2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications

Enhanced Fault-Tolerant Fixed-Priority Scheduling of Hard Real-Time Tasks on Multi-Core Platforms

Qiushi Han Tianyi Wang Gang Quan
Electrical and Computer Engineering Department
Florida International University
Miami, FL, 33174
{qhan001, twang004, gaquan}@fiu.edu

***Abstract*—In this paper, we study the problem of partitioned scheduling of periodic real-time tasks with the capability of tolerating transient faults on multi-core platforms under Rate Monotonic Scheduling (RMS) policy. In our approach, we exploit the implicit relations among periods and recovery costs among tasks and develop a novel metric, called "compatibility index", to quantify how "compatible" a task set is when they are allocated on the same core. We theoretically analyze its properties for improving the system schedulability. Based on this metric, we propose two task partitioning schemes to partition hard real-time tasks with fault-tolerance requirements on multi-core platforms. Simulation results demonstrate that our proposed approaches can significantly enhance the performance of existing techniques.**

***Keywords*—fault-tolerance, task partition, rate monotonic scheduling, real-time**

I. Introduction

To keep pace with the demands for increasing processor performance, silicon vendors no longer concentrate wholly on increasing the clock frequency of a single-core platform, as this approach leads to excessive power consumption and heat dissipation [1]. Instead, multi-core platforms have attracted more attentions and become mainstream in the industrial market. Since 2007, many chip manufactures, such as AMD and Intel, have been releasing their new multi-core chips into market with increased number of cores, e.g Intel Xeon Series [2]. It is shown in [3] that computer chips with hundreds of even thousands of cores are being tested or produced. Conceivably, multi-core platform will be the primary choice for real-time system design in the future, and as a result, many research efforts have been made to address real-time constraints on multi-core platforms [4], [5], [6], [7].

Multi-core real-time scheduling can be broadly classified into two categories, namely *Partitioned scheduling* and *Global scheduling* [4]. In partitioned multi-core scheduling, each task is allocated to a core and all of its jobs have to be executed on that core. On the other hand, for global scheduling, the jobs of a task can possibly be executed on any other cores. In this paper, we focus on the partitioned approach due to its low overhead and the reason that the well-known scheduling methods for single-core platforms, e.g. the rate monotonic scheduling (RMS) scheme, [8] can be readily applied.

While we enjoy the benefits of high performance computing in the era of multi-core processors, the reliability of modern computing system has degraded substantially. The proliferation of multi-core processor production is made possible thanks to the rapid advancements of semi-conductor technologies, in particular, the transistor miniaturization and mass transistor integration. However, their adverse effects on system reliability have increasingly become a concern and need to be addressed appropriately.

First, the dramatically decreased feature size of transistors has elevated the radiation-induced fault rates up to several orders of magnitude [9]. Second, the mass integration of transistors increases the power density and temperature on chip considerably, which leads to the acceleration of wear-out of a chip [10], [11]. As many real-time systems are safety-critical in nature, e.g. automobiles and airplanes, the reliability of such systems has been raised to a first-class design requirement. Traditional multi-core scheduling without explicitly considering run-time failures is no longer sufficient any more.

In this paper, we are interested in studying the problem of partitioning hard real-time tasks on multi-core platforms while guaranteeing the fault-tolerance requirements under the RMS scheme. We choose RMS as our underlying scheduling method as it is optimal for single-core scenario [8] and the primary choice for industrial applications today. In considering the task partitioning, one common method is to transform this problem to a standard bin-packing problem, and popular heuristics such as First-fit (FF), Best-fit (FF), and Worst-fit (WF) can be readily applied. In these approaches, tasks are allocated based solely on their utilizations. Fan et al. [12], [7] takes the periods relationship among tasks into considerations when making partitioning decisions and can significantly improve system utilization. However, as we show later in the paper, allocating harmonic tasks together may not always be a good choice in face of fault-tolerance requirements. As such, we develop a novel metric to quantify how "compatible" tasks are when they are allocated to the same processing core. Based on this metric, we develop two partitioning approaches that can take advantages of periods and recovery costs of tasks to attain higher system utilization. We further extend our partition approaches to systems with multiple checkpointing schemes. According to our simulation results, our approach can significantly outperform other related approaches. In what follows, we first discuss some existing works that are related

978-1-4673-7855-0/15 $31.00 © 2015 IEEE
DOI 10.1109/RTCSA.2015.22

to our research.

The rest of the paper is organized as follows. We first discuss the existing works that are related to our research in Section II. Section III introduces the preliminaries and notations used throughout this paper. Section IV studies the schedulability of rate-monotonic fault-tolerant tasks. In section V, we propose a novel metric to quantify how "compatible" a task set is in terms of system schedulability. Two task partitioning techniques are presented in Section VI. In Section VII, we extend our partitioning algorithms to incorporate the checkpointing feature to further enhance system schedulability. Simulation studies are conducted in Section VIII. Finally, we conclude our paper in Section IX.

II. Related work

Searching for the optimal task partitioning is essentially a design space exploration problem. The key to the success of a partitioned algorithm is to efficiently and accurately evaluate a design alternative, i.e. a task allocation. A task allocation is considered to be feasible if the timing constraints of all tasks can be guaranteed under the influence of faults. To determine if such a condition can be met, a number of fault-tolerant schedulability analysis techniques are proposed.

Pandya et al. [13] developed an utilization bound of 0.5 for hard real-time tasks scheduled under RMS policy on single-core platforms when at most one failure can occur. It is a sufficient condition to test the schedulability of tasks under the influence of a failure. However, 0.5 is far from being a tight bound of the system utilization, and the constraint that the system can only experience one failure is too stringent. Burns et al. [14] extended the traditional Worst Case Response Time Analysis (WCRT) for fixed-priority tasks to incorporate run-time faults. A necessary and sufficient schedulability test was derived. However, they considered failure as a special sporadic task that each failure is separated by a minimum inter-arrival time. This assumption severely limits the applicability of this approach. Zhang et al. [15] relaxed the constraints regarding the fault pattern and proposed an exact timing analysis based on the WCRT for fixed-priority tasks subject to a maximum number of faults. While the exact timing analysis can achieve high accuracy in feasibility analysis, it is computationally prohibitive and is not suitable for design space explorations. These aforementioned approaches are either too computationally expensive or too pessimistic and are deemed to be unsuitable for design space explorations.

Task partition is well-known as a NP-complete problem [4]. Therefore, developing effective and efficient heuristics to achieve sub-optimal results is usually a practical approach. A plethora of papers have been published on partitioned multi-core scheduling of fixed-priority periodic tasks.

