Energy Minimization for Fault Tolerant Scheduling of Periodic Fixed-Priority Applications on Multiprocessor Platforms

Qiushi Han*, Ming Fan†, Linwei Niu‡, Gang Quan*,
*Department of Electrical and Computer Engineering, Florida International University, Miami, FL, 33174
‡Broadcom Corporation, 3151 Zanker Road, San Jose, CA 95134
‡Department of Mathematics and Computer Science, West Virginia State University, WV, 25112
Emails: {qhan001, gang.quan}@fiu.edu, mingfan@broadcom.com, lniu@wvstateu.edu

***Abstract*—While technology scaling enables the mass integration of transistors into a single chip for performance enhancement, it also makes processors less reliable with ever-increasing failure rates. In this paper, we study the problem of energy minimization for scheduling periodic fixed-priority applications on multiprocessor platforms with fault tolerance requirements. We first introduce an efficient method to determine the checkpointing scheme that guarantees the schedulability of an application under the worst-case scenario, i.e. up to K faults occur, on a single processor. Based on this method, we then present a task allocation scheme aiming at minimizing energy consumption while ensuring the fault tolerance requirement of the system. We evaluate the efficiency and effectiveness of our approaches using extensive simulation studies.**

***Keywords*—energy-aware, fault tolerance, checkpointing, fixed-priority, partitioning**

I. Introduction

With the continuing advancements in technology scaling, mass integration of transistors into a single chip has been a primary method to enhance computing capabilities. However, with the tremendous increase of transistors in an IC chip, the power consumption has increased dramatically, i.e. in an exponential order [1]. Power/energy awareness has thus been a first-class design concern for the past several decades, and much research has been conducted (e.g. [2]–[5]) to minimize energy consumption while guaranteeing timing and other performance requirements under a large variety of system and task models. In the meantime, the relentless technology scaling has also drastically degraded the reliability of computing systems [6]–[8]. For example, it has been shown in [8] that the soft error rate (SER) per chip of logic circuits increased nine orders of magnitude when the size of transistors shrunk from 600nm to 50nm. For safety-critical systems, e.g. aircrafts and power plants, it is imperative that run-time faults be handled properly in a timely manner.

We are interested in the problem on how to develop real-time systems with joint consideration of energy efficiency and fault tolerance. We are particularly interested in the methods that can judiciously employ the common energy-saving and fault tolerance mechanisms, i.e. dynamic voltage and frequency scaling (DVFS) and checkpointing, respectively, in the design of real-time systems with high energy efficiency and reliability. DVFS dynamically adjusts the supply voltage and working frequency of a processor to reduce power consumption. Most of the modern processors, if not all, are equipped with such capability [9], [10]. The checkpointing with rollback recovery is one of the most popular approaches for fault tolerance [11]–[13]. It consists of storing a snapshot of the current system state and rolling back to it in case of failure.

When dealing with both energy conservation and fault tolerance, one big challenge is how to balance the resource usage between the two, since energy conservation strategies need additional resources for lowering down system speed, and fault tolerance strategies need additional resources for fault detection and recovery. A number of techniques have been presented in the literature. For example, Zhang et al. [13] proposed a genetic algorithm to determine the DVFS schedule and checkpointing interval for fixed-priority tasks aiming at minimizing fault-free energy consumption while tolerating a predefined number of faults. This approach was later extended by Wei et al. [12] to an online scheme that dynamically explores slacks to further save energy consumption. Zhao et al. [14] exploited the concept of sharing recoveries among multiple tasks to reduce the amount of resource reservations, hence leave more space for power management. However, all the aforementioned approaches are restricted to uniprocessor platforms.

There are also several papers published that are closely related to our research. Pop et al. [15] presented a constraint logic programming method to develop fault-tolerant DVFS schedules for real-time tasks with precedence constraints on distributed heterogeneous platforms. The task allocation is assumed to be known a priori. Fault tolerance is achieved by reserving passive backup(s) for a task on the same processor and activating it in case of failure. With the slacks mostly being occupied by reserved recoveries, the space for DVFS is severely limited. Haque et al. [16] proposed a stand-sparing technique for fixed-priority applications on a dual-processor platform. Active replication with delayed starting time is employed for the purpose of maintaining task reliability and reducing energy consumption. Again, an entire task needs to be re-executed in presence of a failure and active replication can consume extra energy even under fault-free scenario. Pop et al. [11] proposed a more comprehensive approach to the synthesis of fault tolerant schedule for applications on heterogeneous distributed systems. They used the combination of checkpointing and active replication to deal with the fault tolerance problem. A meta-heuristic (Tabu search) is constructed to decide the fault tolerance policy, the placement of checkpoints and the mapping of tasks to processors, but energy consumption is not considered in their approach. Han et al [17] proposed an optimal checkpointing scheme for minimize the worst case response time of an application on a single processor and developed a task allocation scheme for energy minimization. However, this approach is limited to frame-based task sets, hence it

This work is supported in part by NSF under project CNS-1423137 and CNS-1018108.

does not apply to a much more complicated fixed-priority periodic task model.

