs when all tasks in a task set are instantiated simultaneously and are ready for execution immediately after their initiation. This time instant is called the critical instant, and it has been shown that a schedule of independent periodic tasks at the critical instant is feasible if the first instance of each task is schedulable [23]. The asymptotic bound of RMA is given by $m_j(2^{1/m_j} - 1) \rightarrow 0.69$ as $m_j \rightarrow \infty$, where m_i denotes the number of tasks allocated to processor P_i.

It is well known that the asymptotic bound of RMA is a conservative estimate, and in practice, RMA can achieve a utilization as high as 0.90. Lehoczky et al. proposed an efficient systematic scheme based on ECRMA to derive both necessary and sufficient conditions for the feasibility analysis of a task schedule [23]. It was proven that a set of periodic tasks is schedulable if and only if the cumulative demand of each task on a processor is equal to or less than the current available processor time. Specifically, let $W_i(t) = \sum_{(1 \leq j \leq i)} C_j \lceil t/T_j \rceil$ denote the cumulative demand of task τ_i on a processor over $[0, t]$, assuming 0 as the critical instant. The necessary and sufficient condition for a set of periodic tasks to be schedulable is given as

$$\mathrm{Max}_{(1 \leq i \leq n)} \{ \mathrm{Min}_{(0 < t \leq Ti)} [W_i(t)/t] \} \leq 1. \quad (4.2)$$

It was shown that only a finite number of time instants, called scheduling points, need to be checked for feasibility analysis since the normalized demand of task τ_i on a processor, $W_i(t)/t$, is strictly decreasing except at scheduling points. Scheduling points SP_i associated with task τ_i are defined as multiples of T_g for $T_g \leq T_i$, that is

$$SP_i = \{ hT_g | g = 1, 2, \ldots, i; h = 1, 2, \ldots, \lfloor T_i/T_g \rfloor \}. \quad (4.3)$$

Therefore, (4.2) can be rewritten as

$$\mathrm{Max}_{(1 \leq i \leq n)} \{ \mathrm{Min}_{(t \in SPi)} [W_i(t)/t] \} \leq 1. \quad (4.4)$$

This technique offers two significant advantages over the asymptotic feasibility analysis technique: 1) since ECRMA is based on time-demand analysis, it provides a much higher utilization bound for a given task set, and 2) it needs to be evaluated only at a finite and often small number of scheduling points and hence is more conducive to efficient runtime extensions for online fault tolerance and DVS policy reevaluations. Since tasks allocated using MWFD and feasibly scheduled based on the asymptotic bound of RMA may become infeasible in the presence of faults and voltage scaling, ECRMA could be used to efficiently verify the feasibility of an offline schedule in the presence of faults and voltage scaling and to adapt DVS policies online to variations in task execution time and fault occurrences.

Two offline techniques, application-level voltage scaling (A-DVS) and task-level voltage scaling (T-DVS), are proposed for scheduling the task subset allocated to a processor. A-DVS selects the lowest common processor frequency for all tasks in the subset at which each task meets its deadline, while T-DVS selects individual frequencies for each task in the task subset in order to further reduce the energy consumption at the cost of increased complexity in the scheduler. Both techniques systematically integrate fault tolerance and DVS policy evaluations to produce an energy-efficient schedule as the first feasible solution and are less complex when compared to the previous techniques [10].

A-DVS is suitable for systems in which frequent scaling of processor voltage, and hence frequency, is inefficient, such as the systems where 1) processor voltage scaling is slow and/or consumes significant energy, 2) task sets are updated frequently, and hence, the overhead due to scheduler complexity is significant, and 3) system utilization is very low, and hence, the extra cost of T-DVS to compute individual task frequencies is unjustified. Without loss of generality, it is assumed that in the absence of faults, the task subset allocated to a processor is schedulable at the lowest frequency level. The assumption can be easily modified to accommodate other frequency levels. A-DVS starts by calculating the scheduling points for each task and then iteratively performs feasibility analysis using ECRMA to select the optimal DVS strategy while tolerating L faults in each task instance. Every time a task fails to meet its deadline, the common frequency is increased by one level, and the feasibility test is repeated. This process continues until all tasks are scheduled successfully or the highest frequency level is reached and the task set is found to be infeasible. The T-DVS algorithm is similar to A-DVS except that whenever a task is found to be unschedulable, the scheme iteratively increases the frequencies of equal and higher priority tasks with the lowest frequency level until the current task becomes schedulable or the highest frequency level for all tasks is reached and the task set is found to be infeasible.

The offline scheduling schemes assume that tasks exhibit the worst case execution times and the worst case fault occurrences. However, the runtime behavior of the task execution time and occurrences of faults can vary significantly [36], and the slack generated in the runtime due to less than expected number of faults and due to better than expected task execution time could be used to dynamically

scale down the processor speed. Hence, the online reevaluation of DVS policies that adapt to the runtime characteristics of tasks and faults can save significant energy. The proposed algorithms, A-DVS and T-DVS, provide efficient mechanisms to exploit the slack time generated during the runtime to further slow down the processor speed and save energy without compromising the feasibility of the offline schedules. The runtime reevaluation of DVS policies is performed at the end of execution of each task. At the application level, online DVS policies determine whether the generated slack time is sufficient to scale down the processor speed by comparing the amount of time needed for the schedule of the remaining tasks to be feasible at the next lower frequency level. The time overflow at each successive frequency level can be precomputed during the offline feasibility analysis and stored in the system memory. The dynamic reevaluation of DVS strategies at the task level is more straightforward since offline T-DVS derives the optimum combination of frequency allocation to tasks, and during runtime, the frequencies of individual tasks can be scaled independently. The detailed description and pseudocode of the A-DVS and T-DVS schemes can be found in [11].

4.3 Optimistic and Adaptive Fault-Tolerant Algorithm for Hard Real-Time Systems

The proposed MWFD-A/TDVS scheme operates in two phases. In phase 1, MWFD determines the task-to-processor allocation and binding under the assumption that all processors run at their highest speed and a fault does not occur. The goal of allocation is to achieve a uniformly balanced workload in order to minimize energy consumption under the optimistic assumption of no or a small number of faults. In phase 2, ECRMA-based A/T-DVS jointly optimizes the fault tolerance and DVS policies. The goal of the scheduling scheme is to derive an energy-efficient offline schedule and dynamically adapt to the runtime behavior in order to maximize energy savings. Overall, the proposed scheme optimistically aims to minimize system energy consumption under no or less than expected fault occurrences while meeting application timing requirements under the worst case behavior.

Figs. 1 and 2 show the operation of the optimistic fault-tolerant MWFD (OFT_MWFD) algorithm and the schedulability checking algorithm (SCA), respectively. The notations used in the two algorithms are defined as follows:

- min: index of the processor with minimum workload.
- m_{min}: the number of tasks on the processor with index min.
- W_{min}: workload (utilization) without fault recovery overhead on the processor with index min.
- U_{min}: total utilization including the fault recovery overhead on the processor with index min.
- *Schedulable*: flag to indicate if a task to be allocated is schedulable on the target processor.
- *Demand*: accumulative time demand of a task on processor.
- u_i: the updated utilization of task τ_i with checkpointing overhead but without fault recovery overhead.
- $ufault_i$: the updated utilization of task τ_i including the checkpointing and fault recovery overhead.