Andersson et al. [16] showed that the maximum utilization a fixed-priority multi-core scheduling can achieve on each core is no more than 50%. AlEnawy et al. [6] studied the schedulability and energy performance for periodic tasks scheduled on a homogeneous multiprocessor platform with different allocation methods, e.g. Best-Fit, Worst-Fit and First-Fit, and speed assignments. They concluded that the overall performance of Best-Fit dominates the other well-known heuristics in terms of schedulability. Task partitioning under multiple resource constraints was studied in [17], and efficient heuristics were proposed to improve system schedulability considering resource assignment. Fan [7] et al. exploited the fact that harmonic tasks (tasks that have periods being integer multiples of each other) can achieve higher system utilization and developed a metric to quantify how harmonic a task set is. Based on this metric, they proposed an partition approach by grouping the most harmonic tasks together and showed that it can significantly outperform traditional bin-packing approaches. Unfortunately, these approaches are fault-oblivious.

There are only a few papers which are closely related to our research. Pop et al. [18] investigated the problem of guaranteeing the schedulability and reliability of tasks with precedence constraints on a heterogenous multi-core platform. They used the combination of checkpointing and active replication to deal with the fault-tolerance problem. A meta-heuristic approach, i.e. Tabu search was adopted to search for the best task allocation and fault-tolerance policy for each task. However, this approach is computational inhibitive and it is not scalable with increasing number of tasks and cores. Guo et al. [19] developed a standby-sparing technique to tolerate faults by replicating task schedules on spare cores. This approach requires extra processing cores and the aim is to save energy rather than to improve system schedulability. Han et al. [20] presented a method to determine the number of checkpoints for each real-time task when making the tradeoffs between checkpointing overhead (including both timing and energy) and recovery cost. Our research focuses on improving the system feasibility by judiciously partitioning tasks, and later in this paper, we discuss how this approach can be integrated into our approach for tasks with multiple checkpoints.

In what follows, we first introduce some preliminaries crucial to this paper and use an example to motivate our research. Then we formulate our research problem formally.

III. Preliminaries

In this section, we introduce some basic concepts and notations used throughout this paper.

A. Application and system model

The application under investigation is modeled as a periodic task set Γ with n tasks, i.e. $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task τ_i is associated with a tuple (C_i, D_i, T_i) where C_i, D_i and T_i denote the worst case execution time, relative deadline and minimum inter-arrival time (period) of τ_i, respectively. We consider implicit-deadline tasks, i.e. $D = T$ in this paper. Each task can release an infinite number of jobs. We assume that Γ is sorted by non-decreasing period order, i.e. for $\forall \tau_i, \tau_j \in \Gamma$, $T_i \leq T_j$ if $i < j$. We use $u_i = \frac{C_i}{T_i}$ to denote the utilization of task τ_i. The total utilization of task set Γ is represented by

$$U(\Gamma) = \sum_{\tau_i \in \Gamma} \frac{C_i}{T_i}. \quad (1)$$

We consider a multi-core platform that consists of M homogenous preemptive cores, i.e. $\mathcal{P} = \{P_1, P_2, ..., P_M\}$. The system average utilization is denoted as

$$U_{avg} = \frac{U(\Gamma)}{M} \tag{2}$$

Partitioned scheduling is adopted in this paper and the tasks assigned to each core are scheduled according to RMS. We let Γ_{P_j} denote the set of tasks assigned to core P_j.

B. Fault-tolerance/reliability requirement

In this paper, we focus our efforts on tolerating transient/soft errors that do not cause permanent damage to a processing core. Transient/soft errors are the predominant type of failures in modern computing systems [21]. In particular, we consider that the system is subject to a maximum of K faults during one operation cycle of the system (its length is the least common multiple (LCM) of all task periods and is denoted by L). We adopt this K-fault model for the following three reasons: 1) it is a widely accepted fault model and well studied in the literature [22], [23], [**?**], [15]; 2) it is more general in a sense that it does not assume any particular fault pattern; 3) it can be readily translated to the statistical reliability requirement, as explained in [23].

To deal with the fault, we first consider the option to re-execute the entire task once a fault is detected at the end of the execution. Then the worst case recovery time for τ_i under a single failure is denoted as

$$F_i = \max_{j=1,...,i} C_j. \tag{3}$$

The max() function is used since a lower priority job of task τ_i can be preempted by job(s) from any higher priority task $\tau_j, j \in [1, i-1]$. Therefore, the worst-case delay it may suffer due to a failure is the longest re-execution of a job among all higher-priority tasks and τ_i itself.

C. Problem formulation

With the system models defined above, we formally formulate our research problem as follows.

Problem 1. *Given a task set Γ scheduled under RMS on a multi-core platform $\mathcal{P}$, develop efficient and effective task partitioning methods such that all tasks in Γ can meet their deadlines when no more than K faults occur.*

D. Motivation example

Problem 1 is a traditional NP-complete problem even without the fault-tolerance requirements. To understand the unique challenges of Problem 1, we first present a motivate example.

Consider a 2-core platform and a task set consists of 5 tasks, the task parameters are shown in Table I. Assume that in order to satisfy the reliability requirement of the task set, the task set needs to tolerate 1 fault in the worst case scenario. It is a well-known fact that, when real-time tasks are scheduled according to RMS, allocating the harmonic tasks to the same processor can achieve the maximum utilization of 1. As shown in [7], algorithm HAPS takes advantage of this fact and, by grouping harmonic tasks together, it can significantly improve the system schedulability. Note that the sub task set $\{\tau_3, \tau_4, \tau_5\}$ and $\{\tau_1, \tau_2\}$ are perfect harmonic. Therefore, one intuitive approach is to assign $\{\tau_3, \tau_4, \tau_5\}$ to one core and $\{\tau_1, \tau_2\}$ to a different core, as shown in Figure 1.

TABLE I
EXAMPLE I: A TASK SET WITH FIVE REAL-TIME PERIODIC TASKS ARRANGED IN DECREASING PRIORITY ON A 2-CORE PROCESSOR WITH K =1

τ_i	C_i	T_i	u_i
1	3.5	10	0.35
2	3.1	10	0.31
3	6	19	0.32
4	3	19	0.16
5	4	19	0.21

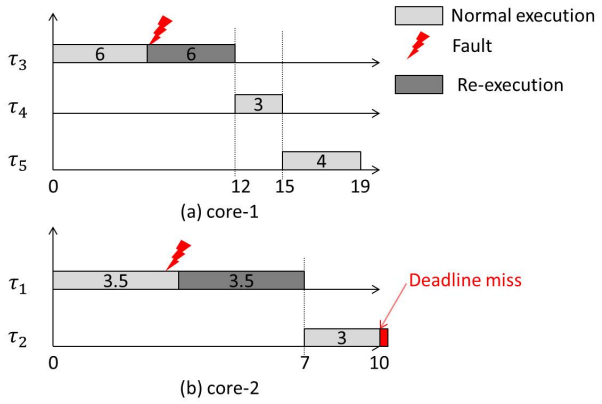

Fig. 1. Task partition based on HAPS. Task τ_2 misses deadline under the worst case.