In this paper, we study the problem of minimizing the energy consumption for periodic fixed-priority hard real-time systems running on homogeneous multiprocessor platforms while ensuring that the systems can tolerate up to K transient faults. We adopt the widely used DVFS and checkpointing as the energy management method and the fault tolerance policy, respectively. We focus our efforts on fixed-priority scheduling due to its simpler implementation and better practicability [18] compared with dynamic priority-based scheduling. It is well known that multiprocessor scheduling is an NP-hard problem. Therefore, to solve our problem, we first present an efficient and effective method, i.e. ECHK, on how to identify a checkpointing scheme for fix-priority tasks on a single processor to guarantee their schedulability under the worst case, i.e. K faults occur. We theoretically prove the validity of ECHK and its superiority to the state-of-art technique. We then propose an task allocation scheme, i.e. TACHK, based on ECHK to minimize the system energy consumption. Simulation results show that, we can achieve as much as 13% and 59% energy reduction compared with two different related approaches, respectively.

The rest of the paper is organized as follows. Section II introduces the system models and notations used throughout this paper. We introduce our efficient algorithm for obtaining a feasible checkpointing solution for a given task set on a single processor in Section III. We then present our energy efficient fault-tolerant task-allocation algorithm in section IV. The effectiveness and efficiency of our algorithms are evaluated in Section V. Finally, section VI concludes the paper.

II. Preliminaries

A. Application model

The real-time application considered in this paper consists of n independent sporadic tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task is characterized by a tuple (C_i, D_i, T_i). C_i denotes the worst-case execution time of a task τ_i, whereas D_i and T_i represent its deadline and minimum inter-arrival time (period), respectively. Each task can generate an infinite number of instances or jobs, we use these two terms interchangeably in this paper. The utilization of task τ_i is represented as $u_i = \frac{C_i}{T_i}$. The system utilization U is therefore calculated as $U_{total} = \sum_{i=1}^{n} \frac{C_i}{T_i}$.

B. Fault model and checkpointing

In this paper, we assume that there are at most K transient faults within one least common multiple (LCM) of all the task periods in Γ but we do not make any assumptions regarding the fault pattern. In other words, the transient faults can strike any task instance at any time, and multiple faults may affect the same task instance. Once a fault is detected, the task instance being affected rolls back to the last saved checkpoint and re-executes the faulty segment. We consider the checkpoint to be self fault-tolerant.

We assume all the jobs of the same task have the identical number of checkpoints. Inserting one checkpoint to an instance of task τ_i refers to the operation of saving its current state and condition to memory, with its the timing and energy overhead denoted as o_i and eo_i, respectively. Before inserting a checkpoint, a fault detection is always performed to ensure the sanity of the to-be-saved state. We use q_i and eq_i to denote the timing and energy overhead for such an operation. Moreover, once a fault is detected during the execution of an instance of task τ_i, it needs to rollback to the latest checkpoint, i.e. to retrieve the latest-saved correct information. The time and energy overhead of this operation are represented by r_i and er_i, respectively.

The fault-free execution time of an instance of task τ_i is a function of the number of checkpoints, and is formulated in equation (1a). Note that, with m_i checkpoints, the fault detections are performed $m_i + 1$ times including the one at the end of the job's execution. The recovery time of τ_i with m_i checkpoints under a single failure includes three parts, namely the time to rollback to the latest checkpoint, the time to re-execute the faulty segment and the time to perform a fault detection operation at the end. We denote it as $F_i(m_i)$ and formulate it in equation (1b).

$$C_i(m_i) = C_i + m_i o_i + (m_i + 1) q_i \tag{1a}$$

$$F_i(m_i) = r_i + \frac{C_i}{m_i + 1} + q_i \tag{1b}$$

Since a lower priority task τ_i is subject to the workload interference (including recoveries) from higher priorities tasks, the worst case recovery time for τ_i is the longest recovery time among all tasks with higher priority and τ_i itself. Specifically, we denote it as

$$MR_i = \max(F_1, F_2, ..., F_i). \tag{2}$$

Regarding $\tau_i's$ schedulability, adding more checkpoints to its higher priorities tasks increases the interference caused by fault-free workloads, which may undermine $\tau_i's$ schedulability. However, it may decrease the recovery time needed for τ_i, i.e. MR_i, which is in favor of $\tau_i's$ schedulability. Therefore, to determine the appropriate number of checkpoints for scheduling real-time tasks under the fault tolerance constraint is not a trivial task.