Procedure OFT_MWFD (Γ, m, n, L, Cs, Cr)

1. min=0, W_{min}=0, U_{min}=0, i=1, *Schedulable* =0;
   //m is the number of tasks and n is number of processors
   //Update utilizations to include checkpointing overhead
2. **for** τ_i, $1 \le i \le m$ **do**
3. $X_i = \max\{|| \sqrt{L \times C_i / Cs} - 1||, 0\}$;
4. $u_i = (C_i + X_i Cs)/T_i$;
5. **end for**
6. Sort τ_i, $1 \le i \le m$, in non-increasing order of u_i;
   //Update utilizations to include fault recovery overhead
   //ufault denotes the array of $ufault_i$
7. **for** τ_i, $1 \le i \le m$ **do**
8. $ufault_i = u_i + L \times C_i/((X_i+1) \times T_i) + L \times (Cs+Cr)/T_i$;
9. **end for**
   //Optimistically allocate m tasks to n processors
10. **for** τ_i, $1 \le i \le m$ **do**
11. Find processor P_{min} with $W_{min} = \text{MIN}_{(1 \le j \le n)} W_j$;
12. **if** $0.69 \ge (U_{min} + ufault_i)$ **then**
13. $U_{min} = U_{min} + ufault_i$;
14. $W_{min} = W_{min} + u_i$;
15. **else**
16. *Schedulable* = SCA (min, $ufault$);
17. **if** *Schedulable* = 1 **then**
18. $U_{min} = U_{min} + ufault_i$;
19. $W_{min} = W_{min} + u_i$;
20. **else print** "Infeasible Partition"; **return**; **end if;**
21. **end if**
22. **end for**
    //Voltage scaling and feasibility analysis
23. **for** P_i, $1 \le i \le n$ **do**
24. Arrange allocated task subset according to RMA;
25. **call** A-DVS;
    //Or call T-DVS, depending on system characteristics
26. **end for**

Fig. 1. OFT_MWFD.

- $ufault$: the array of $ufault_i$.
- SP_i: the set of scheduling points of task τ_i.

The inputs to OFT_MWFD are the task set (Γ), the number of tasks in the task set (m), the number of processors in the system (n), the maximum number of faults each task instance should tolerate (L), the checkpoint saving cost (Cs), and the checkpoint recovery cost (Cr). Lines 2-5 of the OFT_MWFD algorithm update task utilizations to include the checkpoint saving overhead, that is, $u_i = (C_i + X_i Cs)/T_i$ for $1 \le i \le m$. Line 6 sorts the updated tasks in the order of nonincreasing utilization. Lines 7-9 update task utilizations to include the fault recovery overhead. All processors are considered open during task allocation. Lines 10-22 optimistically allocate m tasks to n processors to achieve energy efficiency. The processor P_{min} with minimum workload W_{min} is identified and selected for task assignment, and the asymptotic utilization bound of RMA is used to check if the

```
Procedure SCA (min, ufault)
1.  Schedulable=0; Demand=0; i=p=q=1;
    //m_min is the number of tasks on processor min
2.  for τ_i, 1≤i≤m_min do
      //Derive SP_i, the set of scheduling points of task τ_i
3.    SP_i = {h×T_g | g = 1,2,...,i; h = 1,2,...,⌊T_i/T_g⌋}
      //sp_iq in SP_i is the q^th scheduling point of task τ_i
4.    for sp_iq ∈ SP_i do
5.      Demand =0;
6.      for τ_p, 1≤p≤i do
7.        Demand = Demand +ufault_p×T_p×⌈s_iq/T_p⌉;
8.      end for
9.      if Demand ≤sp_iq then
10.       Schedulable = 1; break;
11.     else Schedulable = 0; q++; end if
12.   end for
13.   if Schedulable = 0 then return (Schedulable);
14.   end if
15. end for
16. return Schedulable;
```

Fig. 2. SCA.

current task is schedulable on the processor. If schedulable, it is assigned to processor P_{min}, and workload W_{min} and utilization U_{min} are updated by adding u_i and $ufault_i$, respectively, as shown in lines 11-14. Otherwise, the algorithm performs ECRMA-based feasibility analysis by calling the SCA subroutine in line 16 and updates W_{min} and U_{min} in lines 18 and 19, respectively. Lines 10-22 repeat until a feasible partition is found for the task set or the flag *Schedulable* is set to zero by the SCA subroutine and the task set cannot be feasibly partitioned using the algorithm. Note that u_i, the utilization of task τ_i without fault recovery overhead, is used to update W_{min} for the identification of the processor with minimum workload, while $ufault_i$, the utilization with fault recovery overhead, is used to update U_{min} for feasibility analysis.

OFT_MWFD first uses the asymptotic utilization bound of RMA to check the schedulability of a task to be allocated. If the task cannot be feasibly scheduled on the processor P_{min}, the ECRMA-based approach is launched for enhanced feasibility analysis. This two-step strategy reduces the average (practical) time complexity of the algorithm, especially when the system workload is light, since the ECRMA-based approach is much more complex than the asymptotic-utilization-bound-based feasibility analysis.

The SCA shown in Fig. 2 accepts as inputs the index of the processor with minimum workload (min) and the array of $ufault_i$ $(ufault)$. SCA iteratively checks the schedulability of the tasks on the processor P_{min}. The algorithm starts by initializing both the flag *Schedulable* and the flag *Demand* to zero in line 1. Line 3 derives scheduling points of the ith task on the processor P_{min}, which is denoted by SP_i. The qth scheduling point of the task τ_i is denoted by sp_{iq}. Lines 4-12 calculate the accumulative time demand of the task on processor, denoted by the flag *Demand*, check the schedulability of the task at its scheduling points, and set the flag *Schedulable* accordingly. If the current task cannot be feasibly scheduled at any of its scheduling points, the flag *Schedulable* is set to zero and returned, as shown in line 13. Else, SCA moves on to the next task for feasibility analysis. This process iterates for all tasks on the processor P_{min} to verify their schedulability.

DVS policy evaluation is applied to task subsets that are feasibly assigned to processors, as is shown in lines 23-26 in Fig. 1. The voltage scaling algorithm, A-DVS or T-DVS, is called depending upon system characteristics such as context switching overhead and task utilizations. For A-DVS, where all tasks in a task subset run at the same processor speed, the A-DVS algorithm examines various processor voltage levels from low to high to determine the lowest voltage level that satisfies the timing constraints of tasks subject to L faults in each task instance. For T-DVS, where each task in a task subset is assigned an individual voltage level, the T-DVS algorithm derives the greedily optimum combination of speed assignments for each task such that all timing constraints are satisfied subject to L faults in each task instance. The T-DVS algorithm is similar to the A-DVS algorithm except that whenever a task is found to be unschedulable, T-DVS repeatedly selects one task from among tasks of equal and higher priorities and scales its voltage level up by one level until the current task becomes schedulable or the highest voltage level for all tasks is reached and the task set is found to be infeasible. A detailed description of A-DVS and T-DVS can be found in [11]. The next section presents the results of MWFD simulation studies.

5 Experimental Results

Extensive experiments were carried out over a simulated SMP system to validate the proposed schemes for energy efficiency and feasibility performance. It is assumed that the system consists of processors with identical characteristics, all processors support continuous voltage scaling, and the normalized speed of each processor ranges from zero to one. Interprocessor and intraprocessor communication is zero since task sets are assumed to comprise independent tasks, and the time and energy overhead of memory accesses is accounted for in task execution times and the processor energy model, respectively. The number of processors in a system was varied from two processors to eight processors. Simulation experiments were performed over task sets with widely varying temporal characteristics. Tasks within a task set were generated randomly by following the approaches suggested in [34], [39], and [40], and a total of 10,000 different configurations were used to account for the statistical anomalies. Task periods were generated with uniform probability such that each task has the same probability of being short (1-10 ms), medium (10-100 ms), and long (100-1,000 ms). Task utilizations were generated randomly based on the Beta distribution of probability and were limited to less than ln2, the asymptotic bound of RMA. The standard deviation of utilizations, σ, was limited to a maximum value $\sigma_{\max}$, which is a function of the mean of task utilizations. The total number of faults tolerated by a system during a

