Journal of Systems Architecture 60 (2014) 152–164

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Supporting soft real-time parallel applications on multiprocessors ☆

Cong Liu [a,*], James H. Anderson [b]

[a] Department of Computer Science, The University of Texas at Dallas, United States
[b] Department of Computer Science, The University of North Carolina at Chapel Hill, United States

ARTICLE INFO

Article history:
Available online 31 July 2013

Keywords:
Multiprocessor scheduling
Real-time systems
Parallel applications
Response time bounds

ABSTRACT

The prevalence of multicore processors has resulted in the wider applicability of parallel programming models such as OpenMP and MapReduce. A common goal of running parallel applications implemented under such models is to guarantee bounded response times while maximizing system utilization. Unfortunately, little previous work has been done that can provide such performance guarantees. In this paper, this problem is addressed by applying soft real-time scheduling analysis techniques. Analysis and conditions are presented for guaranteeing bounded response times for parallel applications under global EDF multiprocessor scheduling.

Published by Elsevier B.V.

1. Introduction

The growing prevalence of multicore platforms has resulted in the wider applicability of parallel programming models such as OpenMP [3] and MapReduce [5]. Such models can be applied to parallelize certain segments of programs, thus better utilizing hardware resources and possibly shortening response times. Many applications implemented under such parallel programming models have soft real-time (SRT) constraints. Examples include real-time parallel video and image processing applications [1,7] and computer vision applications such as colliding face detection and feature tracking [11]. In these applications, providing fast and bounded response times for individual video frames is important, to ensure smooth video output. However, achieving this at the expense of using conservative hard real-time (HRT) analysis is not warranted. In this paper, we consider how to schedule parallel task systems that require such SRT performance guarantees on multicore processors.

Parallel task models pose new challenges to real-time scheduling since intra-task parallelism has to be specifically considered. Recent papers [12,29] on scheduling real-time periodic parallel tasks have focused on providing HRT guarantees under global-earliest-deadline-first (GEDF) or partitioned deadline-monotonic (PDM) scheduling. However, as discussed above, viewing parallel tasks as HRT may be overkill in many settings and furthermore may result in significant schedulability-related utilization loss. Thus, our focus is to instead ensure bounded response times in supporting parallel task systems by applying SRT scheduling analysis techniques. Specifically, we assign deadlines to parallel tasks and schedule them using GEDF, but in contrast to previous work [12,29], we allow deadlines to be missed provided the extent of such misses is bounded (hence response times are bounded as well). Moreover, we consider a generalized parallel task model that removes some of the restrictions seen in previous work (as discussed below).

Response time bounds have been studied extensively in the context of global real-time scheduling algorithms such as GEDF [6,13–25]. It has been shown that a variety of such algorithms can ensure bounded response times in ordinary real-time sporadic task systems (i.e., without intra-task parallelism) with no utilization loss on multiprocessors [6,13].[1] Motivated by these results, we consider whether it is possible to specify reasonable constraints under which bounded response times can be guaranteed using global real-time scheduling techniques, for sporadic parallel task systems that are not HRT in nature.

Related work. Scheduling non-real-time parallel applications is a deeply explored topic [4,5,8,9,26,31,32]. However, in most (if not all) prior work on this topic, including all of the just-cited work, scheduling decisions are made on a best-effort basis, so none of these results can provide performance guarantees such as response time bounds.

Regarding scheduling HRT parallel task systems, Lakshmanan et al. proposed a scheduling technique for the *fork-join* model, where a parallel task is a sequence of segments, alternating between sequential and parallel phases [12]. A sequential phase

☆ Work supported by AT&T, IBM, Intel, and Sun Corps.; NSF Grants CNS 0834270, CNS 0834132, and CNS 0615197; and ARO Grant W911NF-06-1-0425.
* Corresponding author. Tel.: +1 9193601521; fax: +1 9199621799.
E-mail address: cong.liu@utdallas.edu (C. Liu).

