The Journal of Systems and Software xxx (2011) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Zero-laxity based real-time multiprocessor scheduling

Jinkyu Lee [a], Arvind Easwaran [b], Insik Shin [a,*], Insup Lee [c]

[a] Department of Computer Science, KAIST, South Korea
[b] Cister Research Unit, Polytechnic Institute of Porto, Portugal
[c] Department of Computer and Information Science, University of Pennsylvania, PA, USA

ARTICLE INFO

Article history:
Received 27 May 2011
Received in revised form 1 July 2011
Accepted 1 July 2011
Available online xxx

Keywords:
Real-time systems
Multiprocessor scheduling
Zero-laxity

ABSTRACT

It has been widely studied how to schedule real-time tasks on multiprocessor platforms. Several studies have developed optimal scheduling policies for implicit deadline task systems. So far however, studies have failed to develop effective scheduling strategies for more general task systems such as constrained deadline tasks. We argue that a narrow focus on deadline satisfaction (urgency) is the primary reason for this lack of success. In particular, few studies have considered the impact on scheduling of the restriction that a job cannot simultaneously execute on more than one core (parallelism). In this paper we look at one such simple, but effective, characterization of urgency and parallelism – the zero laxity first policy (ZL policy). We study in detail how beneficial the ZL policy is to schedulability. We then develop an improved schedulability test for any algorithm that employs the ZL policy, and prove that the test dominates previously known tests. Our simulation results show that the improved ZL schedulability test outperforms the existing ones.

1. Introduction

Real-time scheduling theories have been studied for achieving predictability on satisfying timing constraints. Since 1970s, scheduling algorithms for uniprocessor systems have been extensively studied. Based on a comprehensive understanding of how task deadlines affect schedulability, Earliest Deadline First (EDF) (Liu and Layland, 1973) was developed as an optimal scheduling algorithm. While uniprocessor scheduling has successfully matured over years, the same cannot be said about scheduling theory for multi-cores (multiprocessors).

Some multiprocessor studies in the past (e.g., Cho et al., 2002; Srinivasan and Baruah, 2002; Andersson et al., 2001) have focused on adapting existing uniprocessor scheduling to multiprocessors, and some others have developed novel policies specific to multiprocessors (e.g., Cheng et al., 1997; Baruah et al., 1996; Cho et al., 2006; Anderson and Srinivasan, 2000; Andersson and Tovar, 2006; Funaoka et al., 2008; Andersson and Bletsas, 2008; Easwaran et al., 2009; Levin et al., 2010; Lee et al., 2011; Stavrinides and Karatza, 2011). In spite of some significant achievements of these studies, many important scheduling problems continue to pose challenges, including the efficient scheduling of general task systems such as those in which task deadlines differ from their periods. We believe that one of the primary reasons for this lack of success is the sole focus on deadline satisfaction (or "urgency") by these existing approaches. When a task cannot be simultaneously scheduled on more than one processor at the same time ("parallelism" restriction), it becomes equally important to consider task "parallelism" when assigning priorities to tasks. Otherwise, there can exist only a few tasks (less than the number of processors) with long remaining execution times, instead of many tasks (no less than the number of processors) with short remaining execution times, and this entails that some tasks fail to meet their deadlines.

In this paper we consider two job parameters to quantify the notions of urgency and parallelism at any time instant t: (1) remaining time to deadline of a job ($D(t)$) for urgency, and (2) remaining execution time of a job ($C(t)$) for parallelism. It is intuitive that $D(t)$ captures the notion of urgency. To understand how $C(t)$ captures parallelism, consider two jobs J_1 and J_2 such that J_1 has a larger $C(t)$ in comparison to J_2. Then, by scheduling J_1 ahead of J_2 we can ensure that the number of unfinished jobs at the next time instant is maximized. This in turn implies that the jobs can use processing capacity with more parallelism than otherwise. We prove that $C(t)$ is optimal in terms of parallelism in Section 3.1, where we consider an algorithm that assigns job priorities based solely on $C(t)$.

One of the simple but effective ways to consider both urgency and parallelism is to assign the highest priority to any zero or negative laxity job, where the laxity of a job is defined as $D(t) - C(t)$. We denote this policy as the *ZL policy*, and any work-conserving, preemptive scheduling algorithm that employs this policy as a ZL-based scheduling algorithm. By work-conserving, we mean an

* Corresponding author.
E-mail addresses: jinkyu@cps.kaist.ac.kr (J. Lee), aen@isep.ipp.pt (A. Easwaran), insik.shin@cs.kaist.ac.kr (I. Shin), lee@cis.upenn.edu (I. Lee).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.07.002

algorithm that always schedules any unfinished, ready-to-execute task if there are available processors. Once the ZL policy is incorporated into a scheduling algorithm, the ZL-based scheduling algorithm assigns the highest priority to any zero or negative laxity job and then prioritizes the remaining jobs based on the policy of the original algorithm.

The ZL policy has been effectively used in global EDZL (Earliest Deadline first until Zero Laxity) scheduling (Lee, 1994; Cho et al., 2002), which assigns the highest priority to zero or negative laxity jobs (the ZL policy) and then uses EDF for remaining jobs. It has been shown that EDZL dominates[1] EDF (Park et al., 2005), and further it has also been observed in simulations that EDZL is more efficient (being able to schedule more task sets) in scheduling general task systems than many other algorithms (Cho et al., 2002). To generalize such a dominance relationship between EDF and EDZL, we present and prove a beneficial property of the ZL policy to schedulability: any work conserving, preemptive algorithm employing the ZL policy dominates the original algorithm itself. An obvious conclusion to draw here is that the ZL policy, although a simple technique for considering urgency and parallelism together, is in fact quite effective in handling general task systems on multiprocessors. Therefore it is interesting, and hence the focus of this paper, to study the ZL policy in detail and investigate the general family of ZL-based algorithms.

