Journal of Systems Architecture 60 (2014) 247–257

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Deadline and activation time assignment for partitioned real-time application on multiprocessor reservations

Yifan Wu *, Zhigang Gao, Guojun Dai

Department of Computer Science and Technology, Hangzhou Dianzi University, 310018 Hangzhou, Zhejiang, China

ARTICLE INFO

Article history:
Available online 7 December 2013

Keywords:
Real-time systems
Resource reservation
Deadline assignment
Multiprocessor

ABSTRACT

Providing temporal isolation between critical activities has been an important design criterion in real-time open systems, which can be achieved using resource reservation techniques. As an abstraction of reservation servers, virtual processor is often used to represent a portion of computing power available on a physical platform while hiding the implementation details. In this paper, we present a general framework of partitioning an application comprised of hard real-time tasks with precedence constraints onto multiple virtual processors in consideration of communication latencies between tasks. A novel method is proposed for assigning deadlines and activation times to tasks such that tasks partitioned onto different virtual processors can be analyzed separately using well-established theories for uniprocessor. Extensive simulations have been performed and the results have shown that, compared to existing algorithms, the proposed method achieves better performance in terms of minimizing both total bandwidth and the maximum individual bandwidth.

1. Introduction

Modern embedded systems are shifting towards computation platform with multiple processing units. Multiprocessor or multi-core architectures are favored for their potential of providing higher computational power with reduced energy consumption and heat dissipation. However, to fully exploit this capability, the intrinsic parallelism of software application must be taken into account, leading to increased dimension of system design space. In fact, the problem of allocating and scheduling real-time tasks to multiple processing units is known to be NP-hard, and has received considerable amount of attention from the research community.

On the other hand, the increasing functionality and services provided by the embedded systems add significant complexity to the design and verification of the system. While off-the-shelf modules and legacy software are used to reduce the development cycle and cost, applications from various software vendors characterized with diverse timing properties are competing for the same resources. In such an *open system* [1], isolating the temporal behavior of different applications is of crucial importance for preventing reciprocal interference among critical activities. To accomplish this, *resource reservation* [2,3] has been proposed as a promising technique to provide temporal isolation between real-time tasks.

On uniprocessor platform, resource reservation technique is used to partition the CPU processing capacity into a set of reservations, each characterized with a couple (Q_k, P_k) indicating that Q_k units of processing time are available every period P_k. In that case, each reservation is equivalent to a virtual processor with a reduced bandwidth $\alpha_k = Q_k/P_k$. The use of virtual processor has several advantages. Besides the protection of temporal behavior of the allocated tasks from interference of other tasks, analysis and design based on virtual processors also improve modularity and simplify portability on different architectures using different server mechanisms. In recent years, resource reservation on multiprocessor platform has received increasing research attention, where reservation servers for uniprocessor platform are extended [4], and new abstraction models are proposed [5–7]. In this paper, we abstract the physical multiprocessor platform using a set of uniprocessor reservations described through the Multi-Supply Function (MSF) model proposed by Bini et al. [8]. Each reservation is represented by a bounded-delay time partition, denoted by the pair (α, Δ), where α is the allocated bandwidth and Δ is the maximum service delay. This method, originally proposed by Mok et al. [9], is general enough to express several types of resource reservation servers.

We present a general framework of partitioning real-time applications to multiple virtual processors. A real-time application is described as a set of tasks with precedence relations that forms a directed acyclic graph. Upon dividing the task set into several subsets, the activation time and deadline of each task are set in accordance with timing and precedence constraints. This allows exploiting the well-established analysis techniques of real-time systems on uniprocessor for each individual task subset. Each subset of tasks is then allocated to a virtual processor, and the reservation parameters for each virtual processor are optimally

* Corresponding author. Tel.: +86 13967104602.
E-mail address: yfwu@hdu.edu.cn (Y. Wu).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.11.011

chosen to minimize the resource provision while maintaining the schedulability of the system.

The theoretical foundation of this paper is based on previous work presented in [10]. However, several key contributions have been made. (a) The communication costs between tasks were completely ignored in [10], which made the proposed method only applicable to computation-intensive applications. Our work takes communication cost as a major design concern and hence can be applied to both homogeneous multi-core or multiprocessor systems. (b) The deadline setting method in [10] is merely based on intuition and may not achieve optimal performance under certain scenarios. In this paper, a novel method for activation time and deadline assignment is proposed aiming at minimizing both the total resource demands and the maximum individual resource requirement. The problem can be efficiently solved using linear programming and produces better performance in general case.

The remainder of the paper is organized as follows. Related works are described and evaluated in Section 2. The system model and terminology are introduced in Section 3. Section 4 presents the general framework, together with some important adaption of previous works in consideration of a more general system model. Section 5 proposes a novel method for deadline and activation time assignment, followed by simulation and experimental evaluations in Section 6. Finally, Section 7 concludes the paper and plans for the future work.