[1] Technically, bounded response times can only be ensured for task systems that do not over-utilize the underlying platform. In all claims in this paper concerning bounded response times, a non-over-utilized system is assumed.

1383-7621/$ - see front matter Published by Elsevier B.V.
http://dx.doi.org/10.1016/j.sysarc.2013.07.001

contains only one thread while a parallel phase contains multiple threads that can be executed concurrently on different processors. In their model, all parallel phases are assumed to have the same number of parallel threads, which must be no greater than the number of processors. Also, all threads in any parallel segment must have the same execution cost. The authors derived a resource augmentation bound of 3.42 under PDM scheduling.

In [29], Saifullah et al. extended the fork-join model so that each parallel phase can have a different number of threads and threads can have different execution costs. The authors proposed an approach that transforms each periodic parallel task into a number of ordinary constrained-deadline periodic tasks by creating per segment intermediate deadlines. They also showed that resource augmentation bounds of 2.62 and 3.42 can be achieved under GEDF and PDM scheduling, respectively. In [27], Nelissen et al. proposed techniques that optimize the number of processors needed to schedule sporadic parallel tasks. The authors also proved that the proposed techniques achieve a resource augmentation bound of 2.0 under scheduling algorithms such as U-EDF [28] and PD^2 [30].

In this paper, we seek to efficiently support parallel task systems on multiprocessors with bounded response times. We consider the general parallel task model as presented in [27,29]. A fundamental difference between this work and prior work is that we propose a SRT schedulability analysis framework to derive conditions for guaranteeing bounded response times.

Contributions. In this paper, we show that by assigning deadlines to parallel task systems and scheduling them under GEDF, such systems can be supported on multiprocessors with bounded response times. Our analysis shows that on a two-processor platform, no utilization loss results for any parallel task system. Despite this special case, on a platform with more than two processors, utilization constraints are needed. To discern how severe such constraints must fundamentally be, we present a parallel task set with minimum utilization that is unschedulable on any number of processors. This task set violates our derived constraint and has unbounded response times. The impact of utilization constraints can be lessened by restructuring tasks to reduce intra-task parallelism. We propose optimization techniques that can be applied to determine such a restructuring. Finally, we present the results of experiments conducted to evaluate the applicability of the derived schedulability condition.

Organization. The rest of this paper is organized as follows. Section 2 describes our system model. In Section 3, we present our analytical results. In Section 4, we discuss the above mentioned optimization technique. In Section 5, we experimentally evaluate the proposed analysis. Section 6 concludes.

2. System model

We consider the problem of scheduling a set $\tau = \{\tau_1, \ldots, \tau_n\}$ of n independent sporadic parallel tasks on m processors. Each parallel task τ_i is a sequence of s_i segments, where the jth segment τ_i^j contains a set of v_i^j threads ($v_i^j > m$ is allowed). The kth ($1 \leqslant k \leqslant v_i^j$) thread $\tau_i^{j,k}$ in segment τ_i^j has a worst-case execution time of $e_i^{j,k}$. We assume that each thread $\tau_i^{j,k}$ executes for exactly $e_i^{j,k}$ time units. This assumption can be eased to treat $e_i^{j,k}$ as an upper bound, at the expense of more cumbersome notation. For notational convenience, we order the threads of each segment τ_i^j of each parallel task τ_i in largest-worst-case-execution-time-first order. Thus, thread $\tau_i^{j,1}$ has the largest worst-case execution time among all threads in any segment τ_i^j. For any segment τ_i^j, if $v_i^j > 1$, then the threads in this segment can be executed in parallel on different processors. The threads in the jth segment can execute only after all threads of $(j-1)$th segment (if any) have completed. We let v_i^{max} denote the maximum number of threads in any segment of task τ_i. We assume $v_i^{max} \geqslant 2$ holds for at least one task τ_i; otherwise, the considered task system is simply an ordinary sporadic task system (without intra-task parallelism).