Although just incorporating the ZL policy is sufficient to improve the schedulability of any work-conserving algorithm, the true potential of the policy will only be realized when a corresponding improvement in the schedulability tests is achieved. Hence, based on an observation of deadline miss under the ZL policy, we develop a new ZL-specific schedulability test. Under this policy, deadline miss occurs only when there are at least $m+1$ tasks with zero or negative laxity *at the same time*, where m denotes the number of processors in the platform. Although there exist schedulability tests for any ZL-based algorithm (Theorem 1 in Lee et al., 2010) and for EDZL (Theorem 7 in Baker et al., 2008), to the best of our knowledge, they do not utilize the "at the same time" property of the above observation. In this paper, we show that, by characterizing this property, it is possible to obtain a new schedulability test for any ZL-based algorithm that dominates the previously known tests.

1.1. Contribution

The contributions of this paper are two-fold. First, we analyze the ZL policy such that it not only serves as a simple yet good example for the effectiveness of considering urgency and parallelism together, but also has a property that any work-conserving, preemptive algorithm employing the ZL policy dominates the original algorithm (Section 3). Second, we develop an improved test using the "at the same time" property described in the previous paragraph (Section 4), and perform simulations to show the performance of the proposed schedulability test (Section 5).

2. System model

2.1. Task model

In this paper we assume a sporadic task model (Mok, 1983). In this model, a task $\tau_i \in \mathcal{T}$ is specified as (T_i, C_i, D_i), where T_i is the minimum separation, C_i is the worst-case execution time requirement, and D_i is the relative deadline. Further, we focus our attention on implicit ($C_i \le D_i = T_i$) and constrained ($C_i \le D_i \le T_i$) deadline task systems. A task τ_i invokes a series of jobs, each separated from its predecessor by at least T_i time units. We also assume that a single job of a task cannot be executed in parallel. We let n denote the total number of tasks in the system.

In this paper we assume quantum-based time and without loss of generality let one time unit denote the quantum length. All task parameters are then specified as multiples of this quantum length.

In constrained deadline task systems, at most one active job per task exists in any time slot, and hence, for simplicity of presentation, we use the term "task" also to refer to "active job of a task" in the rest of this paper. We use $D_i(t)$ and $C_i(t)$ to denote the remaining time to deadline and the remaining execution time, respectively, of a job of τ_i at time t. We express that a job of τ_i is *active* at t when $C_i(t)$ is non-zero. Also, we define $D_i(t) - C_i(t)$ as the *laxity* of task τ_i at time instant t.

2.2. Multiprocessor platform

We assume that the platform is comprised of m identical unit-capacity processors, and therefore restrict the system static-utilization U_{sys} to at most m. It has been previously shown that $U_{sys} \le m$ is a necessary condition for feasibility of the task system on m processors (Baker and Cirinei, 2006).

Like most existing studies in multiprocessor scheduling (for example, see Baruah et al., 1996), we assume that the system does not incur any penalty when a job is preempted or when a job is migrated from one processor to another.

2.3. Scheduling algorithm

A global scheduling algorithm, which maintains a global ready-queue, can preempt a job on one processor and later resume it on any other processor in the platform. In this paper we only consider global scheduling algorithms, and therefore, hereafter do not explicitly use the term "global" for each one. In addition, we are only interested in priority-driven, work-conserving, preemptive scheduling algorithms so that we do not explicitly use the term "priority driven, work-conserving, preemptive" for each one as well in this paper.

3. Analysis of the ZL policy

In this section, we analyze the ZL policy. We first discuss the relevance of two properties of real-time jobs in multiprocessor scheduling: "urgency" characterized by the remaining time to deadline of the job ($D(t)$) and "parallelism" characterized by the remaining execution time of the job ($C(t)$). Then, we present that the ZL policy is not only an effective technique to consider both $D(t)$ and $C(t)$, but also has an important dominance property.

3.1. Job parameters: urgency and parallelism

One of the primary objectives in any hard-real-time setting is to meet job deadlines, and therefore urgency is an important property to consider when scheduling jobs. This has indeed been the case in almost all scheduling theory research on single processor systems (Baruah et al., 1990; Liu and Layland, 1973), and many of the studies on multiprocessor scheduling (Baruah et al., 1996; Andersson et al., 2001; Bertogna et al., 2005; Andersson and Bletsas, 2008; Easwaran et al., 2009; Anderson and Srinivasan, 2000). These multiprocessor studies have resulted in many scheduling algorithms such as PFair (Baruah et al., 1996), fixed-priority (Andersson et al., 2001), EDF (Bertogna et al., 2005), task-splitting (Anderson et al., 2005; Andersson and Bletsas, 2008) and virtual clustering (Easwaran et al., 2009).

[1] Algorithm *A dominates* or is *tighter* than Algorithm *B* if any task set schedulable by Algorithm *B* is also schedulable by Algorithm *A*, but the reverse is not true. Such a definition of dominance or tightness can also be applied to schedulability tests.

