Energy Minimization for Fault Tolerant Real-Time Applications on Multiprocessor Platforms Using Checkpointing

Qiushi Han Ming Fan Gang Quan
Electrical and Computer Engineering Department
Florida International University
Miami, FL, 33174
{qhan001,mfan001,gaquan}@fiu.edu

***Abstract*—Relentless technology scaling not only dramatically increases the energy consumption of modern processors, it also makes processors less reliable. In this paper, we study the energy minimization problem for real-time applications on multi-processor platforms while tolerating K transient faults using checkpointing. We first introduce an efficient method to determine the checkpointing scheme that minimizes the worst-case response time for a task set that shares the reserved recoveries on a single processor. We then present a fault-tolerant task assignment algorithm to minimize the overall energy. Experimental results show that the proposed algorithm significantly outperforms other related approaches in energy savings.**

***Keywords*—energy-minimization, fault-tolerance, checkpointing, shared recovery**

I. Introduction

As the aggressive scaling in transistor size continues, more and more transistors are integrated into a single die to boost computing performance, which causes the power consumption of IC chips to increase exponentially [1]. This in turn poses severe constraints on operations of real-time systems, especially the battery-operated ones due to their limited energy supply. For the past two decades, energy awareness has become a first-class design constraint. Dynamic voltage and frequency scaling (DVFS) is one of the most popular and widely deployed schemes to conserve energy consumption. DVFS dynamically adjusts the supply voltage and working frequency to reduce power consumption at the cost of increasing response time, which may undermine the performance of real-time systems. Therefore, extensive studies have been proposed (e.g. [2], [3]) to guarantee timing constraints while minimizing energy consumption with DVFS-enabled settings for various system/task models.

Meanwhile, as transistors become smaller and smaller, the reliability of IC chips is increasingly becoming a serious concern. First, the rapidly decreased feature size of the transistors has dramatically increased the rate of radiation-induced faults, up to several orders of magnitude [4]. Second, the ever-increasing on-chip power consumption and temperature have imposed serious threats for the lifetime reliability of IC chips[5]. Due to the nature of safety-critical real-time systems, e.g. automobiles and industrial controls, catastrophical consequences may occur if system faults can not be handled timely or properly.

Processor faults can be largely classified as *transient* or *permanent* [6]. A transient fault happens for a short period time and then disappears without physical damage to the processor. On the contrary, a permanent fault disables a processor permanently. In this paper, we only consider transient faults because they occur more frequently than permanent faults (with a ratio of 100:1 or higher) [7]. Fault tolerance is usually achieved through *time redundancy* or *backward error recovery*. Checkpointing with rollback recovery provides an efficient method to reduce reexecution time in the presence of faults and is well adopted by many researchers [7], [8]. We adopt checkpointing scheme in our research to address the fault tolerance issue.

A plethora of techniques has been presented in the literature on real-time scheduling with both fault tolerance and energy minimization requirements. For example, Zhang et al. [8] introduced a static combination of checkpointing and DVFS scheme for fixed-priority tasks for tolerating K transient faults while minimizing energy consumption. This approach was extended by Wei et al. [9] to explore run-time slacks for further reducing energy consumption. Zhao et al. [10] considered the negative effects of DVFS on transient fault rate and proposed a task-level reliability model. They developed algorithms to determine DVFS schedules and resource-reservation schemes to minimize energy consumption while meeting task-level reliability requirements. All these approaches are restricted to uniprocessor platforms.

As more and more transistors are integrated to the same chip, and due to problems such as the power/thermal issues and limitations in instruction level parallelism [11], multi-processor platforms are becoming mainstream. As a result, most of the research efforts are turned to multi-processor platforms. Pop et al. [12] presented a constraint logic programming method to design low-power fault-tolerant hard real-time applications on distributed heterogeneous platforms. They assumed that the task allocation is fixed and known a priori, and an entire task needs to be re-executed when a transient fault occurs. Qi et al. [13] derived a reliability-aware global scheduling scheme aiming at reducing the system energy consumption for a set of framework-based tasks running

on a homogeneous multi-processor platform. They assumed that different tasks can share the same reserved sources to recover when faults happen. Again, the entire task has to be re-executed in case of faults, which can greatly affect the energy efficiency of the system. Pop et al. [7] 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.