As shown in Figure 1(a), processing core 1 is fully utilized when the worst case, i.e. a fault strikes τ_3, occurs. Still, all tasks can meet their deadlines. However, as shown in Figure 1(b), if a fault strikes τ_1, τ_2 will miss its deadline.

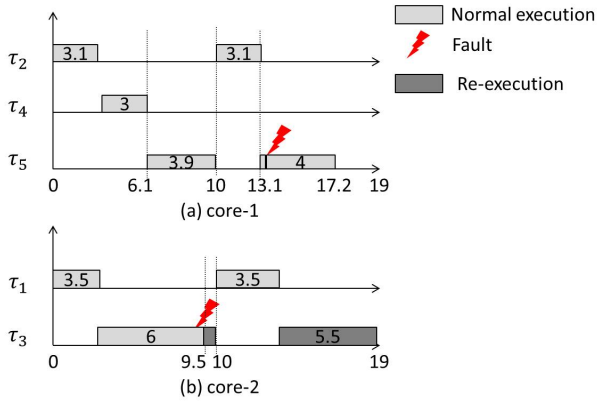

Fig. 2. An alternative partition, all tasks are schedulable under the worst case.

An alternative partition is to assign tasks τ_1 and τ_3 to core-2 and the rest to core-1. As shown in Figure 2, even though τ_1 and τ_3 are not entirely harmonic, neither are τ_2, τ_4 and τ_5, it can be readily verified that with this partition, all tasks can meet their deadlines under the worst case as shown in Figure 2.

The above motivation example implies that, while harmonic task sets can achieve high system utilization, making partition decisions without considering fault-tolerance requirements may undermine the schedulability of a system. In what follows, we first conduct the feasibility analysis for real-time tasks with fault-tolerance requirements and see how we can enhance the real-time system schedulability by partitioning tasks appropriately.

IV. Fault-tolerant schedulability analysis for fixed-priority task sets

While it is a common sense that harmonic task sets can better utilize processor resource, as indicated in our motivation example above, grouping harmonic tasks together does not necessarily always lead to the best solution when fault-tolerance requirement is considered. To uncover the fundamental reason for this problem, we start with the feasibility analysis for tasks with fault-tolerance requirements since the key to a successful partitioning algorithm is to evaluate a partition result effectively in a efficient manner. One advantage of partitioning algorithms over global algorithms is that well-established single-core scheduling methods (e.g. RMS) can be readily adopted in partitioned settings. In what follows, we first introduce an existing method with pseudo-polynomial running timing for determining the schedulability of RMS-scheduled real-time tasks under the influence of transient faults. Then, we present a much more efficient schedulability test by exploiting the implicit harmonic relations between task periods.

For a task set Γ with K-fault-tolerance requirement, its feasibility can be determined using the traditional exact worst case timing analysis. Specifically, the following theorem is established in [22] for this purpose.

Theorem 1. *A task $\tau_i \in \Gamma$ is schedulable if and only if there exists a scheduling point $t \in [0, T_i]$, such that*

$$C_i + \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil \cdot C_j + K \cdot F_i \leq t, \tag{4}$$

where t is defined in the set $\{t_x | t_x = n \cdot T_j, n \in [1, \lfloor \frac{T_i}{T_j} \rfloor], j \in [1, i]\}$. Therefore, a task set Γ is schedulable if $\forall \tau_i, \tau_i \in \Gamma$ is schedulable.

Note that, while the exact worst case response time analysis in Theorem 1 helps to identify the exact schedulability of a *given* real-time task set, it does not provide any guidance, except for being applied in traditional heuristics such as bin-packing methods, on which tasks should be grouped together and assigned to the same core to improve the system schedulability. In addition, since the complexity of this test is pseudo-polynomial, it is not suitable for design space explorations when designing large and complex systems.

As a harmonic task set is schedulable if its total utilization is no more than 1 [24], the computational complexity for schedulability checking is greatly reduced. Similarly, for a harmonic task set with K-fault-tolerance requirement, the feasibility condition can also be greatly simplified as shown in the following lemma and theorem.

Lemma 1. *Given a harmonic task set Γ, a task τ_i is schedulable with no more than K fault occurrences if and only if the following condition is met,*

$$C_i + \sum_{j=1}^{i-1} \lceil \frac{T_i}{T_j} \rceil \cdot C_j + K \cdot F_i \leq T_i \tag{5}$$

Proof: We prove this lemma in two steps.

Sufficient condition:if equation (5) is met, then τ_i must be schedulable according to Theorem 1.

Necessary condition: if task τ_i is schedulable, there must exist a scheduling point t such that equation (4) is satisfied. Furthermore, according to the definition of scheduling points, t must be some arrival time(s) of higher priority task(s). As a result, T_i must be some integer multiple of t, we denote it by $T_i = a \cdot t$ where a is an arbitrary integer. Moreover, T_i can be divided by any period $T_j, j \in [1, i]$. Therefore, we have the following property,

$$\begin{aligned} C_i + \sum_{j=1}^{i-1} \lceil \frac{T_i}{T_j} \rceil \cdot C_j + K \cdot F_i &= C_i + \sum_{j=1}^{i-1} \frac{T_i}{T_j} \cdot C_j + K \cdot F_i \\ &\leq a \cdot C_i + a \cdot \sum_{j=1}^{i-1} \lceil \frac{t}{T_j} \rceil \cdot C_j + a \cdot K \cdot F_i \\ &\leq a \cdot t = T_i. \end{aligned}$$

In other words, if task τ_i is feasible, then equation (5) must be met. Thus far, this lemma is proved. ■

Theorem 2. *A harmonic task set Γ is schedulable with no more than K fault occurrences if and only if the following condition holds,*

$$\max_{i=1,\ldots,n} (U_{eff,i} + UF_i) \leq 1 \tag{6}$$

*where $U_{eff,i} = \sum_{j=1}^{i} u_j$ and $UF_i = K \cdot \frac{F_i}{T_i}$ denotes the **effective utilization** and the **recovery utilization** of task τ_i, respectively.*

The proof of this Theorem 2 can be readily obtained from Lemma 1 and is therefore omitted. Thus far, we develop an efficient and effective schedulability tests for harmonic task sets. However, it is a very stringent constraint for tasks to be strictly harmonic. Therefore, we relax this constraint and extend our method to more general task sets in this section. For a given task set Γ, a corresponding transformed harmonic task set Γ'_i is defined as follows.