C. Platform and energy model

We assume that there are a total number of ϕ processors on a homogeneous multiprocessor platform Ψ, i.e. $\Psi = \{\psi_1, ..., \psi_\phi\}$ and there exist a set of L-level discrete speeds/frequencies for each processor, which is denoted as $FR = \{f_1, f_2, ..., f_L\}$. Without loss of generality, we assume $0 \leq f_L \leq f_{L-1} \leq ... \leq f_1 = 1$.

We adopt the power model in [17], [19] by considering the frequency-independent and frequency-dependent power components. Specifically, the overall power consumption P can be formulated as

$$P = P_{ind} + P_{dep} = P_{ind} + C_{ef} f^{\alpha}, \tag{3}$$

where P_{ind} is the frequency-independent power, including the power consumed by off-chip devices such as main memory and external devices and constant leakage power. C_{ef} is the effective switching capacitance. α is a constant usually no smaller than 2. P_{dyn} is the dynamic power consumed by switching transistor state. As a result, the fault-free energy consumption of a job from task τ_i with m_i checkpoints executed under speed f_i is calculated as:

$$\begin{aligned} E_i(f_i) = &(P_{ind} + C_{ef} f_i^{\alpha}) \cdot \frac{C_i}{f_i} \\ &+ m_i(eo_i + o_i P_{ind}) + (m_i + 1)(eq_i + q_i P_{ind}), \end{aligned} \tag{4}$$

where the first part is the energy consumed by executing the job (the scaled execution time of task τ_i under frequency f_i is $\frac{C_i}{f_i}$), and the second and the third part represent the energy overhead from checkpointing and fault detections, respectively. Similar to [13], [17], we assume that checkpointing, fault detections and checkpoint retrievals are not affected by processor frequency. Note that, during those

operations, the frequency-independent power is still consumed. As $E_i(f_i)$ is a convex function, one intuition to save energy is to lower the operating frequency as much as possible, provided it is larger than so-called *critical frequency* ($f_c = \sqrt[\alpha]{\frac{P_{ind}}{(\alpha-1)C_{ef}}}$) [20].

Γ_j is used to denote the set of tasks assigned to the processor ψ_j. As we only study the energy consumption within one LCM of the task periods, the energy consumption of processor ψ_j is formulated in equation (5).

$$E(\Gamma_j) = \sum_{\tau_i \in \Gamma_j} \frac{LCM}{T_i} E_i(f_i). \quad (5)$$

The total energy consumption of the system is thus $E(\Gamma) = \sum_{j=1}^{\phi} E(\Gamma_j)$.

III. Feasible checkpointing configuration for fixed-priority tasks on a single processor

Our goal is to minimize the energy consumption while being able to tolerate, in the worst case, K faults when scheduling a fixed-priority task set on a multiprocessor platform. One key to this problem is to choose an appropriate number of checkpoints for each task. Adding more checkpoints to tasks may reduce the recovery overheads, which is in favor of system schedulability. However, excessive checkpointing overheads may outweigh the benefits of decreasing recovery overheads, which might undermine the schedulability of the system. Therefore, to determine the number of checkpoints for each task is not a trivial problem and must be carefully studied.

As a closely related work, Zhang et al. [13] showed that the optimal number of checkpoints to minimize the worst case latency of a single task τ_i, denoted as m_i^*, can be calculated as

$$m_i^* = \begin{cases} \lceil \sqrt{\frac{K*c_i}{o_i+q_i}} - 1 \rceil & \text{if } c_i > \frac{(m_i^-+1)(m_i^-+2)(o_i+q_i)}{K} \\ \lfloor \sqrt{\frac{K*c_i}{o_i+q_i}} - 1 \rfloor & \text{if } c_i \leq \frac{(m_i^-+1)(m_i^-+2)(o_i+q_i)}{K} \end{cases}$$

where $m_i^- = \lfloor \sqrt{\frac{K*c_i}{o_i+q_i}} - 1 \rfloor$. However, when considering multiple fixed-priority tasks on a single processor, the individual optimal checkpointing configuration does not necessarily lead to a feasible checkpointing configuration for a task set.