We are interested in the problem of minimizing energy consumption while tolerating up to K transient faults with checkpointing scheme for a real-time system running on a homogeneous multiprocessor platform. A key to solve this problem is to make the judicious tradeoffs between the number of checkpoints for each task and the amount of reserved resources for fault recovery. In this paper, we first study the problem on how to identify the appropriate numbers of checkpoints for tasks on a single processor to minimize the worst case response time. Based on the results, we then develop an efficient method to optimize the energy consumption for a real-time application while ensuring that K transient faults can be tolerated. From our simulation study, our approach can significantly outperform the related approaches.

The rest of the paper is organized as follows. Section II introduces the system models and notations used throughout this paper. Section III presents our method to minimize the worse case latency for real-time tasks on a single processor. We then present our energy efficient fault-tolerant 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 applications considered in this paper consist of n independent tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. All tasks in Γ have the same deadline D, but with different execution requirements. We denote the execution time of τ_i as c_i. The utilization of task τ_i is represented as $u_i = \frac{c_i}{D}$. The system utilization U is therefore calculated as $U_{total} = \sum_{i=1}^{n} \frac{c_i}{D}$.

B. Fault model and checkpointing

In this paper, we only consider transient faults that can be tolerated by backward rollback recoveries. We assume that the system needs to tolerate K faults, and the faults can happen on any of the processors and at any time, even in a burst manner. Run-time faults are countered by rolling back to the latest checkpoints and re-executing the corrupted segments. Checkpointing is considered to be self fault-tolerant.

The timing and energy overhead of inserting one checkpoint to task τ_i (saving the fault-free state) are denoted by o_i and eo_i, respectively. In addition, we use r_i (er_i) to denote the time (energy) it takes to retrieve the information needed to rollback to the latest checkpoint when a fault happens during the execution of τ_i. Fault detection is performed at each checkpoint to ensure the correctness of the saved state. The timing and energy overhead for such an operation are represented as q_i and eq_i, respectively. Assuming m_i checkpoints inserted into τ_i, fault detections will be performed for total $(m_i + 1)$ times (including one fault detection at the end of τ_i's execution). Therefore, the fault-free execution time of τ_i with m_i number of checkpoints, denoted as $c'_i(m_i)$, can be specified as shown in equation (1a). The recovery time of τ_i with a single failure, denoted as $R_i(m_i)$, is shown in equation (1b).

$$c'_i(m_i) = c_i + m_i(o_i + q_i) + q_i \tag{1a}$$

$$R_i(m_i) = r_i + \frac{c_i}{m_i + 1} + q_i \tag{1b}$$

C. Platform and energy model

We consider a homogeneous multiprocessor platform Ψ with m processors, i.e. $\Psi = \{\psi_1, ..., \psi_m\}$. We assume all the processors are identical in terms of processing frequency and power characteristics. For the ease of presentation, we assume the speed/frequency of a processor can be changed continuously in $[f_{min}, f_{max}]$ with $0 \leq f_{min} \leq f_{max} = 1$. As discussed later in this paper, this constraint can be easily relaxed to accommodate the fact that most practical processors support a set of discrete levels of frequencies.