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

Aforementioned studies either do not explicitly consider jobs parameters related to parallelism, or avoid the issue of parallelism by splitting tasks into subtasks with unit execution time (e.g., PFair). Parallelism refers to the restriction in our task model that multiple executions of a job cannot be scheduled on more than one processor at the same time. We claim, and demonstrate in the following paragraphs, that the job parameter $C(t)$ is an effective characterization of the parallelism restriction on multiprocessor platforms.

Different from uniprocessor systems, parallelism plays an important role in multiprocessor systems. To substantiate this argument, we now consider the well-studied scheduling algorithm EDF. Although this algorithm is known to be optimal for single processor systems, it is not optimal on multiprocessors (Baker, 2005). In the following paragraph we show that EDF is optimal even on multiprocessors if we remove the parallelism restriction. This suggests that the inefficiency is mainly because parallelism restriction is not considered when assigning priorities to jobs. Note that although this optimality result of EDF is known, we document it here for completeness.

We now prove the optimality of EDF when multiple executions of the same job can be scheduled on more than one processor at the same time. Under these assumptions of EDF scheduling and no parallelism restriction, the following two job sets are equivalent from a scheduling view-point: (1) a job set $\mathcal{J}$ comprised of, among others, a job J_1 with release time r, execution requirement c and relative deadline d, and (2) a job set $\mathcal{J}'$ identical to $\mathcal{J}$ in all respects, except job J_1 is replaced with c jobs each with release time r, execution requirement 1 and relative deadline d. This follows from the trivial fact that at each time instant when EDF schedules i executions of job J_1 in the former case, it will also schedule i out-of c jobs in the latter case. We record this urgency optimality of EDF in the following lemma.

Lemma 1 (Urgency optimal). *Consider a set $\mathcal{J}$ of jobs with release times and deadlines, where each job has one unit of execution time. This job set is feasible if and only if it is schedulable under EDF.*

Proof. ($\Leftarrow$): Trivial.

($\Rightarrow$): Suppose a job J_1 misses its deadline at time t when $\mathcal{J}$ is scheduled under EDF. Further, let t denote the earliest such time instant. Eliminate all jobs in the schedule with deadline greater than t. Note that these jobs have no impact on the schedule of jobs with deadline at most time t. That is, job J_1 will still miss its deadline in the new schedule. Now consider the time instant t' in the new schedule such that 1) $t' < t$, 2) all m processors are always busy in the interval $(t', t]$, and 3) either $t' = 0$ or at least one processor is idle in the interval $(t'-1, t']$. Consider all jobs scheduled in the interval $(t', t]$. All of these jobs have deadline at most t and are released at or after time t'. If they were released earlier, then they would have been scheduled in the interval $(t'-1, t']$. Then the total workload in the interval $(t', t]$ is strictly greater than the total available processing capacity $m(t-t')$. Therefore job set $\mathcal{J}$ is infeasible. □

Having ascertained the importance of parallelism in assigning job priorities, we now turn to the question of "Which job parameter should be used to characterize the parallelism restriction?". In this paper we consider the remaining execution time of a job $C(t)$ for this purpose. In order to justify this choice, we focus on Largest Execution First (LEF) scheduling algorithm (Easwaran et al., 2008). At any time instant, LEF prioritizes jobs based on their remaining execution time; the larger the remaining execution time, the higher will be the job priority. In the following lemma we prove that the overall makespan of a non-real-time job set is minimized when it is scheduled under LEF. In other words, we show that when deadlines do not matter, LEF optimally utilizes the m processors by minimizing the makespan of the job set.

Lemma 2 (Parallelism optimal). *Given a set of jobs with release and execution times and infinite deadline, the latest finish time of this job set (makespan) on m processors is minimized when the jobs are scheduled by LEF.*

Proof. We prove this theorem by contradiction. Suppose the makespan can be shortened. This implies that there exists one or more idle instants in the schedule (time instants at which less than m jobs are scheduled). Further, in order to shorten makespan, it must be possible to delay at least one of these idle instants. That is, there exists an idle instant such that the total workload scheduled up to that instant is lower under LEF, when compared to a makespan optimal schedule. Let t denote the earliest such idle instant. That is, in the interval $(t, t+1]$ less than m jobs are scheduled. Further, let $\mathcal{J} = \{J_1, \ldots, J_i\}$ $(i < m)$ denote the jobs scheduled in $(t, t+1]$ such that each job in this set has non-zero remaining executions at $t+1$. Note that if no such job exists, then the idle instant at t cannot be delayed and this contradicts our assumption.

Let t' denote the latest time instant before t at which at least one job from $\mathcal{J}$ is not scheduled although it is active. Without loss of generality, let J_1 denote that job. That is, at all times in the interval $(t'+1, t+1]$ each job in $\mathcal{J}$ is either scheduled or inactive. Note that if such a t' does not exist, then the idle instant at t cannot be postponed and this again contradicts our assumption.

Now, in the interval $(t', t'+1]$, J_1 is not scheduled although it is active and therefore m other jobs must be scheduled. We claim that out of these m jobs there exists at least one job J' such that $J' \notin \mathcal{J}$ and J' finishes its entire execution by time t. We now prove this claim. Clearly, since $|\mathcal{J}| < m$, at least some jobs scheduled in the interval do not belong to $\mathcal{J}$. Suppose there are k such jobs and suppose all these jobs execute until $t+1$. Then in the interval $(t', t+1]$ the following statements hold: 1) total available processing capacity is $m(t+1-t')$, 2) total workload scheduled from $\mathcal{J}$ is $(m-k)(t+1-t')$ plus at least one execution of J_1, and 3) total remaining workload scheduled is $k(t+1-t')$. This is clearly not feasible. Therefore our claim is true.