Definition 1. *Given a task set $\Gamma = \{\tau_1, \ldots, \tau_i, \ldots, \tau_n\}$ where $\tau_i = (C_i, T_i)$ is the **base task**, then*

$$\Gamma'_i = \{\tau'_{1,i}, \ldots, \tau_i, \ldots, \tau'_{n,i}\} \tag{7}$$

*is a transformed **harmonic** task set with $\tau'_{j,i} = (C'_{j,i}, T'_{j,i}), \forall j \neq i$ where $C'_{j,i} = C_j$ and $T'_{j,i}$ is the largest possible period that is*

less than T_j and can form a harmonic relationship with all the other task periods. For two arbitrary tasks, i.e. $\tau'_{j,i}$ and $\tau'_{k,i}$ and $j<k$, the period $T'_{j,i}$ divides $T'_{k,i}$ (denoted as $T'_{j,i}|T'_{k,i}$). The utilization of task $\tau'_{j,i}$ is denoted as $u'_{j,i}=\frac{C'_{j,i}}{T'_{j,i}}$.

In this paper, we adopt the DCT algorithm [24] to construct harmonic task sets from an arbitrary task set Γ. Note that our algorithms proposed in this paper are not restricted to any transformation method. To make this paper self-contained, we reiterate the steps of the DCT algorithm as below,

- sort task set Γ with non-decreasing period;
- using each $\tau_i \in \Gamma$, transform Γ to Γ'_i

$$T'_{j,i}=\begin{cases} T'_{j+1,i}/(\lceil T'_{j+1,i}/T_j \rceil), & \text{if} j<i \\ T_j, & \text{if} j=i \\ T'_{j-1,i}\cdot \lfloor T_j/T'_{j-1,i} \rfloor, & \text{if} j>i \end{cases} \tag{8}$$

Under DCT, task execution times and task orderings remain the same, but task periods become smaller. Therefore we can determine the schedulability of task set Γ from that of its transformed harmonic task sets, which is formulated in the theorem below.

Theorem 3. *Given a task set Γ with n tasks and its transformed harmonic task set through DCT, i.e. $\Gamma'_i, 1\le i\le n$, if there exists i, such that Γ'_i is schedulable with the maximum of K fault-occurrences, then Γ is also schedulable with the maximum of K fault-occurrences.*

Theorem 3 can be readily proved noting that periods for tasks in the transformed task set is no larger than that in the original task set. A straightforward implementation of Theorem 2 has a computational complexity of $O(n)$. Therefore, the computational complexity to check the schedulability based on Theorem 3 is $O(n^2)$, which is usually much smaller than that of Theorem 1.

V. Compatibility index and its properties

The feasibility analysis results presented above clearly reveal the reason why allocating harmonic tasks to the same core can in fact lead to inferior solutions. Note that, from equation (6), the schedulability of a harmonic task set with fault-tolerance requirement depends not only on the task set utilization itself but also recovery utilization as well. Therefore, to partition tasks with fault-tolerance requirements, we need to consider not only if tasks are harmonic but also if they are "compatible". To this end, we design a new metric to quantify the *compatibility* of tasks to be allocated to the same processing core.

Definition 2. *Given an arbitrary task set Γ and its transformed harmonic task set Γ'_i as defined in Definition 1, then the* **compatibility index** *of task τ_j, $\tau_j \in \Gamma$ measured under configuration Γ'_i is defined as*

$$COMP(\tau_j,\Gamma'_i)=\Delta H_{j,i}+\Delta EF_{j,i}, \tag{9}$$

where $\Delta H_{j,i}=u'_{j,i}-u_j$ and $\Delta EF_{j,i}=K\cdot\frac{F_j-C_j}{T'_{j,i}}$ denote the **harmonic distance** *of task τ_j to its counterpart in Γ'_i and the* **extra recovery utilization** *task τ_j has to endure considering all the higher-priority tasks in Γ.*

In what follows, we study the impacts of each factor exclusively. A *harmonic distance* $\Delta H_{j,i}$ [7] quantifies how much utilization of a task τ_j needs to be increased in order to transform a task set Γ to a harmonic task set Γ'_i. In other words, it measures how harmonic the task τ_j is with respect to all the remaining tasks in Γ. The less the harmonic distance for each task is, the better the system schedulability is. We formally formulate this property in the following theorem.

Theorem 4. *Consider a task set Γ and its two transformed harmonic task set Γ'_p and Γ'_q (using τ_p and τ_q as base tasks, respectively), where $\Delta EF_{i,p}=\Delta EF_{i,q}$ and $\Delta H_{i,p}\le\Delta H_{i,q}$, for $\forall\tau_i$. The task set Γ'_p must be schedulable if Γ'_q is schedulable.*

Proof: If the task set Γ'_q is schedulable, then for each task $\tau'_{i,q}\in\Gamma'_q$, the following condition must be satisfied according to Theorem 2,

$$\frac{C_i}{T'_{i,q}}+\sum_{j=1}^{i-1}\frac{C_j}{T'_{j,q}}+K\cdot\frac{F_i}{T'_{i,q}}\le 1. \tag{10}$$

Since $\Delta EF_{i,p}=\Delta EF_{i,q}$ and $\Delta H_{i,p}\le\Delta H_{i,q}$, $\forall\tau_i$, we have

$$u'_{i,p}\le u'_{i,q}\Longleftrightarrow T'_{i,p}\ge T'_{i,q}. \tag{11}$$

Then equation (10) can also be satisfied with a larger period $T'_{i,p}$, which means that task set Γ'_p is schedulable. ■

The *extra recovery utilization* represents the extra recovery overheads that task τ_j needs to tolerate when it is subject to preemptions from all higher-priority tasks in Γ. A task is more likely to be schedulable when there is less extra recovery overhead. Therefore, with less extra recovery overhead for each task, the system can potentially achieve better schedulability. We summarize this property in Theorem 5.

Theorem 5. *Given two harmonic task sets Γ_1 and Γ_2 with identical number of tasks, let $\tau_{j,1}$ and $\tau_{j,2}$ be their j_{th} task in Γ_1 and Γ_2, respectively. Assume that $\forall j$, $u_{j,1}=u_{j,2}$ and $\Delta EF_{j,1}\le\Delta EF_{j,2}$. If Γ_2 is schedulable, then Γ_1 must also be schedulable.*

Proof: If the task set Γ_2 is schedulable, then for each task $\tau_{j,2}\in\Gamma_2$, the following condition must be satisfied according to Theorem 2,

$$u_{j,2}+\sum_{i=1}^{j-1}u_{i,2}+K\cdot\frac{F_{j,2}}{T_{j,2}}\le 1. \tag{12}$$

Since $u_{j,1}=u_{j,2}$, and $\Delta EF_{j,1}\le\Delta EF_{j,2}$, we have

$$\begin{aligned}\Delta EF_{j,1}-\Delta EF_{j,2}&=K\cdot\frac{F_{j,1}-C_{j,1}}{T_{j,1}}\\&\quad-K\cdot\frac{F_{j,2}-C_{j,2}}{T_{j,2}}\\&=K\cdot\frac{F_{j,1}}{T_{j,1}}-K\cdot\frac{F_{j,2}}{T_{j,2}}-K\cdot(u_{j,1}-u_{j,2})\\&=K\cdot\frac{F_{j,1}}{T_{j,1}}-K\cdot\frac{F_{j,2}}{T_{j,2}}\le 0.\end{aligned}$$

Therefore, for each task $\tau_{j,1} \in \Gamma_1$, the following condition is met,

$$u_{j,1} + \sum_{i=1}^{j-1} u_{i,1} + K \cdot \frac{F_{j,1}}{T_{j,1}} \leq 1. \quad (13)$$

In other words, Γ_1 is schedulable. Thus far, this theorem is proved. ■

The above two theorems show that both of the factors in compatibility index, i.e. harmonic distance and extra recovery utilization, can play significant roles in reflecting the schedulability of a task set. We consider both factors equally important, and we define the compatibility index of a task set as follows.