Our system-level power model is similar to that in [14] by distinguishing the dynamic and leakage power components. Specifically, the overall power consumption P can be formulated as

$$P = P_{leak} + P_{dyn} = P_{leak} + C_{ef} f^{\alpha} \tag{2}$$

where P_{leak} is the constant leakage power and can be only eliminated by turning down the processor. C_{ef} is the effective switching capacitance. α is a constant usually larger than 1. P_{dyn} is the dynamic power consumed when the device changes logic states. Hence, the energy consumption of a task τ_i with m_i checkpoints running under the frequency f_i can be expressed as:

$$E_i(f_i) = (P_{leak} + C_{ef} f_i^{\alpha}) \cdot \frac{c_i}{f_i} + m_i(eo_i + o_i P_{leak}) + (m_i + 1)(eq_i + q_i P_{leak}), \tag{3}$$

which includes the energy consumption incurred by executing task τ_i and the energy overheads caused by checkpointing and fault detections. Similar to [8], we consider checkpointing, fault detections and checkpoint retrievals are frequency independent, but leakage power is still consumed during their operations. As $E_i(f_i)$ is a convex function, the minimum system energy is achieved when f_i is as small as possible, provided it is larger than so-called *critical frequency* ($f_c = \sqrt[\alpha]{\frac{P_{leak}}{(\alpha-1)C_{ef}}}$) [14].

We use Γ_j to represent the set of tasks assigned to the processor ψ_j. The energy consumption of processor ψ_j can be calculated using equation (4).

$$E(\Gamma_j) = \sum_{\tau_i \in \Gamma_j} E_i(f_i). \tag{4}$$

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

III. Optimal checkpointing scheme for minimizing the worst case latency on a single processor

Our goal is to develop a method that can minimize the energy consumption while ensuring the K-fault tolerance using checkpointing. A key to solve this problem is to make judicious decisions on inserting checkpoints to each task. As shown in the previous section, increasing the numbers of checkpoints for real-time tasks incurs larger checkpointing overhead which may compromise the feasibility and/or energy efficiency of real-time systems. On the other hand, however, increasing the checkpoint numbers decreases the needs of larger resource reservation for fault recovery, which can be in favor of both system feasibility and energy efficiency. As a result, the number of checkpoints (or the checkpointing interval) must be carefully chosen to balance the checkpointing overhead with the fault recovery cost.

As a closely related work, Zhang et al. [8] 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 tasks that share recovery resources on a single processor, the individual optimal checkpointing configuration does not necessarily lead to the global optimal result. Pop et al. [7] resorted to meta-heuristic (i.e. Tabu search) to search for the global optimal solution. It is desirable that a more efficient and effective method can be developed to identify the optimal global checkpointing settings, especially during the design space exploration process.

Assuming all tasks on the same processor share the same recovery resources, to tolerate K faults, we must reserve enough CPU time, i.e. $K \times SR$, to re-execute the corresponding program segments, where $SR = \max_{i=1,...n} \{R_i(m_i)\}$, m_i is the number of checkpoints for τ_i, and $R_i(m_i)$ is defined in equation (1b). We call SR as the *shared recovery block*. Considering the task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ is allocated to the same processor, the worst case latency of task set Γ with shared recovery block of SR, denoted as $L(\Gamma, SR)$, can be formulated in equation (5)

$$L(\Gamma, SR) = \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} (m_i * o_i + m_i * q_i + q_i) + K * SR. \quad (5)$$

To find the optimal checkpointing scheme that minimize the worst case latency, i.e. $L(\Gamma, SR)$, we have the following theorem.

Theorem 1: If $\{m_1, m_2, ..., m_n\}$ is the optimal checkpointing configuration to minimize $L(\Gamma, SR)$, then we have $\forall i, m_i \leq m_i^*$, where m_i^* is the optimal number of checkpoints to run a task τ_i individually.

Proof: We prove this theorem by contradiction. $\{m_1, m_2, ..., m_n\}$ is assumed to be the optimal checkpointing configuration but $\exists i \in [1,n], m_i > m_i^*$. Let $\{m_1, m_2, ..., m_i^*, ...m_n\}$ be another configuration that distinguishes the former one only by the number of checkpoints for task τ_i. SR and SR' denote the sizes of the shared recovery blocks under two configurations, respectively. δ represents the difference between the two worst case latencies, i.e. $\delta = L(\Gamma, SR) - L(\Gamma, SR')$. Then, we have $\delta = m_i(o_i + q_i) + K * SR - (m_i^*(o_i + q_i) + K * SR')$ according to equation (5).