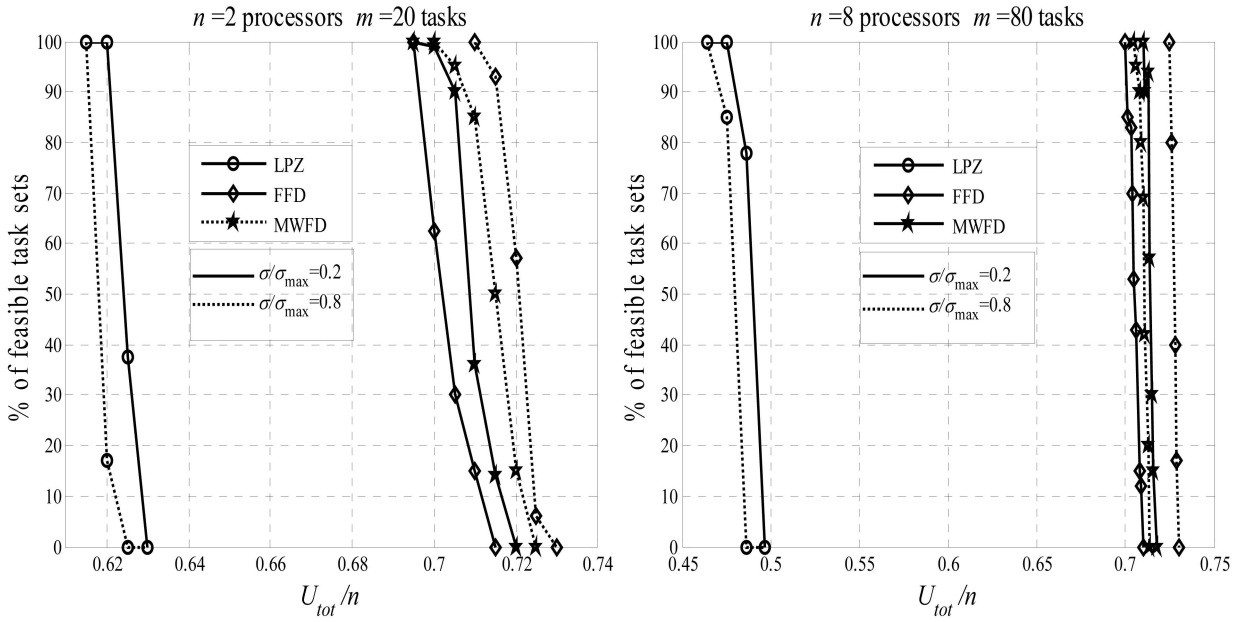

Fig. 3. Performance of the LPZ, FFD, and MWFD allocation schemes when used in conjunction with RMA-asymptotic-bound-based scheduling scheme.

hyperperiod T is limited to K, and faults are randomly distributed among processors under the assumption that each task instance can tolerate at most $L = 1$ fault. Note that the number of faults each task instance is designed to tolerate is independent of the proposed task allocation and scheduling schemes and hence can be any number for $L < K$. Results are reported for exponentially distributed fault occurrences and were obtained by averaging over all 10,000 simulation runs.

5.1 Feasibility Performance of MWFD

To validate the claim that the proposed allocation scheme, MWFD, does not degrade the schedulability of a task set, its feasibility performance was compared with that of other state-of-art schemes. Feasibility performance is defined as the percentage of task sets, with varying workloads, that can be successfully scheduled on a given multiprocessor system. Since, to the best of our knowledge, this is the first work on energy-efficient fault-tolerant real-time multiprocessor systems [13], [14], [38], classic FFD, WFD, and the theoretical lower utilization bound for multiprocessor systems derived by Lopez et al. [34] were used for comparison. This lower utilization bound, hereafter called LPZ, is given by $(n\beta + 1)(2^{1/(\beta+1)} - 1)$, where n is the number of processors in a system, $\beta = \lfloor 1/\log_2(\alpha + 1) \rfloor$, and α is the maximum utilization among all tasks in a task set. The classic FFD has been chosen for comparison because it is one of the simplest heuristics and exhibits identical or better feasibility performance compared to BFD, NFD, and WFD when RMA is used for task scheduling [24], [34], [41]. The classic WFD has been chosen for comparison because it tends to outperform other heuristics in energy consumption due to its load-balancing property [24], [34]. Since the feasibility performance of WFD was found to be very similar to that of FFD, it has been omitted from the presented results on feasibility performance in favor of space and reduced cluttering of graphs. As a result, FFD has been chosen for comparison in feasibility performance, as shown in this section, and FFD and WFD have been chosen for comparison in energy, as shown in the next section.

Postallocation RMA was used for scheduling tasks on individual processors, and feasibility performance results were compiled for both the asymptotic and exact characterization of RMA-based bounds. Fig. 3 shows that under fault-free conditions and asymptotic-bound-based scheduling, the feasibility performance of MWFD and FFD is comparable and, as expected, better than the theoretical lower bound of the LPZ scheme. LPZ assumes that the utilization of all tasks equals α, the maximum utilization of all task utilizations in a task set. This renders the theoretical bound extremely conservative, especially if α is large. For example, for task sets with $\sigma/\sigma_{\max} = 0.2$ and with average processor utilizations of 0.69 and 0.72 in a dual-processor system, 100 percent and 0 percent of task sets, respectively, were found to be feasible according to both FFD and MWFD schemes, while no task sets were found to be feasible according to the LPZ scheme. The impact of allocation schemes on the schedulability of task sets can also be represented in terms of the distances between their graphs. For example, 70 percent of task sets with $\sigma/\sigma_{\max} = 0.2$ and with average processor utilizations of 0.7 and 0.71 were found to be feasible using FFD and MWFD schemes, respectively.

Fig. 3 also shows that the performance of MWFD degrades as the ratio $\sigma/\sigma_{\max}$ decreases. Lines in solid represent the feasible task sets with a lower standard variation $(\sigma/\sigma_{\max} = 0.2)$ in task utilizations, while the lines in dots represent the feasible task sets with a higher $(\sigma/\sigma_{\max} = 0.8)$ standard variation in task utilizations. The asymptotic RMA scheduling capacity of an n-processor system depends on the number of tasks allocated on each processor. Let $c_i > 0$ denotes the scheduling capacity of processor P_i $(1 \le i \le n)$. Therefore, the total scheduling

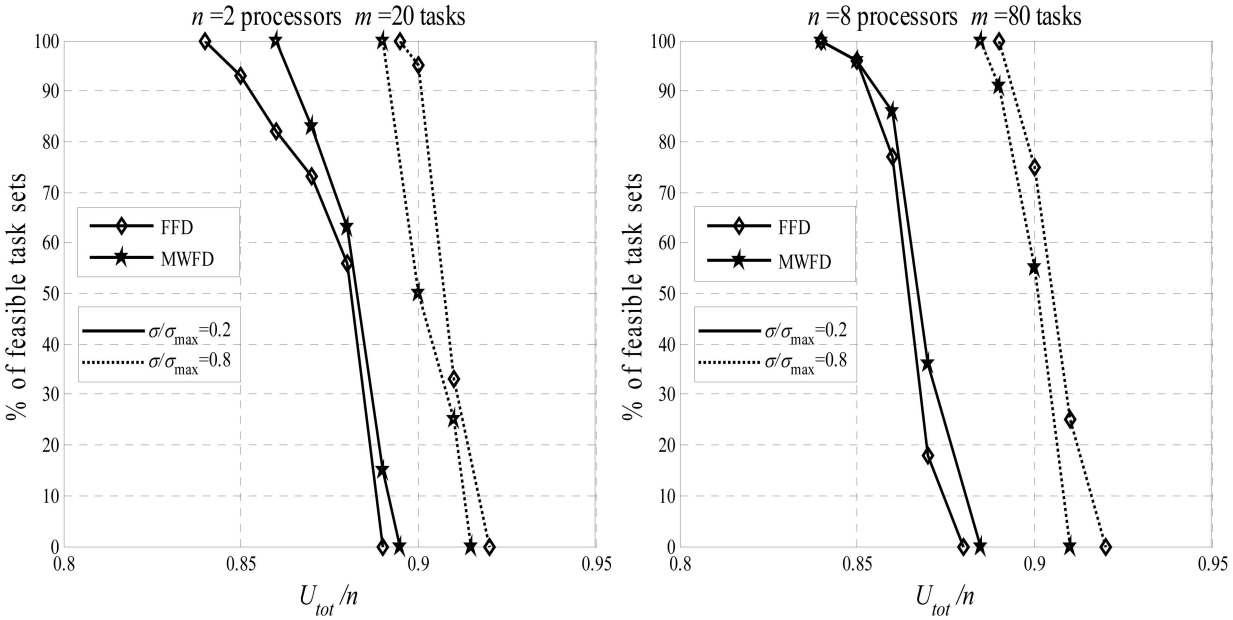

Fig. 4. Performance of the FFD and MWFD allocation schemes when used in conjunction with the ECRMA-based scheduling scheme.