At t' the remaining execution time of J_1 is strictly greater than $t+1-(t'+1)$, because by definition it is always scheduled in the interval $(t'+1, t+1]$ and it has remaining executions at $t+1$. However, the remaining execution time of J' at t' is at most $t-t'$. This contradicts our assumption that jobs are scheduled using LEF, because J_1 has higher priority than J' at t' under LEF. □

The above lemma proves that LEF minimizes makespan, and this also means LEF maximizes the number of active jobs. Then, LEF, prioritizing jobs with larger $C(t)$, enables jobs to execute concurrently since the number of active jobs are maximized, and thereby parallelism restriction does not matter. Therefore, the job parameter $C(t)$ can capture the notion of parallelism.

Some studies in the past have simultaneously considered both urgency and parallelism when assigning priorities to jobs. To the best of our knowledge, these studies include the Earliest Deadline first until Zero Laxity (EDZL) (Lee, 1994; Cho et al., 2002), Least Laxity First (LLF) (Leung, 1989), Dynamic Density First (DDF) (Lee et al., 2010), and Fixed Priority until Zero Laxity (FPZL) (Davis and Burns, 2011) scheduling algorithms. A property common to all these algorithms is that they assign the highest priority to any zero or negative laxity job (the ZL policy). In the next subsection, we present and prove that the ZL policy has a dominance property, which is beneficial to schedulability.

3.2. The dominance property of the ZL policy

As we mentioned in the introduction, the ZL policy can be incorporated into any work-conserving, preemptive algorithm. We denote a base algorithm A employing the ZL policy as AZL. That is, AZL assigns the highest priority to any zero or negative laxity job

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

(with arbitrary tie-breaking), and then prioritizes the remaining jobs based on the policy of *A*. *AZL* is said to dominate *A*, if every task set that can be scheduled by *A* is also schedulable by *AZL*. Such a dominance relationship demonstrates the impact of the ZL policy on schedulability, and is therefore useful to establish. Such a relationship has already been proved between EDZL and EDF (Park et al., 2005), and in the following theorem, we generalize this result to any scheduling algorithm *A*.

Theorem 1. *If a task set is schedulable by Algorithm A , it is also schedulable by AZL.*

Proof. The proof is similar to Park et al. (2005). Let $\sigma_A(t) = \{j_1, j_2, \ldots, j_m\}$ be a set of jobs which have the m highest priority under Algorithm *A* at time t.

Suppose that there is a job J_1 with zero laxity at t, but it is not in $\sigma_A(t)$. Then, J_1 will eventually miss its deadline by Algorithm *A* since after t its remaining execution time will be strictly larger than time to its deadline. We conclude if the task set is schedulable by Algorithm *A*, any jobs with zero laxity at t is in $\sigma_A(t)$. Algorithm *AZL* is different from Algorithm *A* in that it gives the highest priority to jobs with zero laxity. Therefore, we conclude that $\sigma_A(t) = \sigma_{AZL}(t)$ for all t if a task set is schedulable by Algorithm *A*. This proves the theorem. □

The dominance of *AZL* over *A* implies that any schedulability test for *A* can also be used for *AZL* without modification. We record this result in the following theorem.

Theorem 2. *Any schedulability test for Algorithm A can also be applied to AZL.*

Proof. Immediately follows from Theorem 1. □

Theorem 1 indicates that the ZL policy is beneficial to the schedulability of base algorithms. While Theorem 2 enables us to apply base-algorithm schedulability tests for the corresponding ZL-based algorithm, such tests do not use any characteristic of the ZL policy itself. In the next section, using a property of deadline miss under the ZL policy, we develop a new schedulability test for any ZL-based algorithm.

4. The improved ZL schedulability analysis

In this section, we first derive a condition for deadline miss under the ZL policy. Based on this condition, we then develop a new schedulability test for any ZL-based algorithm. Finally, we compare our test with the existing schedulability test for any ZL-based algorithm (Lee et al., 2010).

4.1. The condition of deadline miss under any ZL-based algorithm

Suppose there is a task that misses a deadline at t_2, and let t_1 ($\leq t_2$) denote the first time instant before the deadline miss when there is a task with negative laxity. Then, there must be more than m tasks with zero or negative laxity at $t_1 - 1$, and we present this observation generally as follows:

Observation 1. *When a task misses its deadline under any ZL-based algorithm, there are at least $m+1$ tasks which have zero or negative laxity at the same time before the deadline miss.*

According to Observation 1, we know that for a deadline miss to occur in any ZL-based algorithm there must exist tasks $\mathcal{T}' = \{\tau_1, \ldots, \tau_{n'}\}$, such that $n' \geq m+1$ and all the following properties hold.

1. For each task $\tau_k \in \mathcal{T}'$, τ_k has zero or negative laxity within D_k time units from its release.
2. There exists a time instant t_0 such that for each task $\tau_k \in \mathcal{T}'$,
   (a) τ_k has zero or negative laxity at t_0;
   (b) τ_k has at least one unfinished execution at t_0 (i.e., $C_k(t_0) \geq 1$); and
   (c) τ_k has higher priority than any task $\tau_i \notin \mathcal{T}'$ in the interval $[t_0, t_0 + 1)$.