Note that SR can be potentially increased after reducing m_i to m_i^*, we discuss the two possible scenarios separately in the following.

- *Case 1:* $R_i(m_i^*) \leq SR$. In this case, reducing m_i to m_i^* does not change the size of the shared recovery block, i.e. $SR' = SR$. Because $m_i > m_i^*$, we know $\delta > 0$.
- *Case 2:* $R_i(m_i^*) > SR$. This means that the share recovery block is increased due to the decrease in the checkpointing number of task τ_i and $SR' = R_i(m_i^*)$. Since $SR \geq R_i(m_i)$, if we replace SR (SR') with $R_i(m_i)$ ($R_i(m_i^*)$), respectively, we have

$$\delta \geq m_i(o_i + q_i) + K * R_i(m_i) - (m_i^*(o_i + q_i) + K * R_i(m_i^*)). \quad (6)$$

Note that the right hand side of equation (6) represents the difference of two worst case latencies when running τ_i individually using two different checkpointing schemes. Since m_i^* is the optimal checkpoint solution, we must have $\delta > 0$.

For both cases, we have $\delta > 0$. This contradicts our assumption that M is optimal. ■

Theorem 1 helps to prune the search space for the checkpointing configurations. However, a brute-force method based on Theorem 1 still has a very high computational complexity, i.e. $\prod_{i=1}^{n} m_i^*$, which can be computationally prohibitive for large task sets with a considerable amount of possible values of m_i^*. In what follows, we introduce a novel approach to further prune the search space.

Since $SR = \max_{i=1,...n} \{R_i(m_i)\}$, from equation (1b), for a given SR, we have

$$m_i = \lceil \frac{c_i}{SR - (r_i + q_i)} - 1 \rceil. \quad (7)$$

Therefore, equation (5) can be transformed to

$$L(\Gamma, SR) = \sum_{i=1}^{n} (c_i + q_i) + \sum_{i=1}^{n} \lceil \frac{c_i}{SR - (r_i + q_i)} - 1 \rceil (o_i + q_i) + K * SR \quad (8)$$

Therefore, to search for the optimal checkpointing configurations, we only need to search the optimal value of SR that can optimize $L(\Gamma, SR)$. To achieve this purpose, we first introduce the following lemma.

Lemma 1: If $M = \{m_1, m_2, ..., m_n\}$ is the optimal checkpointing configuration for task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$, then the size of the shared recovery block SR under configuration M is no less than $\max_{i=1,...n} \{R_i(m_i^*)\}$.

Proof: The proof of Lemma 1 is similar to that of Theorem 1. We also prove it by contradiction. The configuration M is assumed to be optimal but the resulting $SR <$

$\max_{i=1,...n}\{R_i(m_i^*)\}$. Let task τ_k have the longest recovery time, i.e. $R_k(m_k^*) = \max_{i=1,...n}\{R_i(m_i^*)\}$. According to equation (7), the number checkpoint of τ_k is calculated as $m_k = \lceil \frac{c_k}{SR-(r_k+q_k)} - 1 \rceil > \lceil \frac{c_k}{R_k(m_k^*)-(r_k+q_k)} - 1 \rceil = \lceil m_k^* \rceil$. This contradicts Theorem 1. ■