Definition 3. *The* **compatibility index** *of a task set* Γ *consisting of n tasks is defined as*

$$COMPTS(\Gamma) = \min_{i=1,\ldots,n} \sum_{j=1}^{n} COMP(\tau_j, \Gamma'_i), \quad (14)$$

where $COMP(\tau_j, \Gamma'_i)$ *is formulated in Definition 2. The less the value* $COMPTS(\Gamma)$ *is, the more compatible* Γ *is.*

This metric measures not only the harmonicity of a task set but also the fault-compatibility among all tasks. Let us use the example in Section III-D to illustrate the efficacy of this metric. We have $COMPTS(\{\tau_1, \tau_2\}) = 0.04$ where $COMPTS(\{\tau_1, \tau_3\}) = 0.018$. Then τ_1 and τ_3 are deemed to be more compatible, though their periods are not strictly harmonic.

VI. Fault-tolerant task partitioning

In light of Section V, task sets with lower "compatibility index" (more compatible) are more likely to be schedulable under the influence of transient faults.

We are now ready to present our multi-core partition algorithm "Compatibility Aware Task Partition (CATP)" in Algorithm 1.

Algorithm 1 *CATP*(Γ, $\mathcal{P}$, K)

Require:
Γ - task set with n tasks, $\mathcal{P}$ - multi-core platform with m cores, K- number of faults.

1: sort tasks in non-increasing utilization order;
2: **for** $i = 1$ to n **do**
3: $p\_index = 0$; $c\_min = +\infty$;
4: **for** $j = 1$ to M **do**
5: **if** τ_i can be assigned to P_j **then**
6: **if** $COMPTS(\{\tau_i, \Gamma_{P_j}\}) < c\_min$ **then** $p\_index = j$
7: **end if**
8: **end for**
9: **if** $core\_index == 0$ **then** return"FAILURE";
10: **else** $\tau_i \rightarrow P_{p\_index}$;
11: **end for**
12: **return** "SUCCESS" and partition results;

The task set Γ is first arranged in non-increasing utilization fashion (Line 1). The algorithm allocates one task at a time. Within each step, it tentatively assigns the current task to each core and measures how compatible the task is with the existing tasks on the core (Lines 3-8). If the current task can not be allocated to any of the cores, the algorithm reports that a feasible allocation can not be found (Line 9). Otherwise, the task is assigned to the core with the minimum compatibility value (Line 10). The partition result is returned if all tasks can be successfully allocated.

Algorithm 1 is simple yet effective. It is a greedy approach as it intends to find the best candidate core in each step when assigning a task. However, the limitation of assigning task one at time comes at the ignorance of the fact that a task to be assigned in later stage may not be packed with the most compatible tasks due to schedulability constraints.

Let us revisit Example 1. By running Algorithm CATP, a feasible partition can be found with tasks τ_1, τ_4, τ_5 assigned to core-1 and tasks τ_2 and τ_3 to core-2. If we add another task τ_6 with parameters $C_6 = 2.6$ and $T_6 = 19$, it can be verified that τ_6 can not be allocated to neither of the core, which results in a FAILURE. With a careful examination, we can see that the most compatible tasks are τ_1 and τ_3 with a $COMPTS(\{\tau_1, \tau_3\}) = 0.018$, if we first group these two tasks together and assign them to core-1, the rest of the tasks with a $COMPTS(\{\tau_2, \tau_4, \tau_5, \tau_6\}) = 0.053$ to core-2, all tasks are schedulable under the worst case.

Next, we present our "Group-wise Compatibility Aware Task Partition (G-CATP)" method in Algorithm 2.

Algorithm 2 *G-CATP*(Γ, K)

Require:
Γ - task set with n tasks, $\mathcal{P}$ - multi-core platform with m cores, K- number of faults.

1: sort tasks in non-decreasing period order;
2: **while** isNOTempty(Γ) AND isNOTempty($\mathcal{P}$) **do**
3: $\Gamma_{opt} = \emptyset$;
4: **for** $i = 1$ to $|\Gamma|$ **do**
5: Transform Γ into Γ'_i with base task τ_i;
6: Find a subset Γ'_{sub} from task set Γ'_i (corresponding to Γ_{sub} from Γ) such that
1. Γ'_{sub} is schedulable;
2. $U(\Gamma'_{sub})$ is maximized;
3. $COMPTS(\Gamma'_{sub})$ is minimized.
7: **if** $U(\Gamma_{sub}) > U(\Gamma_{opt})$ **then** $\Gamma_{opt} = \Gamma_{sub}$;
8: **end for**
9: **if** $U(\Gamma_{opt}) == \emptyset$, **then** return "FAILURE";
10: **else** $\Gamma = \Gamma - \Gamma_{opt}$; $\Gamma_{opt} \rightarrow$ an empty core;
11: **end while**
12: **if** isNOTempty(Γ) **then** return "FAILURE";
13: **else** return "SUCCESS" and partition results

Different from Algorithm 1, in each step, Algorithm 2 assigns a group of tasks together to a core. Under each harmonic transformation, determining the most compatible subset of tasks while simultaneously guaranteeing all three conditions at Line 6 is not a trivial task. A brute-force exhaustive search is apparently computationally inhibitive and impractical. Therefore, we use the heuristic as follows. With a

given base task τ_i, we first assign τ'_i to Γ'_{sub}. Then, we scan all the remaining tasks in Γ'_i and find the task that results in the minimum increase of $COMPTS(\Gamma'_{sub})$ if it is assigned to Γ'_{sub}. We repeat the process until no more tasks can be added to Γ'_{sub}. After a group of tasks with the largest total original utilization (utilizations before harmonic transformation) are determined (Lines 3-8), they will be assigned to the first available core and removed from Γ (Line 10). The algorithm reports "SUCCESS" if all tasks can be assigned but otherwise report "FAILURE".