Property 1 is identical to Observation 1 except the "at the same time" part of the observation, and hence it is a necessary condition of the observation. Property 2a paraphrases Observation 1 using a specific time instant which satisfies the "at the same time" condition. Property 2b trivially holds since we only consider tasks that are active at t_0. This is because a task with no active job at a time instant cannot affect executions of other tasks at that instant. Property 2c comes from the ZL policy: a task with zero or negative laxity has higher priority than tasks with positive laxity.

Properties 2a, 2b and 2c together capture Observation 1 including the *at the same time* constraint. Since the state-of-the-art schedulability tests (for any ZL-based algorithm Lee et al., 2010 and for EDZL Baker et al., 2008) only capture Property 1, they fail to consider Observation 1 in its entirety. In the following subsection we develop a new ZL schedulability test that captures all the above properties, and then compare it to the existing tests in Section 4.3.

4.2. A new ZL schedulability test

To check whether a task τ_k can have zero or negative laxity, existing approaches (Baker et al., 2008; Lee et al., 2010) have used the concept of the worst-case interference of higher-priority tasks on τ_k between its release and deadline. Following the notations in these studies, we denote the total interference of a task τ_i on a task τ_k in an interval $[a, b)$ as $\bar{I}_{k,i}(a, b)$. It represents the cumulative length of all intervals within $[a, b)$ in which τ_k is ready to execute and τ_i is executing while τ_k is not. The worst-case interference of task τ_k from task τ_i in any interval of length l is then defined as

$$I_{k,i}(l) = \max_{t_0} \bar{I}_{k,i}(t_0, t_0 + l), \tag{1}$$

and the overall worst-case higher priority interference is defined as

$$\sum_{i \neq k} I_{k,i}(l). \tag{2}$$

Note that the above equation over-estimates interference, because it does not consider the fact that the worst-case interference scenario for each task may occur in different time intervals. It is known that computing $I_{k,i}(l)$ precisely is computationally intractable, and therefore existing approaches have used an upper bound that is valid for any work-conserving scheduling algorithm (Bertogna and Cirinei, 2007; Bertogna et al., 2009). These studies describe the pattern of a job release corresponding to the largest workload of a task τ_i that can interfere with a task τ_k. This worst-case interference pattern is depicted in Fig. 1. Given an interval

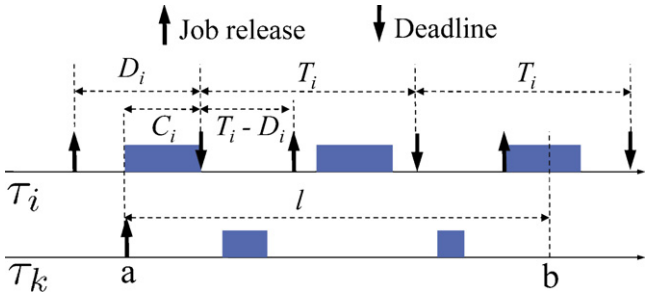

Fig. 1. The situation where the maximum interference occurs under any work-conserving algorithm.

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

$[a, b)$ of length l, the first job of τ_i begins its execution at a and completes the execution at $a + C_i$. Here $a + C_i$ is also the deadline of the first job. Thereafter, jobs are released and scheduled as soon as possible. We denote by $\eta_i(l)$ the number of jobs of τ_i that can execute completely within the interval of interest (including the first job).

$$\eta_i(l) = \left\lfloor \frac{l - (C_i + T_i - D_i)}{T_i} \right\rfloor + 1 = \left\lfloor \frac{l + D_i - C_i}{T_i} \right\rfloor \quad (3)$$

The contribution of the last job can then be bounded by $\min(C_i, l + D_i - C_i - \eta_i(l) \cdot T_i)$. The maximum interference of a task τ_i on a task τ_k during an interval of length l under any work-conserving scheduling algorithm (denoted by $I^{WC}_{k,i}(l)$) is therefore

$$I^{WC}_{k,i}(l) = \eta_i(l) \cdot C_i + \min(C_i, l + D_i - C_i - \eta_i(l) \cdot T_i) \quad (4)$$

Using $I^{WC}_{k,i}(l)$, we now develop a new schedulability test for any ZL-based algorithm.

According to Observation 1, the task set $\mathcal{T}$ described in Section 4.1 must have at least $m+1$ tasks in order for a deadline miss to occur under the ZL policy. We now focus on each task $\tau_k \in \mathcal{T}$, and try to see what inequalities concerning interference should be satisfied for τ_k to belong to $\mathcal{T}$. As discussed above, the amount of interference of a task τ_i on a task τ_k during an interval of length D_k is upper-bounded by $I^{WC}_{k,i}(D_k)$. This interference function however, does not consider any of the properties associated with the ZL policy; in particular, it does not consider the properties described in Section 4.1. We now look at how each property reduces the interference of τ_i on τ_k.