From Lemma 1, we can immediately set up a lower bound for SR as

$$SR \geq \max_{i=1,...n}\{\frac{c_i}{m_i^*+1}\}. \tag{9}$$

Moreover, based on the properties of ceiling(floor) functions and equation (8), we can set an upper bound and a lower bound as follows:

$$L_{upper}(\mathcal{T}, SR) = \sum_{i=1}^{n}(c_i+q_i) + \sum_{i=1}^{n}\frac{c_i}{SR-\phi_{max}}(o_i+q_i) + K*SR \tag{10a}$$

$$L_{lower}(\mathcal{T}, SR) = \sum_{i=1}^{n}(c_i-o_i) + \sum_{i=1}^{n}\frac{c_i}{SR-\phi_{min}}(o_i+q_i) + K*SR \tag{10b}$$

where $\phi_{max} = \max_{i=1,2,...,n}(r_i+q_i)$ and $\phi_{min} = \min_{i=1,2,...,n}(r_i+q_i)$.

Note that the two curves defined in equation (10a) and (10b) constrain the optimal SR as shown in Figure 1. Moreover, from equation (10a), we can readily calculate the minimum upper bound by setting

$$\frac{\partial L_{upper}(\Gamma, SR)}{\partial SR} = 0. \tag{11}$$

As can be seen from Figure 1, the optimal SR can only be located in the shaded range between $[low, high]$, beyond which L is always greater than $L1$, which is the solution of equation (11). The exact values of low and $high$ can be calculated accordingly by solving the following equation

$$L_{lower}(\Gamma, SR) = L1. \tag{12}$$

As such, equation(9) and solutions of equation (12) can be effectively used for pruning the solution space for the optimal checkpoint configurations. We summarize the procedures in Algorithm 1. It is not difficult to see that the complexity of Algorithm 1 is linear to the possible values of SR. In section V, we use experimental results to test the efficiency of our approach.

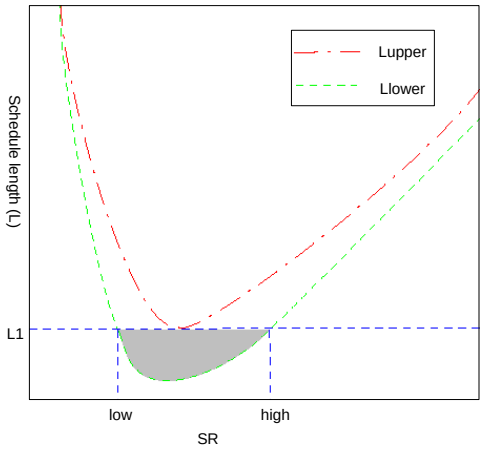


Fig. 1. Upper and lower bounds of $L(\Gamma, SR)$

Algorithm 1 OPT_CHK(Γ, K)

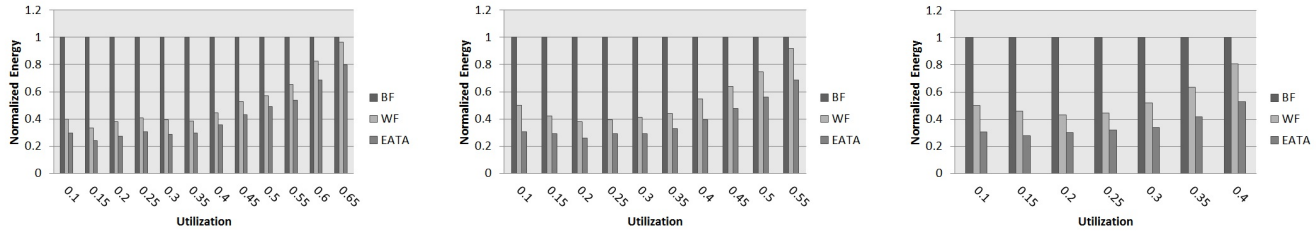
1: obtain m_i^*, for $i = 1, 2, ..., n$ according to [7]
2: $L_{min} = INF$;
3: $\mathcal{S} = \{SR_{i,k} | SR_{i,k} = R_i(k), i = 1, ..., n; k = 1, ...m_i^*\}$;
4: Prune $\mathcal{S}$ based on equation (9) and solutions of equation (12);
5: **for** $i = 1$; $i \leq sizeof(\mathcal{S})$; $i++$ **do**
6: calculate $L(\Gamma, \mathcal{S}(i))$ according to equation (8);
7: **if** $L(\Gamma, \mathcal{S}(i)) < L_{min}$ **then**
8: $L_{min} = L(\Gamma, \mathcal{S}(i))$;
9: $SR_{opt} = \mathcal{S}(i)$;
10: **end if**
11: **end for**
12: $m_i = \lceil \frac{c_i}{SR_{opt}-(r_i+q_i)} - 1 \rceil \ \forall i = 1, ...n$;
13: **return** $L_{min}, SR_{opt}, M = m_i, i = 1, ...n$