To this end, Zhang et al. [13] proposed a recursive approach for identifying a feasible checkpointing scheme for a given fixed-priority task set on a single processor. Specifically, the recursive algorithm, i.e. (ZCP(p,q)), takes two parameters p and q as inputs, where p and q are the indexes for the first and last task in the sub-task set with checkpoint numbers to be determined. The algorithm works as follows:

1) Initially, let $m_i = 0$ and obtain m_i^* for $1 \leq i \leq n$. Set $p = 1, q = n$.
2) ZCP(p,q): Starting from the first task τ_p, evaluates the schedulability of each task in decreasing order of task priorities, and finishes successfully if all tasks are determined schedulable.
3) If task $\tau_j, j \in [p,q]$ is not schedulable, the task $\tau_h, h \in [1,j]$ with the longest recovery is found and one more checkpoint is added to it, i.e. $m_h = m_h + 1$ to reduce its recovery time, i.e. $F_h = F_h(m_h)$. Since the addition of checkpoints to τ_h affects the schedulability of the tasks from τ_h to τ_j, we need to set $p = h, q = j$ and recursively call ZCP(p,q).
4) ZCP(p,q) terminates and reports that the task set is unschedulable if, for each task $\tau_i, i \in [1,p]$, the number of checkpoints is larger than m_i^*, i.e. the optimal value for a single task.

This approach works well only for small task sets and/or tasks with small optimal checkpoint numbers. Otherwise, it can be extremely time consuming. Note that, a task τ_i is considered unschedulable only when the checkpoint numbers of all tasks in $\{\tau_1, \tau_2, ..., \tau_i\}$ exceed their individual optimal numbers. In addition, each time when a checkpoint is added to a task τ_i, the schedulability of task τ_i along with all the lower-priority tasks has to be re-evaluated. These two factors contribute the most to the excessive running time of ZCP and make it extremely computational expensive for design space explorations for our multiprocessor energy-efficient fault-tolerant real-time scheduling problem, which is NP-hard in nature.

It is therefore desirable that a more efficient and effective method can be developed to rapidly determine the checkpointing configuration for tasks on a single processor. In what follows, we introduce several theorems, and based on which, we develop a much more efficient algorithm.

Theorem 1: Given a checkpointing configuration $M = \{m_1, ..., m_p, ..., m_n\}$, assume that there exists a task τ_p with $m_p > m_p^*$. Let $M' = \{m_1, ..., m_p^*, ..., m_n\}$. Then if the task set Γ is unschedulable under M', it must also be unschedulable under M.

Proof: Detailed proof can be found in our technical report [21]. ■

Theorem 1 implies that, if the task set is not schedulable when the number of checkpoints of any task has already exceeded its individual optimal number, this task set is deemed to be unschedulable. As a result, there is no need to increase the numbers of checkpoints for other tasks until all of them exceed their individual optimal numbers, as in ZCP algorithm stated above. With larger task sets and larger optimal checkpoint numbers for each tasks, Theorem 1 can improve the computational efficiency tremendously.

In addition, changing the number of checkpoints of a higher priority task also changes its preemption impacts to the low priority tasks and thus results in time-consuming schedulability checking operations. The following theorem helps to greatly reduce the computational cost for schedulability checking.

Theorem 2: Let τ_q be the unschedulable task with the highest priority under the checkpointing configuration $M = \{m_1, ..., m_q, ..., m_n\}$. Assume that τ_q becomes schedulable under a new configuration $M' = \{m_1', ..., m_q', ..., m_n'\}$, $\forall i, m_i' \geq m_i$ when gradually adding checkpoints to tasks with the largest recovery cost. Then, for any higher priority task τ_i, where $i \in [1,q)$, if it is schedulable under M then it must be schedulable under the new configuration M'.

Proof: Detailed proof can be found in our technical report [21]. ■

According to Theorem 2, for the first task τ_q that misses its deadline under a checkpoint scheme, if we are able to incrementally add checkpoints to its higher-priority tasks or itself to make it schedulable, all tasks with priorities higher than τ_q are guaranteed to be schedulable. This theorem can eliminate the computational efforts for re-evaluating the schedulability of higher priority tasks when inserting the checkpoints to them. Based on Theorem 1 and 2, we formulate an efficient and effective algorithm for finding a feasible checkpointing configuration for fixed-priority tasks on a single processor, as shown in Algorithm 1.