capacity of an n-processor system is given by $C = \sum_{(1 \le i \le n)} c_i$. According to power mean inequality (3.7), $C/n = \sum_{(1 \le i \le n)} c_i/n = (c_1 + c_2 + \ldots + c_n)/n \le ((c_1^t + c_2^t + \ldots + c_n^t)/n)^{1/t}$, where n is the number of processors, and $t > 1$. The left-hand-side expression is minimized for $c_1 = c_2 = \ldots = c_n$, that is, the system capacity decreases with the decreasing variation in numbers of tasks assigned to individual processors. For a smaller $\sigma/\sigma_{\max}$, that is, a smaller σ since $\sigma_{\max}$ is a constant, balancing the workload leads to balancing the number of tasks among processors, which in turn leads to reduced scheduling capacity. On other hand, for a large $\sigma/\sigma_{\max}$, FFD consistently outperforms MWFD since the goal of FFD is to maximize the number of feasibly scheduled tasks at the cost of an unbalanced (skewed) schedule [22], [24], [34], [36], while the goal of MWFD is to balance the workload among processors. In other words, for the asymptotic-bound-based approach, the scheduling capacity of FFD improves with the increasing variation in task utilizations.

Similar trends were observed for scheduling based on ECRMA, as shown in Fig. 4. The scheduling capacity of ECRMA depends on the number and characteristics of tasks and can be as high as 0.90. The feasibility performance of FFD and MWFD was found to be almost identical, such that the intergraph distance (that is, the difference in average processor load of feasible task sets) was often less than 0.5 percent. Finally, from Figs. 3 and 4, it can be deduced that MWFD is more tolerant to variations in task utilizations, that is, its performance does not significantly change with the variations in $\sigma/\sigma_{\max}$. This can be particularly useful in hard real-time systems where deterministic performance often trumps the other performance metrics.

5.2 Energy Consumption Characteristics of MWFD

The energy consumption of a feasible task set is calculated according to (3.1), which computes the energy consumed by a processor as a function of the processor speed normalized with respect to its maximum speed. Equation (3.9) shows that the energy consumption of an n-processor system depends on the load distribution and scheduling capacity of the allocation and scheduling schemes applied to the system. It is shown that the system energy consumption is minimized when the workload is balanced among processors and each processor has a large scheduling capacity. Fig. 5 shows the energy consumption of the FFD, WFD, and MWFD allocation schemes when used in conjunction with the RMA-asymptotic-bound-based scheduling scheme. It is assumed that all executions are fault free and $\sigma/\sigma_{\max} = 0.2$. Fig. 5a shows that FFD and WFD incur comparable energy consumption except that WFD performs slightly better under medium-workload conditions. For example, in a dual-processor system WFD consistently outdoes FFD when the average processor utilization of task sets is in the range of 0.35-0.55. MWFD incurs up to 70 percent less energy consumption under a light or medium system workload and is comparable to FFD and WFD under a heavy workload $(U_{tot/n} > 0.6)$. For example, for task sets with average processor utilizations of 0.3 and 0.5, FFD, WFD, and MWFD consume (580, 580, 150) and (830, 810, 690) energy units, respectively.

Fig. 6a shows that replacing the asymptotic RMA with ECRMA demonstrates similar trends in energy consumption, but a more careful analysis reveals several other important facts. As shown in Figs. 5 and 6, the operation of a multiprocessor system can be categorized into three distinct workload regions—light, medium, and saturation—according to the average processor utilization. Under a light-workload condition, FFD and WFD incur similar energy consumption, while MWFD incurs significantly less energy consumption. Under a medium-workload condition, MWFD performs better than WFD, which in turn performs

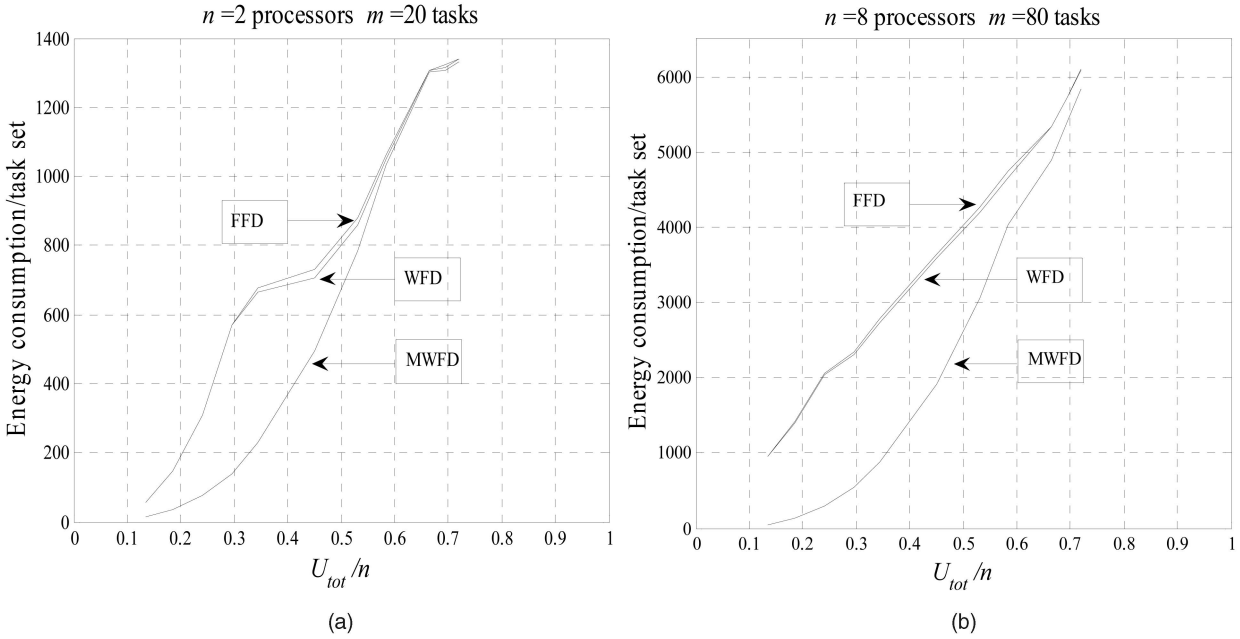

Fig. 5. Energy consumption of the FFD, WFD, and MWFD allocation schemes when used in conjunction with RMA-asymptotic-bound-based scheduling and the A-DVS algorithm $(\sigma/\sigma_{\max} = 0.2)$.

better than FFD, in energy consumption. In the saturation region, FFD, WFD, and MWFD incur comparable energy consumption regardless of the scheduling schemes applied to the system.