IV. Energy Aware Fault-tolerant Task Allocation

With our analysis results and algorithm to search for the optimal checkpointing scheme on a single processor, we are now ready to present our algorithm to minimize the overall energy consumption while tolerating K transient faults on multi-processor platforms.

Without fault tolerance requirement, one intuitive method is to spread the workload among multi-processor platforms as even as possible [3]. When fault tolerance requirements are taken into consideration, however, extra care must be taken since both resource reservation and DVFS compete for system slack time. Aggressively packing as many tasks as possible into one processor helps to reduce the resource reservation since the reserved resource can be shared by all tasks in the same processor. However, with too much workload stacked in one processor, it becomes difficult for a processor to scale down the processor speed. On the contrary, spreading tasks around helps to balance the workload among different processors and thus effectively reduces the processor speed. The problem is that potentially more resources need to be reserved since tasks allocated to different processors cannot share the same reserved resources. Moreover, as indicated in our analysis results before, different sets of tasks may lead to totally different optimal checkpointing results, i.e. resource-reservation schemes.

It is well known that the multi-objective task allocation problem is a NP-hard problem in the strong sense [3]. Therefore, we focus our effort on developing an effective heuristic solution for this problem. Our task allocation scheme for energy minimization with K fault tolerance guarantee is developed based on the algorithm OPT_CHK. Specifically, when allocating a new task τ_i, we assign τ_i to the processor that leads to the minimum energy consumption increase. Note that, when assigning τ_i to a processor (e.g. ψ_j), the optimal checkpoint configurations can be obtained using algorithm OPT_CHK. We assume that the re-execution of a faulty task



(a) 20 tasks on 4 processors, K = 1 (b) 40 tasks on 8 processors, K = 2 (c) 80 tasks on 16 processors, K = 4

Fig. 2. Simulation results of EATA

Algorithm 2 EATA(Γ, Ψ, K)

1: obtain m_i^*, for $i = 1,2,...,n$ according to [7]
2: $E_{total} = 0$;
3: Γ_j = NULL, for $j = 1,2,...,m$;
4: **for** $i = 1$; $i \leq n$; $i++$ **do**
5: $\Delta E = \infty$;
6: assigned = 0;
7: M_{new} = NULL;
8: **for** $j = 1$; $j \leq m$; $j++$ **do**
9: $\{L_{temp}, SR_{temp}, M_{temp}\}$ = OPT_CHK($\Gamma_j \cup \{\tau_i\}$, K);
10: $E_{temp} = E(\Gamma_i \cup \{\tau_i\})$
11: **if** $L_{temp} \leq D$ and $E_{temp} < \Delta E$ **then**
12: assigned = j;
13: $\Delta E = E_{temp}$, $M_{new} = M_{temp}$
14: **end if**
15: **end for**
16: **if** $assigned == 0$ **then**
17: return "not feasible";
18: **else**
19: $\Gamma_{assigned} \leftarrow \Gamma_{assigned} \cup \{\tau_i\}$;
20: $M = M_{new}$;
21: $E_{total} \leftarrow E_{total} + \Delta E$;
22: **end if**
23: **end for**
24: **return** $\{\Gamma_1,...,\Gamma_m\}, E_{total}, M$