2. Related works

Several works have been proposed to solve the problem of partitioning a group of tasks on multiprocessor/multi-core platforms. Baruah and Fisher [11] presented a heuristic algorithm to partition a set of sporadic real-time tasks to multiprocessor architecture, without consideration of tasks' precedence relations. Schranzhofer et al. [12] proposed a method to divide tasks to multiprocessor platform aiming at minimizing the average power consumption. However, they did not consider real-time tasks. Li et al. [13] considered the partition problem of real-time tasks onto multiprocessor, which intends to minimize system workload and number of occupied processors without violating real-time constraints. The method mainly targets at real-time transactions in real-time databases, rather than general parallel applications.

To partition real-time tasks to system with multiple processing units, Peng and Shin [14] employed the task graph to represent a real-time periodic application, which is then divided to distributed processing units using branch and bound search. The design goal is to minimize the response time. Ramamritham [15] presented a search method to allocate periodic tasks to distributed system, considering precedence relations, communication costs and repetition of the tasks. Abdelzaher and Shin [16] considered the problem of partitioning large-scale real-time systems on heterogeneous distributed systems. The proposed method tries to find the best matching of clustered tasks and processing units, and has good scalability. These methods mainly considered how to achieve a feasible solution or minimize the response time, rather than to optimize the resource utilization.

There are two main categories of partitioning tasks with precedence relations and non-negligible communication costs. One category, called List Scheduling [17,18], is based on the idea that assigns the task with the highest priority to the best suitable processor at each step of the process. The other category is based on critical path [19–21] which tries to reduce the length of critical path at each time a task is selected to allocate. The two categories of algorithms are mostly designed to reduce the length of the schedule (*makespan*), and do not consider real-time constraints.

The problem of managing real-time tasks with precedence relations was addressed by Chetto et al. [22], who proposed a method to convert a precedence graph into timing constraints by assigning proper activation times and deadlines to all tasks, in such a way that the schedulability under EDF was guaranteed. However, this algorithm does not consider the possibility of having parallel execution and is only applicable to uniprocessor systems. Di Natale and Stankovic [23] presented a method to assign individual deadlines based on the distribution of the laxity equally among all tasks, while Kao and Garcia-Molina [24] proposed to assign deadlines to real-time tasks by dividing the end-to-end deadline proportionally to the computation time of all tasks. Both methods did not consider resource utilization. Serreli et al. [25] formulated an optimization problem to decide the assignment of intermediate deadlines for EDF scheduled real-time tasks, which however is only valid for real-time transactions.

3. System model

An *application* Γ is modeled as a Directed Acyclic Graph (DAG) $G = (V, E)$, also called a *task graph*, where V is a set of $v = |V|$ hard real-time tasks and E is a set of $e = |E|$ precedence relations between tasks, where $|.|$ is the cardinality of a set. The application is characterized with a minimum inter-arrival time T (also referred to as *period*), which is the smallest time interval between two instances of a task. A *relative deadline* D represents the end-to-end deadline of the application with respect to its activation. That is, assuming an instance of Γ arrives at time t, it must complete all tasks' execution before $t + D$. Assuming D can be no greater than T, it indicates that only one instance of Γ occurs at any time. Without loss of generality, Γ is assumed to start at $t = 0$.

Each *task* $\tau_i \in V$ represents a portion of sequential execution of the application which can not be parallelized, characterized by the *worst-case computation time* C_i, the *deadline* d_i and the *activation time* a_i. Notice that, d_i and a_i are not set by the application designer. Rather, they are assigned as internal parameters only for the purpose of scheduling in the real-time operating system. Tasks are fully preemptible and scheduled by Earliest Deadline First (EDF).

The *precedence relation* in the set E, denoted by $\tau_i \rightarrow \tau_j$, represents that τ_i is a *predecessor* of τ_j, and τ_j is a *successor* of τ_i. In the DAG, a task having no predecessor is called an *entry task*, and a task having no successor is called an *exit task*. Each precedence relation is associated with a *worst-case communication latency* $\delta_{i,j}$, which is the maximum time required to communicate the data from τ_i to τ_j. The *communication-to-computation ratio CCR* of an application is defined as its average worst-case communication latency divided by its average worst-case computation time, i.e., $\frac{\sum_{\forall \tau_i \rightarrow \tau_j \in E} \delta_{i,j}/e}{\sum_{\forall \tau_i \in V} C_i/v}$. An application with $CCR \ll 1$ is said to be computation-intensive.

We assume that the underlying platform has homogeneous processing units, and the communications between tasks can take place simultaneously by incorporating the queuing time for memory access or message transmission into the worst-case communication latency $\delta_{i,j}$. Furthermore, $\delta_{i,j}$ is assumed to be independent with the physical location of tasks on the processing units. This can be achieved by, for instance, fully connecting all the processing units [21] or employing prioritized time-division-multiplexed approach for on-chip communication [26].

A *virtual processor* VP_k is an abstraction of a sequential machine achieved through a resource reservation mechanism characterized by a bandwidth $\alpha_k \leqslant 1$ and a maximum service delay $\Delta_k \geqslant 0$. It is also called a (α, Δ) server [9], where the correct timing behavior of tasks that are allocated to it are guaranteed only by the assigned parameter α_k and Δ_k, and is independent of the behavior of other tasks.

A *flow* F_k is a non-empty subset of tasks $F_k \subseteq V$ allocated on virtual processor VP_k, which is dedicated to the execution of tasks in