Further investigation reveals that although similar in trend, the actual energy consumed by FFD, WFD, and MWFD varies significantly from the asymptotic RMA to ECRMA. According to (3.1), the energy consumption of a processor is given by $E_j = U_j^3/Ub_j^2$, where U_j denotes the total utilization of all tasks assigned to processor P_j and depends upon the allocation schemes, and U_{bj} represents the upper utilization bound of processor P_j, which depends on the scheduling schemes and associated feasibility criteria. Therefore, under light-workload $(U_j^3 \ll U_{bj}^2)$ conditions, the ECRMA with higher scheduling capacity compared to the asymptotic utilization bound consumes significantly less energy for all the three allocation schemes. For example, for task sets with an average processor utilization of 0.30, the energy consumption of (FFD, WFD, MWFD) with the asymptotic utilization bound is (569, 569, 150) units, while the energy consumption of (FFD, WFD, MWFD) with ECRMA is (400, 400, 100) units,

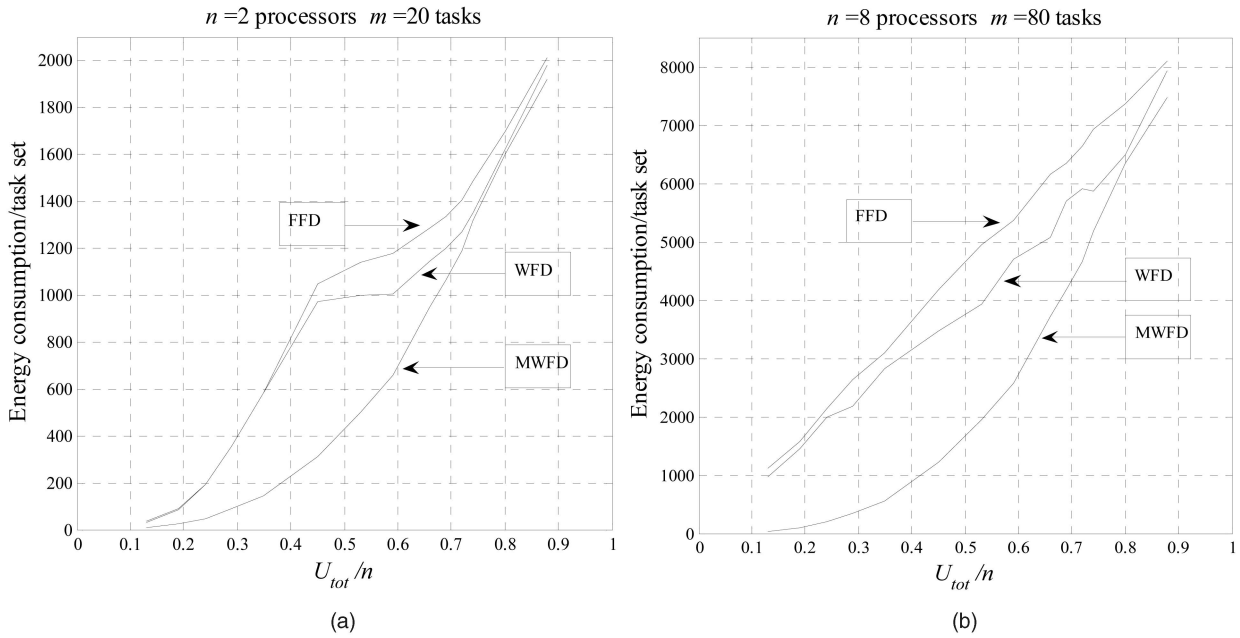

Fig. 6. Energy consumption of the FFD, WFD, and MWFD allocation schemes when used in conjunction with ECRMA-based scheduling and the A-DVS algorithm $(\sigma/\sigma_{\max} = 0.2)$.

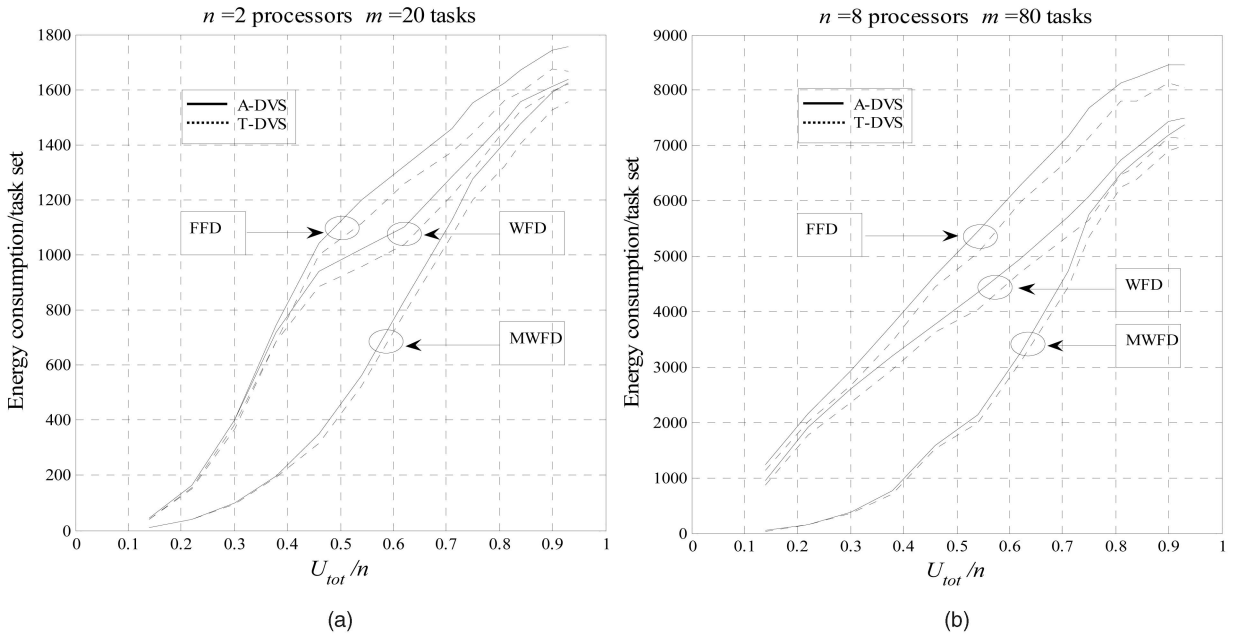

Fig. 7. Energy consumption of the FFD, WFD, and MWFD allocation schemes when used in conjunction with ECRMA-based scheduling and the A/T-DVS algorithm $(\sigma/\sigma_{\max} = 0.8)$.

as is shown in Figs. 5a and 6a. As the system workload increases, the processor utilization (U_j) becomes less significant compared to the utilization bound (U_{bj}), and the ECRMA-based scheme results in a more skewed schedule (unbalanced workload) due to the increased scheduling capacity of each processor. As a result, the energy consumption of FFD and WFD with ECRMA increases compared to the energy consumption of FFD and WFD with the asymptotic RAM in the medium region. For example, the energy consumption of (FFD, WFD) for task sets with an average processor utilization of 0.45 is (731, 706) units with the asymptotic RMA and (1,049, 997) units with ECRMA, as shown in Figs. 5a and 6a. In the saturation region, the energy performance of all the schemes is similar since a high processor utilization leaves lesser opportunities for voltage scaling for saving energy.

Fig. 7 shows the energy consumption of the FFD, WFD, and MWFD allocation schemes when used in conjunction with the ECRMA-based scheduling scheme and the A-DVS and T-DVS algorithms for $\sigma/\sigma_{\max} = 0.8$. It is shown that the energy consumption for the T-DVS algorithm follows the same trend as the energy consumption for the A-DVS algorithm except that T-DVS consumes up to 10 percent less energy compared to A-DVS. For example, in a dual-processor system, the energy consumption of (FFD, WFD, MWFD) for A-DVS and T-DVS is (1,198.0, 1,019.4, 563.1) and (1,114.1, 958.2, 523.7) units, respectively, when the average processor utilization is 0.54.

Overall, in a multiprocessor system using ECRMA for task scheduling, the energy consumption of a feasible task set allocated using MWFD is consistently lower than the energy consumption of the task set allocated using FFD or WFD due to the better load-balancing property of MWFD regardless of the number of processors in the system, as shown in Figs. 5b and 6b, and task set workload characteristics and voltage scaling algorithms, as shown in Fig. 7 for tasks with a utilization variation of $\sigma/\sigma_{\max} = 0.8$ and the A/T-DVS algorithm.