is always performed at the highest speed and the checkpointing overhead is independent to the processor's running mode. Then the processor speed for ψ_j, i.e. f_j, can be determined by

$$f_j = \max\left(\frac{\sum_{\tau_i \in \Gamma_j} c_i}{D - \sum_{\tau_i \in \Gamma_j} c_i'(m_i) - K * \max_{\tau_i \in \Gamma_j} R_i(m_i)}, f_c\right) \qquad (13)$$

where f_c is the critical speed, $c_*'(*)$ and $R_*(*)$ are obtained through equations (1a) and (1b), respectively. Also, the energy consumption of processor ψ_j, i.e $E(\Gamma_j)$, can be calculated according to equation (4). Note that even though we assume the frequency of a processor can be continuously varied, we can still adopt the traditional approach [15] to deal with the scenario when only a set of discrete levels of frequencies are available. Specifically, if the desired constant frequency, i.e. f_i, is not available, we identify two available neighboring frequencies of f_i to run the task set Γ_j on ψ_j. The overall algorithm is described in Algorithm 2. It is not difficult to see that the overall complexity of Algorithm 2 is $O(n \times m \times |\mathcal{S}|)$, where $|\mathcal{S}|$ is the worst case possible values of the shared reservation block on a processor.

V. Experimental results

In this section, we study the effectiveness and efficiency of our proposed algorithms. To our best knowledge, there is no existing approach targeting the exact same problem. As a result, to study the energy saving performance of *EATA*, we compared it with two well-known fault-oblivious approaches, i.e. Best-Fit(BF) and Worst-Fit(WF). Especially, WF is a commonly used energy optimization heuristic and has been shown to be quite effective in the absence of processor faults due to its load-balancing characteristic [3]. To maintain the feasibility under the K faults for both BF and WF, the reserved resource on each processor was considered as part of the workload, and different tasks can share the reserved resource. BF(WF) allocates a task to a feasible processor with the least(most) remaining capacity. Individual optimal number of checkpoints was inserted to each task under these two heuristics. We then evaluate how many speedups that *EATA* can achieve with the techniques proposed in Section III to prune the search space of *OPT_CHK*. To evaluate the energy saving performance, we set up the simulation platforms as follows. For a fixed number (m) of processors, we varied the average utilization, i.e. $\frac{U_{total}}{m}$ from 0.1(light load) to 1 (heavy load). The utilization of each task τ_i was uniformly distributed in the range $[0.01, 0.6]$. The deadline of the application, i.e. D, was set to 100. The fault detection, checkpointing and state retrieval overhead was identically set to 0.5, 1 and 1 respectively for each task. The corresponding energy overhead was set to 0.05, 0.1 and 0.1. In addition, we set $P_{leak} = 0.1$, $C_{ef} = 1$ and $\alpha = 3$ and we assumed the existence of four normalized frequency levels given by $\{0.4, 0.6, 0.8, 1.0\}$.

Due to page limits, we only show three sets of experimental results with different numbers of tasks, processors and total transient faults. Figure 2(a) shows the energy consumption for 20 tasks and 4 processors with K=1. Each point in the figure was averaged over 1000 test cases. As we can see, the energy consumption increases when the system workload becomes heavier for all three techniques, but our approach EATA always outperforms the other two. For instance, when the processor average utilization is 0.55, 12%(46%) energy saving is achieved by EATA over WF(BF). In average, our algorithm reduces energy consumption by 11% (59%) compared to