The worst-case execution time of any segment τ_i^j is defined as $e_i^j = \sum_{k=1}^{v_i^j} e_i^{j,k}$ (when all threads execute sequentially). The worst-case execution time of any parallel task τ_i is defined as $e_i = \sum_{j=1}^{s_i} e_i^j$ (when all threads in each segment of the task execute sequentially). In our analysis, we also make use of the best-case execution time of τ_i on m processors (when τ_i is the only task executing on m processors), denoted e_i^{min}. In general, for any parallel task τ_i, if we allow $v_i^{max} \geqslant m$ and threads in each segment have different execution costs, then the problem of calculating e_i^{min} is equivalent to the problem of *minimum makespan scheduling* [10], where we treat each thread in a segment as an independent job and seek to obtain the minimum completion time for executing all such jobs on m processors. This gives us per segment best-case execution times, which can be summed to yield e_i^{min}. Unfortunately, this problem has been proven to be NP-hard [10]. This problem can be solved using a classical dynamic programming-based algorithm [10], which has exponential time complexity with respect to the per segment thread count. However, for some special cases where certain restrictions on the task model apply, we can easily calculate e_i^{min} in linear time. For example, when $v_i^{max} \leqslant m$ holds, $e_i^{min} = \sum_{j=1}^{s_i} e_i^{j,1}$ since in this case all threads of each segment of τ_i can be executed in parallel on m processors and thread $\tau_i^{j,1}$ has the largest execution cost in each segment τ_i^j. Moreover, when all threads in each segment have equal execution costs, $e_i^{min} = \sum_{j=1}^{s_i} \sum_{k=1}^{\lceil v_i^j/m \rceil} e_i^{j,1}$, because the execution of each segment τ_i^j can be viewed as the executions of $\lceil v_i^j/m \rceil$ sequential sub-segments, each with an equal execution cost of $e_i^{j,1}$.

Each parallel task is released repeatedly, with each such invocation called a *job*. The kth job of τ_i, denoted $\tau_{i,k}$, is released at time $r_{i,k}$. Associated with each task τ_i is a period p_i, which specifies the minimum time between two consecutive job releases of τ_i. We require $e_i^{min} \leqslant p_i$ for any task τ_i; otherwise, response times (defined next) can grow unboundedly. The utilization of a task τ_i is defined as $u_i = e_i/p_i$, and the utilization of the task system τ as $U_{sum} = \sum_{\tau_i \in \tau} u_i$. We require $U_{sum} \leqslant m$; otherwise, response times can grow unboundedly. For any job $\tau_{i,k}$ of task τ_i, its uth segment is denoted $\tau_{i,k}^u$, and the vth thread of this segment is denoted $\tau_{i,k}^{u,v}$. An example parallel task is shown in Fig. 1. For clarity, a summary of important terms defined so far, as well as some additional terms defined later, is presented in Table 1.

Successive jobs of the same task are required to execute in sequence. If a job $\tau_{i,k}$ completes at time t, then its *response time* is $t - r_{i,k}$. A task's response time is the maximum response time of any of its jobs. Note that, when a job of a task completes after the release time of the next job of that task, this release time is not altered.

Assigning deadlines and priority points. Each parallel task τ_i has a specified relative deadline of d_i, which may differ from p_i (thus, our analysis is applicable to soft real-time arbitrary-deadline sporadic parallel tasks). We do not use such deadlines in prioritizing jobs, but rather assign each job $\tau_{i,k}$ a priority point at $d_{i,k} = r_{i,k} + p_i$ and schedule jobs on a global earliest-priority-point-first (GEPPF) basis. That is, earlier priority points are prioritized over later ones.[2] We assume that ties are broken by task ID (lower IDs are favored).

3. Response time bound

We derive a response time bound for GEPPF by comparing the allocations to a task system τ in a processor sharing (PS) schedule and an actual GEPPF schedule of interest for τ, both on m

[2] GEDF becomes a special case of GEPPF when $d_i = p_i$ holds for each τ_i.