Next, we extend our partitioning algorithms to incorporate the checkpointing feature to further enhance system schedulability.

VII. Task set with checkpointing

Till now, we assume that an entire job is re-executed once a fault is detected. As shown in [20], checkpointing with roll-back recovery is a very efficient technique to reduce recovery overhead and improve system schedulability. To our best knowledge, there is no work that targets on improving system schedulability for fixed-priority tasks on multi-core platforms based on exploring the combination of task partitioning and checkpointing. Different task partitions can result in different checkpointing configurations. Moreover, without the knowledge of task partitioning, a predefined checkpointing scheme will most likely lead to poor schedulability performances. Therefore, it is not a trivial problem to search for the best combination of checkpointing and task allocation. In what follows, we endeavor to develop efficient and effective heuristics with the joint consideration of checkpointing and task allocation in order to maximize system schedulability. Specifically, we extend our partitioning algorithms, i.e. CATP and G-CATP, to incorporate the checkpointing scheme. We first introduce some basics on checkpointing for ease of presentation.

Under checkpoint scheme, instead of rolling back to the beginning of the execution of a job, the last saved checkpoint is retrieved and the job is executed thereafter. For a task τ_i with m_i number of checkpoints, the length of a re-execution segment is $\frac{C_i}{m_i+1}$. Therefore, the worst case recovery time for a job of τ_i is modified to

$$F_i = \max_{j=1,\ldots,i}(\frac{C_i}{m_i+1}). \tag{15}$$

Additionally, as inserting checkpoints incurs overhead, the worst case exsection time of τ_i with m_i checkpoints (its overhead is denoted by o_i) is denoted as

$$C_i(m_i) = C_i + m_i \cdot o_i. \tag{16}$$

With the new execution time and recovery for each task $\tau_j \in \Gamma$, the two factors, i.e. harmonic distance and extra recovery overhead, in the compatibility index defined in Definition 2 are modified accordingly to

$$\Delta H_{j,i} = \frac{C_j(m_j)}{T'_{j,i}} - \frac{C_j(m_j)}{T_{j,i}} \tag{17}$$

and

$$\Delta EF_{j,i} = K \cdot \frac{F_j - \frac{C_j}{m_j+1}}{T'_{j,i}}, \tag{18}$$

respectively.

To find a feasible checkpoint scheme for a set of fixed-priority tasks, we adopt the method ECHK in [20]. ECHK iteratively inserts checkpoints to the task which has a higher or equal priority than the first unschedulable task and the largest recovery overhead. The algorithm ECHK returns either the checkpointing configuration if a feasible one can be found or a failure status indicating that the task set is unschedulable.

Then, algorithm CATP can be directly extended to integrate the checkpointing scheme. Tasks are assigned one at a time, and a task-to-core mapping is considered feasible only when there exists a feasible checkpointing configuration for the task set (including the to-be-assigned task) on that core. Given a checkpointing scheme, the corresponding updated task-set compatibility index can be readily obtained. Among all the feasible cores (mappings), the one with the least task-set compatibility index is selected. This process is repeated until all tasks are assigned or no core can accommodate any more tasks. We denote this algorithm as CATP-CHK. CATP-CHK essentially utilizes "compatibility index" to evaluate the fitness of task allocation and checkpointing, as "compatibility index" has been shown to be very effective in reflecting system schedulability.

Similarly, we modify algorithm G-CATP to incorporate the checkpointing scheme. Different from CATP, G-CATP tries to find the most compatible group of tasks with the largest total utilization in each step. However, this problem with the integration of checkpointing becomes more complicated, as different checkpointing configurations can lead to large variations of the "compatibility index" of a task set. Therefore, developing efficient and effective heuristics to solve this problem is practical. Following the same procedures in Algorithm 2, we search the most compatible group of tasks under each harmonic transformation, and the group is initialized with only the base task, i.e. the task used for harmonic transformation. The rationale of choosing the base task as the first task in the group is that the group will have the least amount of task-set "compatibility index", i.e. 0, at the beginning. Then, we add tasks to the group one at a time. A task can be combined into the group only when ECHK returns a feasible checkpointing configuration. Among all the tasks that can be assigned to the group, the task which leads to the minimum increase of "compatibility index" is selected. This process repeats until no more tasks can be added to the group without jeopardizing its schedulability. We denote this algorithm as G-CATP-CHK.

In the following section, we use extensive simulations to demonstrate the effectiveness of our proposed algorithms.

VIII. Simulation results

In this section, we use simulations to evaluate the performance of our proposed partition algorithms. Specifically, we first study the impacts of different parameters, i.e. the number of tasks and cores, system average utilization, and the maximum number of faults on system schedulability. Then, we investigate the effectiveness of incorporating checkpointing scheme to improve system schedulability.

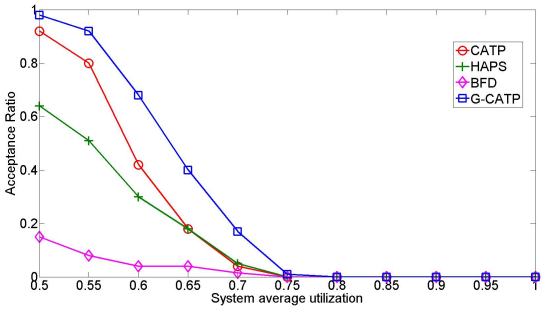

Fig. 3. 32 tasks on 4-core platform, K=2.

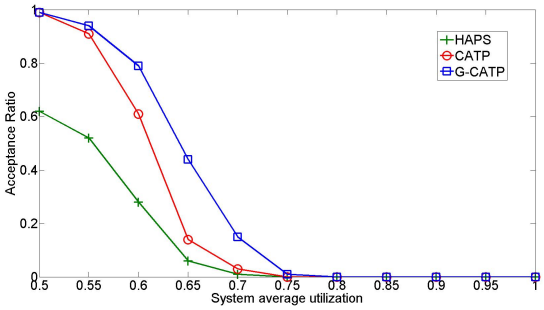

Fig. 4. 64 tasks on 8-core platform, K=2.

As explained in [4], a widely adopted metric to evaluate a partition algorithm is the *acceptance ratio*. First, a number of synthetic task sets are generated, and the acceptance ratio is calculated as the number of successfully partitioned task sets divided by the total number of task sets as shown in equation (19).

$$\text{acceptance ratio} = \frac{\text{The number of schedulable task sets}}{\text{The total number of tasks sets}}, \tag{19}$$

Four algorithms are evaluated in this section, namely, G-CATP, CATP, HAPS and Best-Fit Decreasing (BFD). The first two are proposed and explained in Section VI. The details of the algorithm HAPS is elaborated in [7], where the algorithm uses the *harmonic distance* (equation (9,14) without considering the term *extra recovery utilizaiton*) as the guideline when tasks are partitioned. *HAPS* has been shown to be quite effective in improving system utilization for fault-oblivious systems. BFD orders the tasks in a non-increasing utilization fashion and assigns a task to the core with the minimum remaining utilization.