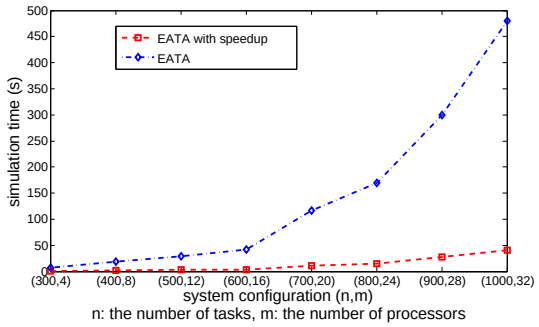


Fig. 3. Performance of two speed-up techniques

WF (BF). The energy savings are more substantial in Figure 2(b), with 8 processors and 40 tasks to tolerate maximum 2 faults, with over 16% and 62% energy savings in average compared to WF and BF, respectively. Similar results are observed for the case of 16 processors and 80 tasks with at most 4 faults as shown in Figure 2(c), where 19% and 65% energy savings are achieved over WF and BF respectively. In general, we can see that our approach can achieve better energy savings for test cases with higher system utilizations, larger numbers of tasks and processors. This is due to the fact that our approach tries to find the best combination of task allocation, checkpointing scheme and speed assignment at each step. High energy savings are achieved by reserving as less resources as possible and leaving more slacks for DVFS.

Next, we evaluated the benefits of our approach proposed in Section III. The complexity of EATA heavily depends on that of *OPT_CHK*. Therefore, the computational efficiency of *OPT_CHK* is critical to the success of EATA. To study the computational efficiency of *EATA* brought by the speedup techniques for *OPT_CHK*, we set the average utilization, i.e. $\frac{U_{total}}{m}$ to be 0.8. The utilization of each task was randomly generated to be uniformly distributed in [0.01, 0.06]. The deadline, i.e. *D* was set to 100. The timing overhead of checkpointing, fault detection and state retrieval were considered as 1% of the average task execution time. We varied the numbers of tasks and processors and recorded the results in Figure 3. In each step, we increase the number of tasks by 100 and the number of processors by 4. As we can see, as the system size grows, the time consumed by both simulations increase. However, our approach proposed in section III can easily achieve a speed up of at least 10X. As the number of tasks and processors increases, the efficiency of the two speed-up technique becomes more prominent and make the algorithm EATA efficiently scalable.

VI. Conclusion

As IC technology continues its evolution into the deep submicron domain, the exponentially increased energy consumption and the deteriorated reliability have become serious concerns in computer system design. In this paper, we study the energy minimization problem for a real-time application on a multi-processor platform that can tolerate *K* transient faults using the checkpointing method. We first develop an efficient method to determine the checkpointing scheme that can minimize the worst case response time for a task set that shares the reserved resources for fault recovery on a single processor. We then present a task assignment algorithm to minimize the overall energy while guaranteeing the fault-tolerance capability. Our experimental results also demonstrate the effectiveness and efficiency of our proposed approach.

Acknowledgement

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, and CNS-1018108.

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] 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.

[3] 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.

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

[5] L. Huang, F. Yuan, and Q. Xu, "On task allocation and scheduling for lifetime extension of platform-based mpsoc designs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 12, pp. 2088–2099, Dec. 2011.

[6] J. Srinivasan, A. S.V., B. P., R. J., and C.-K. Hu, "Ramp: A model for reliability aware microprocessor design," *IBM Research Report, RC23048*, 2003.

[7] 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.

[8] 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.

[9] 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

[10] 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.

[11] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, K. A. Yelick, M. J. Demmel, W. Plishker, J. Shalf, S. Williams, and K. Yelick, "The landscape of parallel computing research: A view from berkeley," TECHNICAL REPORT, UC BERKELEY, Tech. Rep., 2006.

[12] 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.

[13] X. Qi, D. Zhu, and H. Aydin, "Global reliability-aware power management for multiprocessor real-time systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*, Aug., pp. 183–192.

[14] 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.

[15] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the 1998 international symposium on Low power electronics and design*, ser. ISLPED '98. New York, NY, USA: ACM, 1998, pp. 197–202. [Online]. Available: http://doi.acm.org/10.1145/280756.280894