ECHK evaluates the schedulability of each task from the highest priority to the lowest. If an unschedulable task τ_i is encountered, ECHK searches for the checkpointing configuration to make τ_i schedulable by repeatedly adding checkpoints to a higher priority task or τ_i that currently contributes the most to $\tau_i's$ recovery, an

Algorithm 1 ECHK(Γ, K)

Require:
1) Task set : $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$;
2) Number of faults: K

flag = "task set schedulable"
obtain m_i^*, for $i = 1, 2, ..., n$ according to [11]
$\forall i, i =, 1, 2..., n$, initialize m_i to 0;
for $(i = 1; i < n+1; i++)$ **do**
  while τ_i is not feasible **do**
    $F_h = \max(F_1, ..., F_i)$;
    $m_h = m_h + 1$;
    if $m_h > m_h^*$ **then**
      $flag$ = "task set unschedulable"
      return $flag$
    end if
  end while
end for
return $flag$, $M = \{m_1, m_2, ..., m_n\}$

termination condition is set according to Theorem 1. If an feasible checkpointing configuration is found, then the schedulability of all the tasks with higher priorities than τ_i is guaranteed based on Theorem 2.

Algorithm 1 greatly simplifies the process of searching for a feasible checkpointing combination for a given task set on a single processor. The complexity of *ZCP* is $O(\prod_{i=1}^{n} m_i^* \cdot nT)$, where T is the longest time for evaluating the schedulability of a task using exact response time analysis, whereas our ECHK has a complexity of at most $O(\sum_{i=1}^{n} m_i^* \cdot nT)$. Moreover, given that our algorithm can determine a task set to be unschedulable as soon as the number of checkpoints of any task exceeds its individual optimal value, ECHK is much more efficient in practice.

IV. Energy Aware Task allocation

Based on our algorithm ECHK that guarantees the uniprocessor fault tolerance, we now present an algorithm determining the task allocation and the corresponding DVFS schedule on multiprocessor platforms to minimize the overall energy consumption.

Without the fault tolerance requirement, one intuitive method is to balance the workload among multiprocessor platforms as much as possible [2] such that each processor can run at a relatively low speed. When we take fault tolerance into account, however, extra care must be taken since both recovery reservation and energy management compete for system resources. The amount of reserved resources heavily depends on the feasible checkpointing scheme that can be obtained for a given task set. Balancing the workload does not necessarily leads to a favorable checkpointing scheme, since the system utilization itself does not provide any information regarding the fault-tolerant schedulability of a task set. On the other hand, packing as many tasks as possible into one processor helps to reduce the number of processor to be utilized, but leaves less space for slowing down the processor.

In what follows, we focus our effort on developing an effective heuristic for jointly determining the task allocation, checkpointing configuration and DVFS schedule for fixed-priority task sets scheduled on multiprocessor platforms, as it is a NP-Hard problem in strong sense [11].

Our task allocation scheme for energy minimization with K-fault tolerance capability is developed based on the algorithm *ECHK*. The

Algorithm 2 TACHK(Γ, Ψ, K)

Γ_j = NULL, for $j = 1, 2, ..., \phi$;
for $i = 1$; $i \le n$; $i++$ **do**
  $feasible\_speed_i = f_{max}$;
  assigned = 0;
  for $j = 1$; $j \le \phi$; $j++$ **do**
    $\{flag, M_{temp}\}$ = ECHK($\Gamma_j \cup \tau_i$, K);
    if $(!flag)$ **then**
      continue;
    end if
    $speed_{temp} = determine\_core\_speed(\Gamma_j \cup \tau_i, K)$;
    if $speed_{temp} < feasible\_speed$ **then**
      assigned = j; $feasible\_speed_i = speed_{temp}$;
    end if
  end for
  if $assigned == 0$ **then**
    return "not schedulable";
  else
    $\Gamma_{assigned} \leftarrow \Gamma_{assigned} \cup \{\tau_i\}$;
  end if
end for
calculate the energy consumption E_{total} according to equation (4) and (5);
return $\{\Gamma_1, ..., \Gamma_\phi\}, E_{total}$

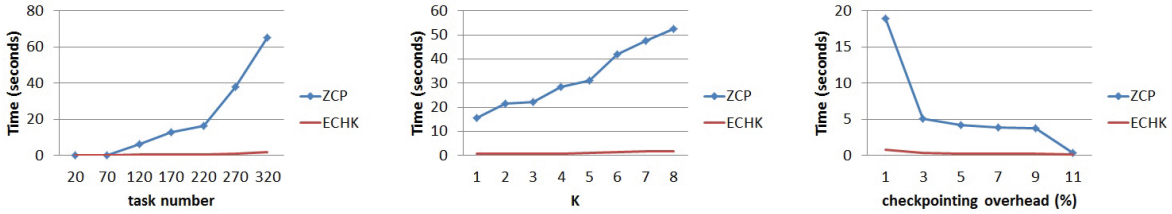
Algorithm 3 *determine_core_speed*(Γ, K)