We first focus on the impact of Property 2a on the interference generated by a task $\tau_i \in \mathcal{T}$ on task $\tau_k \in \mathcal{T}$. Suppose that both τ_k and τ_i have zero or negative laxity at $b - \epsilon$ ($\epsilon \geq 0$), where the release time instant and deadline of τ_k are a and b, respectively, as shown in Fig. 2. Since τ_i has zero or negative laxity at $b - \epsilon$, it is not executed at least during $D_i - C_i$ time units from its release time to the time instant $b - \epsilon$. Thus, given $b - \epsilon$, when both τ_k and τ_i have zero or negative laxity, the worst-case interference pattern of τ_i on τ_k occurs when deadline of τ_i is aligned to $b - \epsilon$ as shown in Fig. 2. Then, we can derive the following interference bound function for $\tau_i \in \mathcal{T}$ during an interval $[a, b - \epsilon)$ of length $l = D_k - \epsilon$ (denoted by $I^{ZL}_{k,i}(l)$):

$$I^{ZL}_{k,i}(l) = \left\lfloor \frac{l}{T_i} \right\rfloor C_i + \min\left(C_i, l - \left\lfloor \frac{l}{T_i} \right\rfloor T_i\right), \quad (5)$$

Then, we represent the upper-bound of the interference of $\tau_i \in \mathcal{T}$ on τ_k during an interval $[a, b - \epsilon)$ of length $D_k - \epsilon$ as follows:

$$I^{ZL}_{k,i}(D_k - \epsilon), \quad \text{for } \tau_i \in \mathcal{T}. \quad (6)$$

We now upper-bound the above interference for all possible values of ϵ, using the observation that the interference is maximized when $\epsilon = 0$. The resulting bound is given as follows:

$$I^{ZL}_{k,i}(D_k - \epsilon) \leq I^{ZL}_{k,i}(D_k), \quad \text{for } \tau_i \in \mathcal{T}. \quad (7)$$

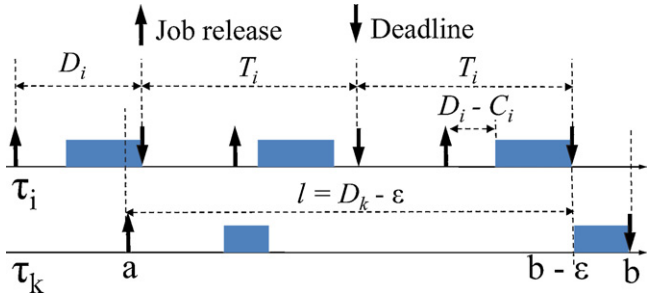

Fig. 2. The situation where deadline of τ_i is aligned at $b - \epsilon$.

Note that the above upper-bound is tighter than $I^{WC}_{k,i}(D_k)$ since it obviously holds $I^{ZL}_{k,i}(l) \leq I^{WC}_{k,i}(l)$ for any $l > 0$.

Next, we look at the impact of Property 2b on the interference of a task $\tau_i \in \mathcal{T}$ on task τ_k. Property 2b indicates that task τ_i has at least one unfinished execution after τ_k reaches zero or negative laxity and before its deadline. Therefore, at least one execution unit of τ_i can be removed from the interference, because τ_k is required to have zero or negative laxity without interference from this execution. Combining this with the interference upper-bound given in Eq. (7), we get that the interference of τ_i on τ_k is bounded by:

$$I^{ZL}_{k,i}(D_k) - 1, \quad \text{for } \tau_i \in \mathcal{T}. \quad (8)$$

We then focus on the impact of Property 2c on the interference of a task $\tau_i \notin \mathcal{T}$ on task τ_k. Since Property 2c indicates that for at least one time unit τ_k has higher priority than τ_i, the interference from τ_i on τ_k must only be considered for an interval of length $D_k - 1$. Therefore, this interference is given by:

$$I^{WC}_{k,i}(D_k - 1), \quad \text{for } \tau_i \notin \mathcal{T}. \quad (9)$$

Based on the interference of τ_i on τ_k in Eqs. (8) and (9), we derive a condition for τ_k to have zero or negative laxity under any ZL-based algorithm as follows:

Lemma 3. *Task τ_k can have zero or negative laxity under any ZL-based algorithm only if the following inequality holds:*

$$\sum_{(i \neq k) \wedge (\tau_i \notin \mathcal{T})} I^{WC}_{k,i}(D_k - 1) + \sum_{(i \neq k) \wedge (\tau_i \in \mathcal{T})} \left\{ I^{ZL}_{k,i}(D_k) - 1 \right\} \geq m \cdot (D_k - C_k) \quad (10)$$

Proof. From Eqs. (8) and (9), we know that the total interference of all the other tasks $\{\tau_i\}_{i \neq k}$ on τ_k is upper-bounded by the LHS of Eq. (10). For τ_k to have zero or negative laxity, τ_k cannot be executed during at least $D_k - C_k$ time units. At each such time slot, interference from at least m other tasks is needed to block the execution of τ_k. Hence, if the total interference of all the other tasks is less than $m \cdot (D_k - C_k)$, τ_k cannot have zero or negative laxity. □

Lemma 3 offers a necessary condition for τ_k to have zero or negative laxity, and we can obtain the following schedulability test once we apply the lemma to the deadline miss condition of Observation 1.

Lemma 4. *A task set is schedulable by any ZL-based scheduling algorithm if there are at most m different tasks τ_k satisfying Eq. (10).*

Proof. Immediately follows from Observation 1 and Lemma 3. □

To further reduce the maximum interference of any task, we can use the following lemma:

Lemma 5 (Lemma 4 in Bertogna et al., 2005).

$$\sum_{i \neq k} I^{Any}_{k,i}(l) \geq m \cdot x \;\Leftrightarrow\; \sum_{i \neq k} \min\left\{ I^{Any}_{k,i}(l), x \right\} \geq m \cdot x \quad (11)$$

where $I^{Any}_{k,i}(l)$ is any interference bound of τ_i on τ_k in an interval of length l.

