The Journal of Systems and Software 99 (2015) 85–96

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Enhanced fixed-priority real-time scheduling on multi-core platforms by exploiting task period relationship☆

Ming Fan[a], Qiushi Han[a,*], Shuo Liu[a], Shaolei Ren[a,c], Gang Quan[a], Shangping Ren[b]

[a] Department of Electrical and Computer Engineering, Florida International University, 10555 West Flagler Street, Miami, FL 33174, United States
[b] Department of Computer Science, Illinois Institute of Technology, 10 West 31st Street, Chicago, IL 60616, United States
[c] School of Computing and Information Sciences, Florida International University, 11200 SW 8th Street, ECS 350, Miami, FL 33199, United States

ARTICLE INFO

Article history:
Received 1 May 2014
Received in revised form 2 September 2014
Accepted 4 September 2014
Available online 16 September 2014

Keywords:
Partitioned scheduling
RMS
Harmonic

ABSTRACT

One common approach for multi-core partitioned scheduling problem is to transform this problem into a traditional bin-packing problem, with the utilization of a task being the "size" of the object and the utilization bound of a processing core being the "capacity" of the bin. However, this approach ignores the fact that some implicit relations among tasks may significantly affect the feasibility of the tasks allocated to each local core. In this paper, we study the problem of partitioned scheduling of periodic real-time tasks on multi-core platforms under the *Rate Monotonic Scheduling* (*RMS*) policy. We present two effective and efficient partitioned scheduling algorithms, i.e. *PSER* and *HAPS*, by exploiting the fact that the utilization bound of a task set increases as task periods are closer to harmonic on a single-core platform. We formally prove the schedulability of our partitioned scheduling algorithms. Our extensive experimental results demonstrate that the proposed algorithms can significantly improve the scheduling performance compared with the existing work.

1. Introduction

Multi-core architecture has been widely accepted as the most important technology in the future industrial market. By providing multiple processing cores on a single chip, multi-core systems can significantly increase the computing performance while relaxing the power requirement over traditional single-core systems. Most of the major chip manufactures have already launched multi-core chips into the market, i.e. AMD *Opteron*™ *6300 Series* (AMD, 2013). It is not surprising that in the coming future, hundreds or even thousands of cores will be integrated into a single chip (Yeh et al., 2008). The quickly emerging trend towards multi-core platform brings urgent needs for effective and efficient techniques for the design of different types of computing systems, e.g. real-time systems.

One major problem in the design of multi-core real-time system is how to utilize the available computing resources most efficiently while satisfying the timing constraints of all real-time tasks. To address this problem, one effective way is to develop an appropriate scheduling algorithm, which plays one of the most significant roles in real-time operating systems.

It is well known that the scheduling problem for multi-core systems is an NP-hard problem (Shin and Ramanathan, 1994). Although there exists optimal scheduling algorithms on single-core systems, i.e. *Rate Monotonic Scheduling (RMS)* and *Earliest Deadline First (EDF)* (Liu and Layland, 1973), none of them are optimal any more (Dhall and Liu, 1978) for multi-core systems. The reason is that, different from single-core scheduling, the multi-core scheduling needs to decide not only when but also where to execute a real-time task. Therefore, developing a sub-optimal heuristic for scheduling strategy on multi-core systems is reasonable and practical.

In this paper, we are interested in studying the problem of partitioned scheduling for periodic tasks on multi-core systems under RMS policy. Compared with the existing works on fixed-priority partitioned scheduling, we have made a number of contributions:

- We develop two new partitioned scheduling algorithms for fixed-priority periodic real-time tasks by taking the relationship among task periods into consideration. The first algorithm, namely *Partitioned Scheduling with Enhanced RBound (PSER)*, improves the traditional R-Bound by applying a more flexible task set scaling method (i.e. *TSS* method) and then partitions tasks under

☆ This work is supported in part by NSF under projects CNS-0969013, CNS-1423137, CAREER-0746643, CNS-1018731, CNS-0917021 and CNS-1018108.
* Corresponding author. Tel.: +1 3059151789.
E-mail addresses: mfan001@fiu.edu (M. Fan), qhan001@fiu.edu (Q. Han), sliu005@fiu.edu (S. Liu), sren@cs.fiu.edu (S. Ren), ganquan@fiu.edu (G. Quan), ren@iit.edu (S. Ren).

http://dx.doi.org/10.1016/j.jss.2014.09.010
0164-1212/

the enhanced utilization bound (i.e. $RBound^{en}$). The second one, namely *Harmonic Aware Partition Scheduling (HAPS)*, captures the "degree" of harmonic for a set of tasks with a novel harmonic metric (i.e. *harmonic index*) and then makes the partitioning decision such that tasks with closer harmonic relationship can be allocated to the same core.
- We analytically prove that both of our proposed partitioned scheduling algorithms, i.e. PSER and HAPS, can guarantee the schedulability of any task set that can successfully pass the partitioning procedure.
- We conduct extensive experiments to evaluate the performance of our proposed techniques. Through the experimental results, we can see that our proposed algorithms can significantly improve the scheduling performance compared with the existing works.

The rest of the paper is organized as follows. Section 2 introduces the work closely related to this paper. Section 3 describes the system models and Section 4 presents our motivational examples for this work. Section 5 and 6 present two partitioned scheduling algorithms we developed. Experiments and results are discussed in Section 7, and we conclude this work in Section 8.