$lowest\_feasible\_speed = f_{max}$;
sort the available discrete speeds of the processors,i.e. FR in decreasing order;
for $i = 1$; $i \le |FR|$; $i++$ **do**
  Γ_{temp}: temporary task set resulting from Γ scaled by frequency FR[i];
  flag = ECHK(Γ_{temp}, K);
  if $(!flag)$ **then**
    break;
  else
    $lowest\_feasible\_speed = FR[i]$;
  end if
end for
return $lowest\_feasible\_speed$

overall algorithm is described in Algorithm 2. Specifically, when allocating a new task τ_i, we tentatively assign τ_i to each processor and determine whether a feasible checkpointing can be obtained. For each feasible candidate processor ψ_j, we search for the lowest constant speed that can guarantee the schedulability of all the tasks assigned to it according to Algorithm 3. As excessive frequency switching can cause significant overhead, we use a constant speed for each processor. Then, τ_i is allocated to the processor with the lowest possible speed among all the feasible candidates.

In Algorithm 3, we reduce the speed of a core one level at a time until the lowest speed that yields a feasible checkpointing scheme is reached. Therefore, the complexity of our algorithm greatly hinges on that of ECHK. We assume that the re-execution of a faulty task is always performed at the highest speed, given the probability of failure is low. The checkpointing overhead is considered independent of the processor's running mode.

It is not difficult to see that the overall complexity of Algorithm



(a) varying the number of tasks (b) varying the number of K (c) varying checkpoint overhead

Fig. 1. Time complexity comparison, ECHK vs. ZCP

3 is $O(nL\sum_{i=1}^{n} m_i^* \cdot T)$, where L is the number of available processor frequencies and T is the longest time for evaluating the schedulability of a task using exact response time analysis. Furthermore, the complexity of Algorithm 2 is $O(n^2 \cdot \phi \cdot L\sum_{i=1}^{n} m_i^* \cdot T)$.

V. Experimental results

In this section, we use simulations to verify the effectiveness and efficiency of our proposed algorithms.

A. Timing complexity evaluation

Firstly, we evaluate the timing complexity of our algorithm *ECHK* against the method proposed in [13], i.e. ZCP, on a uniprocessor platform.

We set the system utilization to be 0.8. Note that, we fixed the system utilization to a high value such that the task set generated was not schedulable under faults without checkpointing. The period of each task was randomly selected in the range [10,1000]. The rest of the task parameters were generated according to UUNIFAST in [22]. In our experiments, ZCP can easily fail even with a small number of task when the execution ratio, i.e. $\frac{C_{max}}{C_{min}}$ is very large (e.g. > 100), where C_{max} and C_{min} are the longest and shortest execution time in the task set, respectively. This is due to the fact that it may keep adding checkpoints to the task with a number of checkpoints already larger than its optimal value and thus incurs unnecessary recursions. Therefore, we first modified the ZCP according to Theorem 1. The running times of ECHK and ZCP greatly rely on the following three factors: the number of tasks, checkpointing overhead and the number of faults. We conducted experiments regarding each factor and recorded the results as shown in Figure 1. The result of each test case is the average from over 1000 task sets.

In Figure 1(a), we set $K = 2$ and the checkpointing overhead of each task τ_i as 3% of its worst case execution time, i.e. C_i. The number of tasks was varied from 20 to 320 with a step of 50. As can be seen from the figure, our approach ECHK significantly outperforms the method ZCP. ECHK can achieve a speedup with the maximum of 38X and 20X in average.

Next we evaluated the impact of increasing K on running time of ECHK and ZCP, respectively. The number of tasks was set to 200 and the checkpointing overhead of each task τ_i was fixed at 3% of its worst case execution time, i.e. C_i. As expected, our ECHK performs much better in terms of timing complexity. In this case, ECHK can achieve a maximum speedup of 37X and an average speedup of 30X.

Finally, we studied the effects of increasing checkpointing overhead. The checkpointing overhead was varied from 1% to 11% of the worst case execution of each task, and the numbers of faults and tasks were set to 2 and 200, respectively. As shown in Figure 1(c), when the checkpointing overhead increases, the individual optimal number of checkpoints for each task decreases, hence the search space becomes smaller. While running time of both algorithms decrease, our algorithm can achieve a speedup of 16X in average.