5.3 Fault-Tolerant Characteristics of MWFD

In the presence of faults, the goal of the proposed OFT_MWFD scheme is to minimize energy consumption by dynamically adapting to runtime fault characteristics, such as the number of faults and the time instances of occurrences, and meet the timing requirements for all tasks in case of the worst case faults. Fig. 8 shows the energy consumed by task sets under varying fault occurrences. The system is designed to tolerate up to 10 randomly distributed faults among processors and assumes ECRMA-based scheduling and A-DVS. As is to be expected, the performance of MWFD degrades at a slightly faster rate, albeit very gradually, compared to FFD and WFD. This is because MWFD starts with an energy-optimum solution based on fault-free execution, and each fault occurrence invalidates the optimistic assumption. Nonetheless, even in the presence of all 10 faults, the energy consumption of the MWFD scheme is considerably less than the energy consumption of FFD and WFD, and the trend is independent of the number of processors in the system and the task set workload characteristics, as shown in Figs. 8a and 8b.

Since the probability of transient faults occurring in bursts is increasing, especially in embedded systems deployed in harsh operating environments, fault occurrence in a system may exceed the specified upper bound the system is designed to tolerate. The proposed MWFD-ECRMA scheme further enhances the fault-tolerant capability of such systems because the load-balancing property of MWFD combined with ECRMA leads to improved and balanced scheduling capacity of processors in a system. Fig. 9 shows that in a dual-processor system designed to tolerate up to $K = 5$ faults, a relatively higher number of task sets scheduled using MWFD-ECRMA were found to be feasible when more than the specified number of faults

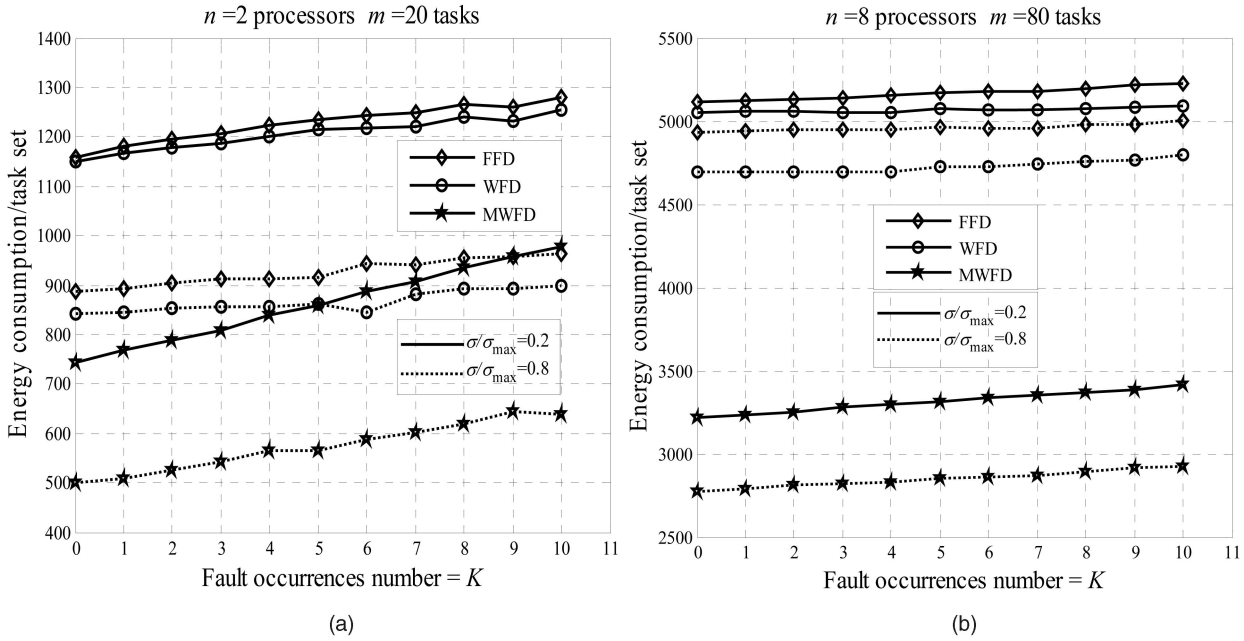

Fig. 8. Energy consumption of the FFD, WFD, and MWFD allocation schemes when used in conjunction with ECRMA-based scheduling and the A-DVS algorithm in the presence of faults.

occur in the system. In other words, MWFD is more resilient to transient system overloading than FFD.

Finally, a number of additional simulations were carried out to validate the energy savings due to the dynamic adaptation of the MWFD-DVS scheme to runtime fault characteristics. It is assumed that the number of faults in a system ranges from 0 to K, where K is the upper bound on faults the system is designed to tolerate, and faults are distributed uniformly among all the processors.

Figs. 10 and 11 show typical simulation runs of the OFT_MWFD algorithm combined with A-DVS and T-DVS, respectively. On the average, the dynamic OFT_MWFD achieves up to 30 percent energy savings compared to the offline OFT_MWFD regardless of the voltage scaling algorithm. For example, in a dual-processor system with nine specified fault occurrences, the average offline energy consumed by a task set scheduled using the offline MWFD-ADVS is (957.1, 645.1) units for $\sigma/\sigma_{\max} = (0.2, 0.8)$, while the average energy consumed by the task set in the runtime is (854.6, 548.5) units for $\sigma/\sigma_{\max} = (0.2, 0.8)$. The energy consumption of task sets allocated and scheduled using the MWFD-TDVS algorithm follows the same trend as the energy consumption of task sets allocated and scheduled using the MWFD-ADVS algorithm except that MWFD-TDVS achieves up to 10 percent energy savings.

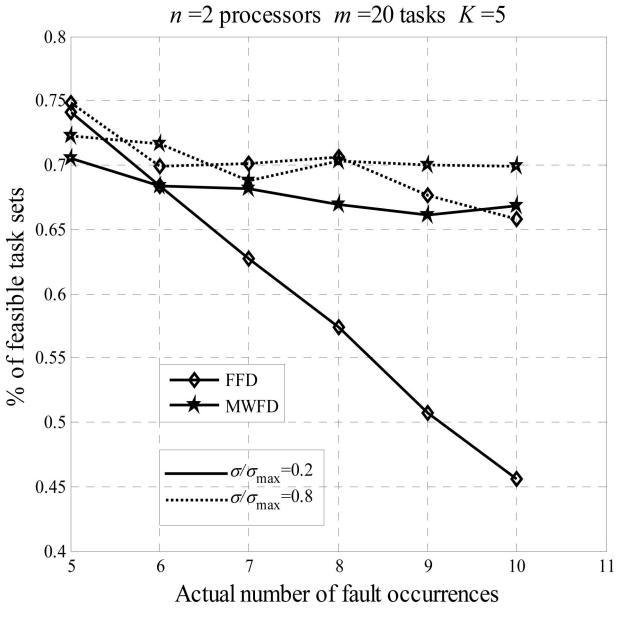

Fig. 9. Performance of FFD and MWFD allocation schemes when faults occur in bursts.

6 Conclusions

This paper presents a low-energy task allocation and scheduling scheme for hard real-time SMP systems. The proposed task allocation scheme, MWFD, is proven to achieve an optimally balanced task partition by opening all processors and sequentially selecting processors in the order of current workload. An optimistic fault-tolerant algorithm, OFT_MWFD, is then proposed to achieve optimum energy savings in the absence of faults while meeting all application timing requirements in the worst case of fault occurrences. The optimistic approach is based on the observation that a fault-free operation is expected to remain dominant in the near future, and designing for corner cases is extremely energy inefficient. Combining OFT_MWFD with an efficient scheduling scheme, which jointly optimizes feasibility performance and DVS policies and dynamically adapts to fault characteristics, significantly reduces the energy consumption without compromising the feasibility or reliability of the system.