The experimental setup is listed in details as follows. Task sets were generated according to the algorithm UUniFast in [25]. UUniFast is an algorithm designed for single-core platform. In order to generate a task set with a total utilization, i.e. U_{total} larger than 1, we need to execute this algorithm M times with the target utilization of $\frac{U_{total}}{M}$ during each run. We discarded the test cases where an individual task utilization exceeds $\frac{1}{K+1}$ since a task with a larger utilization than this can not even be scheduled by itself under the worst case scenario, i.e. K faults occur. The period of each task τ_i, i.e. T_i was randomly generated in the range of $[10, 1000]$, and its execution time C_i was calculated as $u_i \cdot T_i$.

A. Experiment 1, acceptance ratio vs. system average utilization.

In this set of experiments, we study the relationships between system average utilization and acceptance ratio. We fixed the the number of tasks and varied the system average utilization in the range $[0.5, 1]$ with a step of 0.05. We considered a 4-core platform with 32 real-time tasks. A maximum number of 2 faults was assumed in order to satisfy the system reliability constraint. For each utilization value, we generated 1000 task sets and the acceptance ratio was recorded in Figure 3. As we can see, the acceptance ratios for all four algorithms drop as the system average utilization increases. This is reasonable since task sets with high utilizations are difficult to be scheduled, especially when fault-tolerance is considered. BFD has the worst performance since it does not take the characteristics of tasks (e.g. harmonicity or compatibility) into consideration. With a system utilization of 0.5, BFD can only achieve an acceptance ratio less than 20%. In the remaining experiments, we excluded BFD for comparison unless otherwise specified. Both of our proposed algorithms outperforms HAPS, due to the fact that our algorithms utilize a more accurate metric to capture how compatible tasks are during the partition process. In general, CATP has a better performance than HAPS. For example, with a system average utilization of 0.55, CATP has an acceptance ratio of 80% while HAPS only achieves 52%, which is an approximate 60% improvements. $G-CATP$ has the best performance as it tries to search for the most compatible group of tasks in each step. For instance, when the system average utilization is 0.65, $G-CATP$ still achieves an acceptance ratio of 41%, while $CATP$ and $HAPS$ only have an acceptance ratio less than 20%. In average, CATP obtains a 24% improvement over HAPS, and $G-CATP$ manages to get a further 40% enhancement over CATP.

Next, we evaluated these algorithms on a 8-core platform with 64 tasks. Additionally, a maximum of 2 faults was assumed and the system average utilization was varied in the range $[0.5, 1]$ with a step of 0.05. The superiority of our proposed algorithms over HAPS is illustrated in Figure 4, 1000 task sets were generated for each point on the $x-axis$.

As the number of cores and tasks increase, all three algorithms exhibit higher acceptance ratio. This is because with a higher number of tasks, each task is likely to be associated with a smaller utilization given a fixed system average utilization and they are easier to be scheduled. CATP still exhibits significant improvement over HAPS. When the system average utilization is 0.6, HAPS has an acceptance ratios of 30%, while CATP achieves twice the value, i.e. 60%. G-CATP still has the dominated performance among all three algorithms. In average, CATP attains a 48% improvement in performance over HAPS whereas G-CATP further enhances

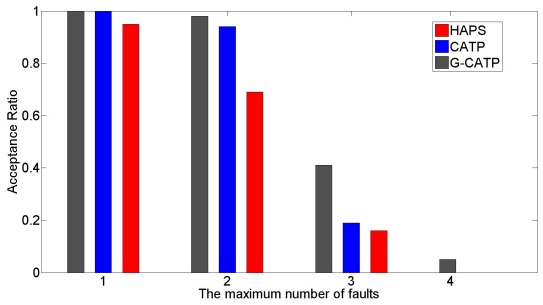

Fig. 5. 32 tasks on 4-core platform, system average utilization is 0.5.

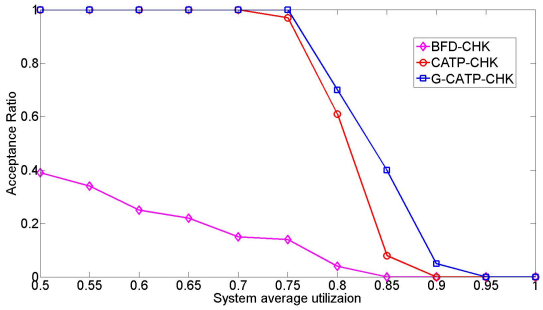

Fig. 6. 32 tasks on 4-core platform, checkpoint overhead is 5 percent of execution time, K=2.

CATP by an approximate 45%. Both our algorithms tend to have better performances for systems with more tasks and cores, since they aggressively find the most compatible tasks in each step.

B. Experiment 2, acceptance ratio vs. the number of faults.

In this section, we investigate the relationships between acceptance ratio and the number of faults that a system needs to tolerate. We fixed the system average utilization as 0.5 with a 4-core platform consisting of 32 tasks. We varied the number of faults from 1 to 4, i.e. $K \in [1,4]$. The result is shown in Figure 5, 1000 task sets were generated for each configuration.

As the number of fault increases, the acceptance ratio decreases dramatically for all three algorithms. However, both of our algorithms, i.e. CATP and G-CATP outperform the algorithm HAPS. When K = 2, HAPS has an acceptance ratio of 70% while CATP and G-CATP can achieve 92% and 95%, respectively. Our G-CATP algorithm still exhibits the best performance, it manages to achieve 40% acceptance ratio while CATP and HAPS only have an acceptance ratio less than 20% when $K = 3$.

The experimental results clearly demonstrate the effectiveness of our proposed algorithms in terms of improving system schedulability under failures. By grouping compatible tasks and assigning them to the same core, we significantly enhance system utilization and leave more space for the remaining tasks.

C. Experiment 3, acceptance ratio vs. checkpointing

In this section, we study the effects of checkpointing on system schedulability. Since there is no existing work in the literature that solves the exact same problem, we first extend the BFD to incorporate the checkpointing feature. The tasks are sorted in non-increasing order of utilization, the algorithm allocates one task at a time and use algorithm ECHK [20] to search for a feasible checkpointing scheme for this task and tasks on each candidate core. Among all feasible cores (a core is considered feasible if there exists a feasible checkpointing configuration among the tasks on the core and the task to be allocated), the task is assigned to the core with the least remanning utilization. BFD reports "FAILURE" is a task can not be assigned. We denote this algorithm as BFD-CHK. The performances of BFD-CHK and our two algorithm CATP-CHK and G-CATP-CHK were evaluated.