Proof. The proof is given in Lemma 4 in Bertogna et al. (2005), but intuition is as follows: if the LHS of Eq. (11) holds and τ_i is executed more than x in an interval of length l, τ_i can interfere with τ_k during at most x. This is because if the amount of execution of τ_i (denoted by y) is larger than x, at least $y - x$ amount of execution of τ_i should be concurrently executed with τ_k's execution. □

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

Applying Lemma 5 to Lemma 4, and unifying the length of the interval in the interference functions, we can derive a tighter ZL schedulability test as follows.

Lemma 6. *A task set is schedulable by any ZL-based scheduling algorithm if either (A) or (B) is true:*

(A) *The following inequality holds for at most m different tasks* $\tau_k \in \mathcal{T}$:

$$\sum_{i \neq k} \min\left\{ I_{k,i}^{\mathrm{WC}^+}(D_k - 1), D_k - C_k \right\} \geq m \cdot (D_k - C_k) \tag{12}$$

(B) *The following inequality holds for at most m different tasks* $\tau_k \in \mathcal{T}$:

$$\sum_{i \neq k} \min\left\{ I_{k,i}^{\mathrm{WC}^+}(D_k), D_k - C_k + 1 \right\} \geq m \cdot (D_k - C_k + 1) \tag{13}$$

where

$$I_{k,i}^{\mathrm{WC}^+}(l = D_k - 1 \text{ or } D_k) = \begin{cases} I_{k,i}^{\mathrm{WC}}(l), & \text{if } \tau_i \notin \mathcal{T}, \\ I_{k,i}^{\mathrm{ZL}}(l), & \text{if } \tau_i \in \mathcal{T} \end{cases} \tag{14}$$

Proof. We now apply Lemma 5 to Eq. (10) in two ways. First, we can unify the interval length using the inequality $I_{k,i}^{\mathrm{ZL}}(D_k) - 1 \leq I_{k,i}^{\mathrm{ZL}}(D_k - 1)$ for each $\tau_i \in \mathcal{T}$. Then, we obtain Eq. (12) by applying Lemma 5.

Second, the number of tasks in $\mathcal{T}$ in Eq. (10) is at least $m+1$, which is at least m excluding task τ_k. Thus, the second term of the LHS of Eq. (10) is at least as much as $\sum_{(i \neq k) \wedge (\tau_i \in \mathcal{T})} I_{k,i}^{\mathrm{ZL}}(D_k) - m$. Then, using the inequality $I_{k,i}^{\mathrm{WC}}(D_k - 1) \leq I_{k,i}^{\mathrm{WC}}(D_k)$ for each $\tau_i \notin \mathcal{T}$, we can unify the interval length to D_k. Moving the term $-m$ to the RHS and then applying Lemma 5, we obtain Eq. (13). □

Since Eqs. (12) and (13) in the above theorem depend on a specific set of tasks in $\mathcal{T}$, the resulting ZL schedulability test will have exponential complexity. This is because we will be required to evaluate Eqs. (12) and (13) for all possible subsets (size $\geq m+1$) of the given task set. However, once we apply the following procedure, we can safely compute Eqs. (12) and (13) for a task τ_k, with complexity $O(n \cdot \log(n))$, which comes from sorting interference values of all tasks as follows:

1. Calculate $\min\left\{ I_{k,i}^{\mathrm{WC}}(l), l - C_k + 1 \right\}$ (denoted by $W_i(l)$) and $\min\left\{ I_{k,i}^{\mathrm{ZL}}(l), l - C_k + 1 \right\}$ (denoted by $Z_i(l)$) for all tasks τ_i ($\neq \tau_k$), where $l = D_k - 1$ for Eq. (12) and $l = D_k$ for Eq. (13).
2. Sort $(W_i(l) - Z_i(l))$ and select those tasks in which their corresponding value $(W_i(l) - Z_i(l))$ belongs to the m smallest $\{(W_i(l) - Z_i(l))\}_{i \neq k}$ values. Denote this set of tasks by $\mathcal{T}_k^*(l, \mathrm{WC})$. We can formally express $\mathcal{T}_k^*(l, \mathrm{WC})$ as follows:

$$\mathcal{T}_k^*(l, \mathrm{WC}) = \{\tau_i | \tau_i (\neq \tau_k) \in m \text{ smallest } W_i(l) - Z_i(l)\} \tag{15}$$

3. If $\tau_i \notin \mathcal{T}_k^*(l, \mathrm{WC})$, $I_{k,i}^{\mathrm{WC}^+}(l) = I_{k,i}^{\mathrm{WC}}(l)$, and if $\tau_i \in \mathcal{T}_k^*(l, \mathrm{WC})$, $I_{k,i}^{\mathrm{WC}^+}(l) = I_{k,i}^{\mathrm{ZL}}(l)$.