2. Related work

In partitioned multi-core scheduling problem, the schedulability for tasks allocated on each processor can be determined based on feasibility conditions on single processors. To search for the optimal task partition for multiple processors is essentially a design space exploration problem, with complexity increasing rapidly with the size of the problem (e.g. the numbers of tasks or processors). How to quickly and accurately evaluate the schedulability of a design alternative (i.e. task partition) is key to the success of the partitioned multi-core scheduling problem. As a result, while there exists exact timing analysis method for feasibility checking for tasks on a single core platform (Liu and Layland, 1973; Kuo and Mok, 1991; Lauzac et al., 1998), they are not commonly used for partitioned multi-core scheduling problem due to their large computational complexity. Instead, many other timing-efficient feasibility checking methods, such as the utilization-bound based feasibility checking methods, are commonly used in the search for task partitions for multi-core scheduling problem.

2.1. Different utilization bounds for single-core systems

A *utilization bound* $f(\Gamma)$ for a task set Γ is a function of the parameters of Γ, and can be used to determine the schedulability of Γ under certain specific scheduling policy (e.g. RMS). By applying the parameters of Γ into $f(\Gamma)$, all tasks in Γ can be guaranteed to meet their deadlines if the task set utilization (denoted as $U(\Gamma)$) is no more than that parametric utilization bound, i.e. $U(\Gamma) \le f(\Gamma)$. Note that $U(\Gamma)$ can be calculated by summing up the task utilizations of all tasks in the task set Γ, where a task utilization is the ratio of its execution time over its period.

For single-core systems, there are several utilization bounds proposed under RMS policy (Liu and Layland, 1973; Kuo and Mok, 1991; Lauzac et al., 1998).

- *LLBound* (Liu and Layland, 1973): The *LLBound* is a function with respect to the number of tasks, and is formulated as

$$LLBound(\Gamma) = N(2^{1/N} - 1), \tag{1}$$

where N is the number of tasks in the task set Γ. When N goes to infinity, the *LLBound* achieves its worst-case as 69%.

- *KBound* (Kuo and Mok, 1991): The *KBound* has a similar form as the *LLBound*, and is formulated as

$$KBound(\Gamma) = K(2^{1/K} - 1), \tag{2}$$

where K, instead of being the number of all tasks as that used by *LLBound*, is the number of tasks in original task sets such that no two tasks are completely harmonic.
- *RBound* (Lauzac et al., 1998): The *RBound* takes not only the number of tasks but also the relationship among periods into consideration, i.e.

$$RBound(\Gamma) = (N-1)(r^{1/N-1} - 1) + 2/r - 1, \tag{3}$$

where N is the number of tasks in the task set, and r is the ratio between the maximum and minimum periods and need to satisfy $1 \le r < 2$.
- *CBound* (Han and Tyan, 1997): The *CBound* is the utilization bound for a *harmonic task set*, in which the periods of any two tasks being integer multiple of each other, i.e.

$$CBound(\Gamma) = 1, \tag{4}$$

where Γ is a harmonic task set.

Among all four utilization bounds shown in the above, it has been proved that for RMS-based single-core scheduling, the *RBound* and *CBound* are higher than the other two (i.e. the *LLBound* and the *KBound*) (Lauzac et al., 1998; Han and Tyan, 1997). However, these two utilization bounds (*RBound* or *CBound*) have critical limitations. The *RBound* can only be applied when a given task set satisfies the period constraint (i.e. $1 \le r < 2$), while the *CBound* can only be used directly to harmonic task sets. Hence, in order to use the *RBound* or *CBound* for checking the schedulability of an arbitrary task set, we need to first transform the task set appropriately such that it satisfies the required condition.

For *RBound*, there are a few methods proposed to transform a task set to satisfy the condition of $1 \le r < 2$, such as Lauzac et al. (1998) and Kandhalu et al. (2012). In particular, Lauzac et al. (1998) proposed a task set scaling method by scaling all tasks with respect to the maximum period. Specifically, given a task set Γ, $\forall \tau_i \in \Gamma$, the period as well as the execution time of τ_i was scaled by

$$\begin{cases} C_i' = C_i \cdot 2^{\lfloor \log(T_{\max}/T_i) \rfloor} \\ T_i' = T_i \cdot 2^{\lfloor \log(T_{\max}/T_i) \rfloor} \end{cases} \tag{5}$$

where T_{max} represents the maximum period among all tasks. Their method scaled all task periods with respect to, but no larger than T_{max}. They formally proved that as long as the scaled task set is feasible then the original task set is also feasible.

Kandhalu et al. (2012) presented another method by scaling the task set with respect to the minimum period. Specifically, given a task set Γ, $\forall \tau_i \in \Gamma$, the period and the execution time of τ_i was scaled by

$$\begin{cases} C_i' = C_i / \lfloor (T_i/T_{\min}) \rfloor \\ T_i' = T_i / \lfloor (T_i/T_{\min}) \rfloor \end{cases} \tag{6}$$

where T_{min} is the minimum period among all tasks. This method scaled all task periods with respect to, but no smaller than T_{min}. However, this approach cannot always guarantee the schedulability of the original task set even when the scaled task set is schedulable. For example, consider a task set Γ consisting of four tasks with execution time and periods as {(3,24),(32,100),(40,135)} and (15,140). According to the scaling method introduced in Kandhalu et al. (2012), we can transform the task set to a new task set Γ' as {(3,24),(8,25),(8,27),(3,28)}. It is not difficult to verify that the new task set Γ' is schedulable while the original task set Γ is not schedulable.