We considered 32 tasks on a 4-core platform. The maximum number of faults, i.e. K, was set to 2 and the checkpointing overhead was assumed to be 5% of the worst case execution time for each task. System average utilization was varied from 0.5 to 1 with a step of 0.05. 1000 task sets were generated for each test case and the acceptance ratios were plot in Figure 6.

Compared with Figure 3, the acceptance ratios are increased significantly. For instance, CATP-CHK and G-CATP-CHK and can still achieve 60% and 70% acceptance ratios under a relatively high system average utilization of 0.8 (zero under CATP and G-CATP), respectively. This is due to the fact that checkpointing can considerably reduce the recovery overhead of each task and enhance the system schedulability under faults. As can be seen, our algorithms substantially outperform BFD-CHK. With a utilization of 0.5, CATP-CHK and G-CATP-CHK both achieve an acceptance ratio of 100% whereas BFD-CHK only achieves 40%. Among all three algorithms, G-CATP-CHK still exhibits the best performance since it always tries to search for the most compatible tasks in each step. When system average utilization is 0.85, G-CATP-CHK has an acceptance ratio about 40%, while that of CATP-CHK is less than 10%. Once again, the simulation results demonstrate that *compatibility index* is an accurate metric to quantify how "compatible" a task set is and can guide task partitioning correctly.

IX. Conclusion and Future Directions

As the computing paradigm shifts toward multi-core platforms, the need for effective and efficient multi-core scheduling is ever-growing. Also, facing the unprecedented reliability challenges brought forth by relentless transistor miniaturizations and mass integrations of transistors into a single chip, traditional multi-core scheduling without explicitly considering system reliability is becoming obsolete. In this paper, we first present an efficient test to evaluate the schedulability of tasks scheduled according to rate-monotonic method under faults. Then, we develop a novel metric to quantify the "compatibility" among tasks, which is a direct indication of system schedulability. In light of this metric, we develop two partitioning approaches CATP and G-CATP. While algorithm

CATP assigns one task at a time to the most compatible core, G-CATP searches for the most compatible group of tasks in each step and assigns them to one core. We further extend our algorithms to incorporate the checkpointing scheme to further improve system utilization. Simulation results have shown that our proposed algorithms can achieve substantial improvements over other related approaches.

Adopting the concept of "compatibility index" for various fault-tolerant real-time task models and studying the corresponding partitioning problem on multi-core platforms are very interesting research directions. For example, when considering constrained-deadline tasks ($D_i \leq T_i, \forall i$) instead of implicit-deadline tasks ($D_i = T_i, \forall i$), RMS is not optimal anymore. What's worse, the *harmonic distance* and the *extra recovery utilization* that are defined solely based task execution times and periods may become inaccurate to quantify how "compatible" a set of task is. Adding the third dimension, i.e. deadline, to "compatibility index" and judiciously make partitioning decisions are not trivial problems and require careful investigation.

Another interesting direction is to extend the approaches in this paper to heterogenous multi-core platforms. Different from homogenous multi-core platform, the execution profile for each task may vary widely from core to core. To improve task set schedulability, one intuitive approach is to assign "heavy tasks" to fast cores. However, a task may take less execution time on one core but is more compatible with the tasks on the other core. How to make tradeoffs between execution speed and compatibility is worth careful studying.

X. Acknowledgement

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

References

[1] 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.

[2] 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

[3] S. Borkar, "Thousand core chips: A technology perspective," in *Proceedings of the 44th Annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 746–749.

[4] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–35:44, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978814

[5] T. Wang, L. Niu, S. Ren, and G. Quan, "Multi-core fixed-priority scheduling of real-time tasks with statistical deadline guarantee," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. San Jose, CA, USA: EDA Consortium, 2015, pp. 1335–1340.

[6] 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.

[7] M. Fan, Q. Han, G. Quan, and S. Ren, "Multi-core partitioned scheduling for fixed-priority periodic real-time tasks with enhanced rbound," in *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, March 2014, pp. 284–291.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, pp. 46–61, January 1973. [Online]. Available: http://doi.acm.org/10.1145/321738.321743

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

[10] 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.

[11] T. Chantem, Y. Xiang, X. S. Hu, and R. P. Dick, "Enhancing multicore reliability through wear compensation in online assignment and scheduling," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13. San Jose, CA, USA: EDA Consortium, 2013, pp. 1373–1378. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485616

[12] M. Fan and G. Quan, "Harmonic-fit partitioned scheduling for fixed-priority real-time tasks on the multiprocessor platform," in *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, Oct 2011, pp. 27–32.

[13] M. Pandya and M. Malek, "Minimum achievable utilization for fault-tolerant processing of periodic tasks," *IEEE Trans. on Computers*, vol. 47, pp. 1102–1112, 1994.

[14] A. Burns, R. Davis, and S. Punnekkat, "Feasibility analysis of fault-tolerant real-time task sets," in *Real-Time Systems, 1996., Proceedings of the Eighth Euromicro Workshop on*, Jun 1996, pp. 29–33.

[15] 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.

[16] B. Andersson and J. Jonsson, "The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50," in *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, July 2003, pp. 33–40.

[17] B. Chattopadhyay and S. Baruah, "Partitioned scheduling of implicit-deadline sporadic task systems under multiple resource constraints," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, Aug 2012, pp. 144–153.

[18] 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.

[19] Y. Guo, D. Zhu, and H. Aydin, "Generalized standby-sparing techniques for energy-efficient fault tolerance in multiprocessor real-time systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2013 IEEE 19th International Conference on*, Aug 2013, pp. 62–71.

[20] Q. Han, M. Fan, L. Niu, and G. Quan, "Energy minimization for fault tolerant scheduling of periodic fixed-priority applications on multiprocessor platforms," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15, 2015, pp. 830–835.

[21] X. Castillo, S. R. McConnel, and D. P. Siewiorek, "Derivation and calibration of a transient error reliability model," *IEEE Trans. Comput.*, vol. 31, pp. 658–671, July 1982.

[22] T. Wei, P. Mishra, K. Wu, and J. Zhou, "Quasi-static fault-tolerant scheduling schemes for energy-efficient hard real-time systems," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1386 – 1399, 2012, special Issue: Agile Development. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212000064

[23] Q. Han, L. Niu, G. Quan, S. Ren, and S. Ren, "Energy efficient fault-tolerant earliest deadline first scheduling for hard real-time systems," *Real-Time Systems*, vol. 50, no. 5-6, pp. 592–619, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11241-014-9210-z

[24] C.-C. Han and H.-Y. Tyan, "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms," in *Proceedings of the 18th IEEE Real-Time Systems Symposium*, ser. RTSS '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 36–. [Online]. Available: http://dl.acm.org/citation.cfm?id=827269.828999

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