Simulation results show that the feasibility performance of MWFD is comparable to other state-of-art schemes, and it

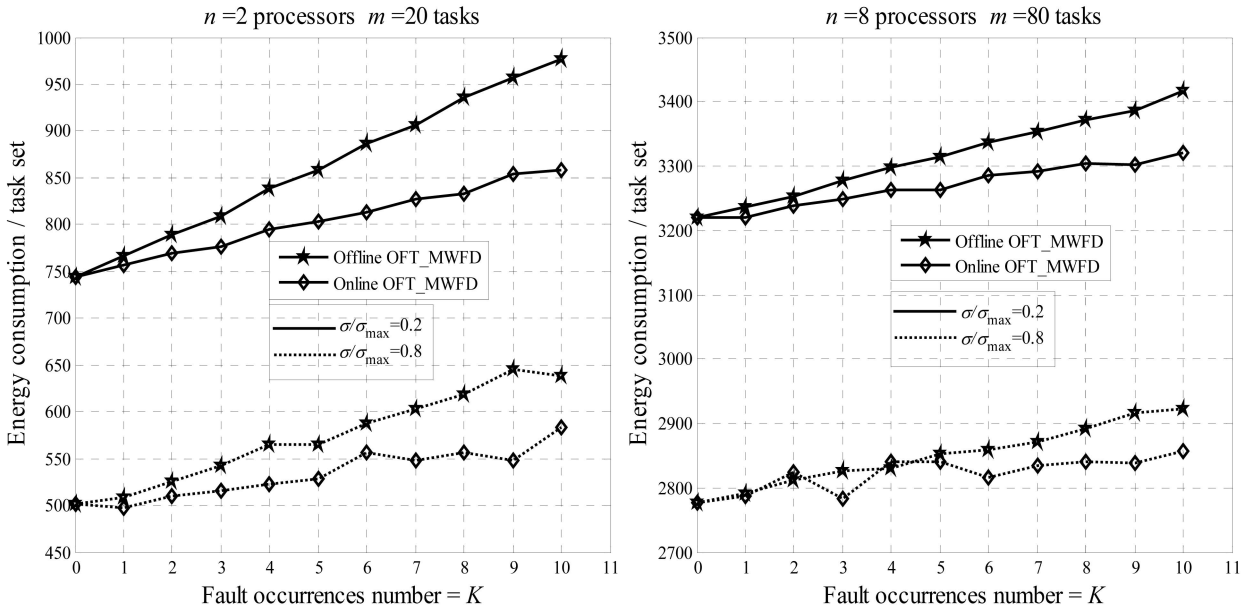

Fig. 10. Runtime energy consumption of OFT_MWFD when used in conjunction with ECRMA-based scheduling and the A-DVS algorithm in the presence of faults.

can achieve up to 70 percent energy savings in the absence of faults and up to 50 percent energy savings in the presence of faults with no degradation of its feasibility performance. The performance of MWFD is consistently independent of the task characteristics, the number of processors in a system, and the voltage scaling algorithms applied on each processor. Further, the MWFD algorithm combined with ECRMA is highly resilient to transient system overloading.

As part of future work, the proposed schemes are being implemented on a multicore motherboard for real-time application development [42]. Considering the increasing application of heterogeneous processors in real-time embedded systems and that in practice, a system function is often provided by a set of related tasks, the proposed schemes will be extended to account for a heterogeneous embedded processor architecture and periodic end-to-end task sets.

ACKNOWLEDGMENTS

This work was in part supported by the Center for Integrated Systems in Sensing, Imaging, and Communication (CISSIC) at MTU. A preliminary version of this paper was presented at the IEEE Workshop on Silicon Errors in Logic-System Effects (SELSE) in 2007.

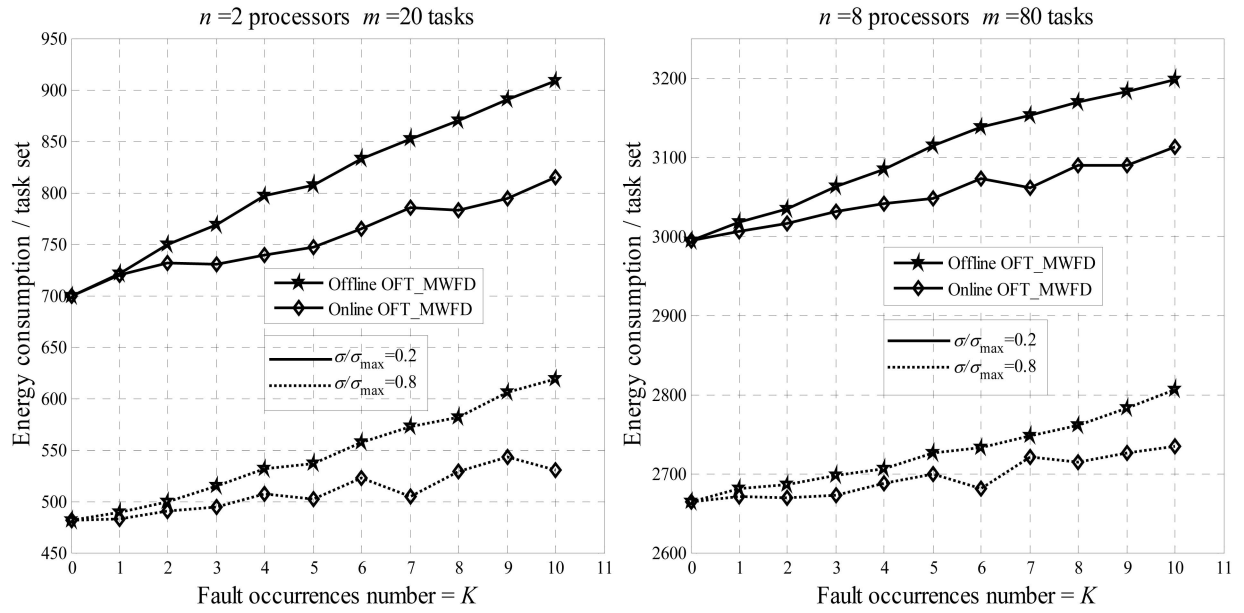

Fig. 11. Runtime energy consumption of OFT_MWFD when used in conjunction with ECRMA-based scheduling and the T-DVS algorithm in the presence of faults.

REFERENCES

[1] S. Reinhardt and S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," *Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA '00),* pp. 25-36, 2000.

[2] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," *Proc. Int'l Conf. Dependable Systems and Networks (DSN '02),* pp. 389-398, 2002.

[3] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid-State Circuits,* vol. 27, no. 4, pp. 473-484, Apr. 1992.

[4] J. Lorch and A.J. Smith, "Software Strategies for Portable Computer Energy Management," *IEEE Personal Comm.,* vol. 5, no. 3, pp. 60-73, June 1998.

[5] L. Benini, A. Bogliolo, and G. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 8, no. 3, pp. 299-316, June 2000.

[6] V. Gutnik and A. Chandrakasan, "An Efficient Controller for Variable Supply-Voltage Low Power Processing," *Proc. Symp. VLSI Circuits,* pp. 158-159, June 1996.

[7] W. Namgoong, M. Yu, and T. Meng, "A High-Efficiency Variable-Voltage CMOS Dynamic DC-DC Switching Regulator," *Proc. IEEE Int'l Solid-State Circuits Conf. (ISSCC '97),* pp. 380-381, Feb. 1997.

[8] R. Melhem, D. Mossé, and E. Elnozahy, "The Interplay of Power Management and Fault Recovery in Real-Time Systems," *IEEE Trans. Computers,* vol. 53, no. 2, pp. 217-231, Feb. 2004.

[9] Y. Zhang and K. Chakrabarty, "Energy-Aware Fault Tolerance in Fixed-Priority Real-Time Embedded Systems," *Proc. Int'l Conf. Computer-Aided Design (ICCAD '03),* pp. 209-213, Nov. 2003.

[10] Y. Zhang and K. Chakrabarty, "Task Feasibility Analysis and Dynamic Voltage Scaling in Fault-Tolerant Real-Time Embedded Systems," *Proc. Design, Automation and Test in Europe Conf. (DATE '04),* vol. 2, pp. 1170-1175, Feb. 2004.

[11] T. Wei, P. Mishra, K. Wu, and H. Liang, "Online Task-Scheduling for Fault-Tolerant Low-Energy Real-Time Systems," *Proc. Int'l Conf. Computer-Aided Design (ICCAD '06),* pp. 522-527, Nov. 2006.