In conclusion, our algorithm ECHK is significantly more efficient than ZCP and more scalable in terms of task numbers, the number of faults and checkpoint overhead.

B. Energy performance evaluation

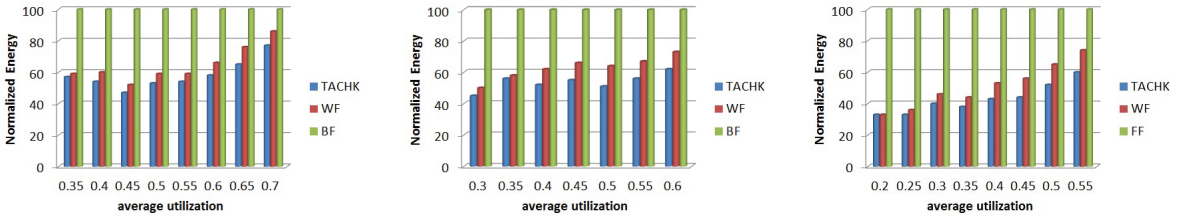
Next, we evaluated the effectiveness of our algorithm *TACHK*.

To our best knowledge, there is no existing approach that solves the exact same problem. Therefore, we evaluated our algorithm against two widely used fault-oblivious approaches, i.e. Best-Fit (BF) and Worst-Fit (WF). In particular, WF is well-known for its effectiveness in fault-oblivious energy reduction as it balances the workload among different processors [2].

To make BF and WF fault-tolerant, we propose a two-step approach. The first step is to identify a feasible task allocation solution. Similar to our approach TACHK, we tentatively allocate the current task to each processor. We use ECHK to check if a processor is a feasible candidate. BF (WF) allocates a task to a feasible processor with the least (most) remaining capacity, i.e. the spare utilization. After obtaining a feasible allocation solution, Algorithm 3 is used to find the lowest constant speed for each processor and then the total energy consumption is calculated. The energy consumptions of *TACHK* and WF are normalized with respect to that of BF.

To evaluate the energy saving performance, we set up the simulation platform as follows. For a fixed number (ϕ) of processors, we varied the average utilization, i.e. $\frac{U_{total}}{\phi}$ from 0.2 (light load) to 0.8 (heavy load). The period of each task τ_i was uniformly distributed in the range $[10, 1000]$. The rest of task parameters were generated according to UUNIFAST [22]. The fault detection, checkpointing and state retrieval overhead were identically set to 1%, 3% and 3% respectively for each task. The corresponding energy overheads were set to 1%, 3% and 3% of the dynamic energy under f_{max} for each task. In addition, we set $P_{ind} = 0.1$, $C_{ef} = 1$ and $\alpha = 3$ [17] and we assumed the existence of the normalized frequency in the range of [0.2, 1] with a step of 0.05. The faults were randomly generated with a probability of $\frac{K}{LCM}$, where K is the maximum number of faults the system needs to tolerate and *LCM* is the least common multiple of all task periods, respectively.

We present three sets of experimental results with various numbers of tasks, processors and total transient faults. Each value reported in the figure is averaged over 1000 test cases. Figure 2(a) shows the energy consumption for a 4-processor system with 40 tasks and K=2. We can see the energy consumption increases



(a) 40 tasks on 4 processors, K = 2 (b) 80 tasks on 8 processors, K = 5 (c) 160 tasks on 16 processors, K = 10

Fig. 2. Energy comparison of different approaches

for all three techniques as the system workload becomes heavier, but our approach TACHK always outperforms the other two. For instance, when the processor average utilization is 0.65, 12%(34%) energy saving is achieved by TACHK over WF (BF). Our algorithm achieves a energy reduction of 7.5% (38.4%) in average when comparing with WF (BF). The energy savings are more substantial in Figure 2(b), on a 8-processor system with 80 tasks that can tolerate 5 faults, TACHK in average saves 10% and 46% energy over WF and BF, respectively. Similarly, for the case of a 16-processor system with 160 tasks that can tolerate at most 10 faults as shown in Figure 2(c), 13% and 59% energy savings are achieved over WF and BF, respectively. In general, we can see that our approach becomes more effective when system utilizations and/or the number of tasks/processors become larger. The reason is that at each step, our approach TACHK tries to determine the best combination of task allocation, checkpointing configuration and speed assignment.