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

[13] D. Mosse, R. Melhem, and S. Ghosh, "Analysis of a Fault-Tolerant Multiprocessor Scheduling Algorithm," *Proc. 24th Int'l Symp. Fault-Tolerant Computing (FTCS '94),* pp. 16-25, June 1994.

[14] Y. Oh and S. Son, "Multiprocessor Support for Real-Time Fault-Tolerant Scheduling," *Proc. IEEE Workshop Architectural Aspect of Real-Time Systems,* Dec. 1991.

[15] S. Ghosh, R. Melhem, and D. Mosse, "Fault-Tolerance through Scheduling of Aperiodic Tasks in Hard Real-Time Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 3, pp. 272-284, Mar. 1997.

[16] S. Albers, F. Müller, and S. Schmelzer, "Speed Scaling on Parallel Processors," *Proc. 19th Ann. ACM Symp. Parallelism in Algorithms and Architectures (SPAA '07),* pp. 289-298, 2007.

[17] D. Zhu, R. Melhem, and B. Chiders, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 7, pp. 686-700, July 2003.

[18] J. Anderson and S. Baruah, "Energy-Efficient Synthesis of Periodic Task Systems upon Identical Multiprocessor Platforms," *Proc. 24th Int'l Conf. Distributed Computing Systems (ICDCS '04),* pp. 428-435, 2004.

[19] J. Chen, H. Hsu, K. Chuang, C. Yang, A. Pang, and T. Kuo, "Multiprocessor Energy-Efficient Scheduling with Task Migration Consideration," *Proc. 16th Euromicro Conf. Real-Time Systems (ECRTS '04),* pp. 101-108, June-July 2004.

[20] C. Yang, J. Chen, and T. Kuo, "An Approximation Algorithm for Energy-Efficient Scheduling on a Chip Multiprocessor," *Proc. Design, Automation and Test in Europe Conf. (DATE '05),* vol. 1, pp. 468-473, Mar. 2005.

[21] Y. Yu and V. Prasanna, "Power-Aware Resource Allocation for Independent Tasks in Heterogeneous Real-Time Systems," *Proc. Ninth Int'l Conf. Parallel and Distributed Systems (ICPADS '02),* pp. 341-348, Dec. 2002.

[22] H. Aydin and Q. Yang, "Energy-Aware Partitioning for Multiprocessor Real-Time Systems," *Proc. 17th Int'l Parallel and Distributed Processing Symp. (IPDPS '03),* p. 9, Apr. 2003.

[23] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *Proc. Real-Time Systems Symp. (RTSS '89),* pp. 166-171, Dec. 1989.

[24] T. AlEnawy and K. Aydin, "Energy-Aware Task Allocation for Rate Monotonic Scheduling," *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '05),* pp. 213-223, Mar. 2005.

[25] J. Stankovic and K. Ramamritham, "The Spring Kernel: A New Paradigm for Real-Time Operating Systems," *ACM SIGOPS Operating Systems Rev.,* vol. 23, no. 3, pp. 54-71, July 1989.

[26] D. Zhu, R. Melhem, and D. Mossé, "The Effects of Energy Management on Reliability in Real-Time Embedded Systems," *Proc. Int'l Conf. Computer-Aided Design (ICCAD '04),* pp. 35-40, Nov. 2004.

[27] E. Normand, "Single Event Upset at Ground Level," *IEEE Trans. Nuclear Science,* vol. 43, no. 6, pp. 2742-2750, Dec. 1996.

[28] X. Fan, C. Ellis, and A. Lebeck, "The Synergy between Power-Aware Memory Systems and Processor Voltage Scaling," Technical Report CS-2002-12, Dept. Computer Science, Duke Univ., 2002.

[29] S. Dhall and C. Liu, "On a Real-Time Scheduling Problem," *Operations Research,* vol. 26, no. 1, pp. 127-140, Jan.-Feb. 1978.

[30] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, Jan. 1973.

[31] *Power Mean Inequality,* http://en.wikipedia.org/wiki/Generalized_mean, 2008.

[32] R. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM J. Applied Math.,* vol. 17, no. 2, pp. 416-429, 1969.

[33] E. Coffman, M. Garey, and D. Johnson, "Approximation Algorithms for Bin Packing: A Survey," *Approximation Algorithms for NP-Hard Problems.* PWS Publishing, pp. 46-93, 1996.

[34] J. Lopez, J. Diaz, and D. Garcia, "Minimum and Maximum Utilization Bounds for Multiprocessor Rate Monotonic Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 7, pp. 642-657, July 2004.

[35] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[36] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. 36th Design Automation Conf. (DAC '99),* pp. 134-139, 1999.

[37] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava, "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors," *Proc. 19th IEEE Real-Time Systems Symp. (RTSS '98),* pp. 178-187, Dec. 1998.

[38] H. Beitollahi and S. Miremadi, "Performance Evaluation of Fault-Tolerant Scheduling Algorithms in Real-Time Multiprocessor Systems," *Proc. IASTED Int'l Conf. Parallel and Distributed Computing and Networks (PDCN '05),* Feb. 2005.

[39] P. Pillai and K.G. Shin, "Real-Time Dynamic Voltage Scaling for Low Power Embedded Operating Systems," *Proc. 18th ACM Symp. Operating System Principles (SOSP '01),* pp. 89-102, 2001.

[40] S. Saewong and R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority Real-Time Systems," *Proc. Ninth IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS '03),* pp. 106-114, May 2003.

[41] J. Lopez, M. Garcia, J. Diaz, and D. Garcia, "Utilization Bounds for Multiprocessor Rate-Monotonic Scheduling," *Real-Time Systems,* vol. 24, no. 1, pp. 5-28, Jan. 2003.

[42] *Multi-Core Board,* http://www.radisys.com/products/datasheet_page.cfm?productdatasheetsid=1406, 2008.

Tongquan Wei received the BE degree in electronics engineering from the Dalian University of Technology, China, and the MS degree in computer engineering from the University of Missouri, Rolla. He is a PhD student in the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton. His research focuses on power and fault-tolerant management in real-time embedded systems.

Piyush Mishra received BE degree in electrical and electronics engineering from Birla Institute, India, and the PhD degree in electrical engineering from Polytechnic University, Brooklyn, New York in 1997 and 2004, respectively. He was an assistant professor of computer engineering in the Department of Electrical and Computer Engineering, Michigan Technological University, Houghton. He is currently with the Electronics and Energy Conversion Group, GE Global Research, Niskayuna, New York. His research interests include reliable real-time systems, resource-optimum security and reliability in embedded computing, and high-performance hardware-software integration. He is the recipient of an IEEE DAC graduate scholarship and a CISCO Information Assurance scholarship and serves on several conference program committees and review boards. He is a member of the IEEE, the Sigma Xi, and the ACM SIGDA.

Kaijie Wu received the BE degree from Xidian University, Xi'an, China, in 1996, the MS degree from the University of Science and Technology of China, Hefei, China, in 1999, and the PhD degree in electrical engineering from Polytechnic University, Brooklyn, New York in 2004. He is currently an assistant professor in the Department of Electrical and Computer Engineering, University of Illinois, Chicago. His research interests include computer-aided design (CAD) of radiation-hardened VLSI systems, countermeasures for side-channel cryptanalysis for cryptodevices, and robust and fault-tolerant nanotechnology designs. He is the recipient of the 2004 EDAA Outstanding Dissertation Award in the "new directions in circuit and system test." He is a member of the IEEE.

Han Liang received the BE and MS degrees from Xi'an Jiaotong University (XJTU), Xi'an, China, in 1999 and 2002, respectively. He is a PhD candidate in the Department of Electrical and Computer Engineering, University of Illinois, Chicago. He has been working with Prof. Kaijie Wu since September 2004. His research interests include computer-aided design, high-performance power-efficient fault-tolerant VLSI systems, and high-speed hardware architectures of cryptographic protocols and algorithms.