VI. Conclusion

With relentless technology scaling and mass integration of transistors into a single chip, the exponentially increased power consumption and the severely degraded reliability have become first-class design issues in modern computing systems. In this paper, we study the energy minimization problem for hard real-time fixed-priority systems running on multiprocessor platforms that can tolerate up to K transient faults. We propose a solution to this problem by jointly considering the task allocation, checkpoint configuration and speed assignment. We first develop an efficient method to judiciously determine the checkpointing scheme that can guarantee the schedulability of a task set on a single processor. From our theoretical analysis and simulation results, we can see that this algorithm is much more efficient than the state-of-art technique. We then present an algorithm that comprehensively takes the task allocation, checkpointing scheme and speed assignment into account for designing systems with high energy-efficiency and fault-tolerance requirements. Its efficiency and effectiveness are clearly validated by extensive simulation results.

References

[1] T. Skotnicki, J. Hutchby, T.-J. King, H.-S. Wong, and F. Boeuf, "The end of cmos scaling: toward the introduction of new materials and structural changes to improve mosfet performance," *Circuits and Devices Magazine, IEEE*, vol. 21, no. 1, pp. 16 – 26, jan.-feb. 2005.

[2] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," in *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, March, pp. 213–223.

[3] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proceedings of the 38th annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 828–833. [Online]. Available: http://doi.acm.org/10.1145/378239.379074

[4] B. Mochocki, X. Hu, and G. Quan, "A unified approach to variable voltage scheduling for nonideal dvs processors," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 9, pp. 1370 – 1377, sept. 2004.

[5] G. Quan and L. Niu, "Fixed priority scheduling for reducing overall energy on variable voltage processors," in *In 25th IEEE Real-Time System Symposium*. IEEE Computer Society, 2004, pp. 309–318.

[6] R. Lawrence, "Radiation characterization of 512mb sdrams," in *Radiation Effects Data Workshop, 2007 IEEE*, vol. 0, july 2007, pp. 204 –207.

[7] T. Langley, R. Koga, and T. Morris, "Single-event effects test results of 512mb sdrams," in *Radiation Effects Data Workshop, 2003. IEEE*, july 2003, pp. 98 – 101.

[8] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, 2002, pp. 389 – 398.

[9] Intel, "Intel xeon processor." [Online]. Available: http://www.intel.com/content/www/us/en/intelligent-systems/crystal-forest-server/xeon-e5-v2-89xx-chipset.html

[10] AMD, "Amd g-series." [Online]. Available: http://www.amd.com/us/products/embedded/processors/Pag\-es/g-series.aspx

[11] P. Pop, V. Izosimov, P. Eles, and Z. Peng, "Design optimization of time- and cost-constrained fault-tolerant embedded systems with checkpointing and replication," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 3, pp. 389–402, March 2009.

[12] T. Wei, P. Mishra, K. Wu, and J. Zhou, "Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1386–1399, Jun. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.01.020

[13] Y. Zhang and K. Chakrabarty, "A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 1, pp. 111 – 125, jan. 2006.

[14] B. Zhao, H. Aydin, and D. Zhu, "Energy management under general task-level reliability constraints," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, april 2012, pp. 285 –294.

[15] P. Pop, K. H. Poulsen, V. Izosimov, P. Eles, and M. M. Dept, "Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems," CODES+ISSS' 2007.

[16] M. Haque, H. Aydin, and D. Zhu, "Energy management of standby-sparing systems for fixed-priority real-time workloads," in *Green Computing Conference (IGCC), 2013 International*, June 2013, pp. 1–10.

[17] Q. Han, M. Fan, and G. Quan, "Energy minimization for fault tolerant real-time applications on multiprocessor platforms using checkpointing," in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, Sept 2013, pp. 76–81.

[18] R. I. Davis and A. Burns, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Refuted, Revisited and Revised. Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[19] B. Zhao, H. Aydin, and D. Zhu, "Generalized reliability-oriented energy management for real-time embedded applications," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, june 2011, pp. 381 –386.

[20] Y. Liu, H. Liang, and K. Wu, "Scheduling for energy efficiency and fault tolerance in hard real-time systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 1444 –1449.

[21] Q. Han, M. Fan, L. Niu, and G. Quan, "Energy minimization for fault tolerant scheduling of periodic fixed-priority applications on multiprocessor platforms." [Online]. Available: https://drive.google.com/file/d/0B3AHHrHn92eGcXN6cEdJMlJiSUk/view?usp=sharing

[22] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, no. 1-2, pp. 129–154, May 2005. [Online]. Available: http://dx.doi.org/10.1007/s11241-005-0507-9