For a task to miss its deadline, there should be at least $m+1$ zero laxity tasks (in $\mathcal{T}$) at the same time as mentioned in Observation 1. The above procedure decides whether τ_k can be one of the zero laxity tasks. To do this, we find an upper-bound of interference of all tasks $\{\tau_i\}_{i \neq k}$ on τ_k if there are at least $m+1$ zero laxity tasks including τ_k itself. When τ_i interferes with τ_k, it can have two interference functions: the smaller function $I_{k,i}^{\mathrm{ZL}}(l)$ for $\tau_i \in \mathcal{T}$, and the larger function $I_{k,i}^{\mathrm{WC}}(l)$ for $\tau_i \notin \mathcal{T}$. While we know that the number of tasks in $\mathcal{T}$ is at least $m+1$, we do not know which tasks belong to $\mathcal{T}$. Since $I_{k,i}^{\mathrm{ZL}}(l) \leq I_{k,i}^{\mathrm{WC}}(l)$, the less tasks τ_i in $\mathcal{T}$, the more the total interference on τ_k. Therefore, we only consider the case where there are $m+1$ tasks in $\mathcal{T}$ for an upper-bound of the interference of all tasks $\{\tau_i\}_{i \neq k}$ on τ_k. Then, the upper-bound is calculated when we choose tasks with the m smallest $\{(W_i(l) - Z_i(l))\}_{i \neq k}$ values as elements of $\mathcal{T}$; here we choose "m" tasks instead of "$m+1$" because we exclude task τ_k. Thus, the LHSs of Eqs. (12) and (13) under any possible choice of $\mathcal{T}$ are clearly upper-bounded by the case where $\mathcal{T}$ is equal to $\mathcal{T}_k^*(D_k - 1, \mathrm{WC})$ and $\mathcal{T}_k^*(D_k, \mathrm{WC})$, respectively. Finally, using the above procedure we can derive the following polynomial time-complexity ZL schedulability test:

Theorem 3 (A new ZL schedulability test). *A task set is schedulable by any ZL-based scheduling algorithm if either (A) or (B) is true:*

(A) *The following inequality holds for at most m different tasks* τ_k:

$$\sum_{i \neq k} \min\left\{ I_{k,i}^{\mathrm{WC}^*}(D_k - 1), D_k - C_k \right\} \geq m \cdot (D_k - C_k) \tag{16}$$

(B) *The following inequality holds for at most m different tasks* τ_k:

$$\sum_{i \neq k} \min\left\{ I_{k,i}^{\mathrm{WC}^*}(D_k), D_k - C_k + 1 \right\} \geq m \cdot (D_k - C_k + 1) \tag{17}$$

where

$$I_{k,i}^{\mathrm{WC}^*}(l = D_k - 1 \text{ or } D_k) = \begin{cases} I_{k,i}^{\mathrm{WC}}(l), & \text{if } \tau_i \notin \mathcal{T}_k^*(l, \mathrm{WC}), \\ I_{k,i}^{\mathrm{ZL}}(l), & \text{if } \tau_i \in \mathcal{T}_k^*(l, \mathrm{WC}) \end{cases} \tag{18}$$

and $\mathcal{T}_k^*(l, \mathrm{WC})$ *is defined in Eq. (15).*

Proof. This theorem safely holds from Lemma 6 and the above procedure to select $\mathcal{T}_k^*(l, \mathrm{WC})$. □

Tip for further improving the test of Theorem 3. We can improve this test by applying the theorem iteratively as follows. Once we apply Theorem 3, we know whether a task τ_i can have zero or negative laxity. From Property 2a, any task τ_i in $\mathcal{T}^*$ should be able to have zero or negative laxity. So tasks which cannot have zero or negative laxity should not be included in $\mathcal{T}_k^*(t, \mathrm{WC})$. If we apply Theorem 3 again with this constraint, we can reduce the candidate set of tasks for m smallest $(W_i(t) - Z_i(t))$ values. Thus interference can decrease further, and lead to improved schedulability. The detailed procedure is as follows:

1. For all tasks τ_k, set *isZero*[k] ← *true* and *oldIsZero*[k] ← *true*.
2. For all tasks τ_k, calculate whether τ_k satisfies Eq. (16) or (17). When the equations for τ_k are calculated, only tasks with *isZero*[i] = *true* can be included in $\mathcal{T}_k^*(t, \mathrm{WC})$. If one of the equations does not hold for τ_k, set *isZero*[k] ← *false*.
   If Theorem 3 holds, the task set is schedulable and this procedure halts.
3. If there is no task such that *isZero*[k] <> *oldIsZero*[k], then the task set is deemed unschedulable and this procedure halts.
   Otherwise, set *oldIsZero*[k] ← *isZero*[k] for all τ_k and go to Step 2.

4.3. Comparison of the existing ZL schedulability test

In this subsection, we compare our new ZL schedulability test in Theorem 3 with the existing one (Lee et al., 2010), both in terms of schedulability and time-complexity. While our new ZL schedulability test accommodates all the properties described in Section 4.1, the existing test does not capture the condition of "at the same time" in Observation 1 (it only accommodates Property 1). The existing test captures two deadline miss conditions which are necessary for Observation 1: (a) there are at least $m+1$ tasks which have zero or negative laxity; and (b) there is at least one task which has negative laxity. Note that (a) is equivalent to Observation 1 except

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

the condition of "at the same time", and (b) is true for any work-conserving scheduling algorithm. Using (a) and (b), the existing test is as follows:

Lemma 7 (Existing ZL Schedulability test Lee et al., 2010). *A task set is schedulable by any ZL-based scheduling algorithm if either (A) or (B) is true:*

(A) *The following inequality holds for at most m different tasks* τ_k:

$$\sum_{i \neq k} \min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k\right\} \geq m \cdot (D_k - C_k) \tag{19}$$

(B) *The following inequality does not hold for any task* τ_k:

$$\sum_{i \neq k} \min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k + 1\right\} \geq m \cdot (D_k - C_k + 1) \tag{20}$$

Proof. The proof is the same as that of Theorem 7 in Baker et al. (2008). To summarize, Eqs. (19) and (20) correspond to (a) and (b), respectively, and inequalities hold in a similar way to Lemma 3, and the minimum operation holds by Lemma 5. □

We formally express dominance relationship between the existing ZL schedulability test and our new one as follows:

Theorem 4. *The new ZL schedulability test in Theorem 3 dominates the existing ZL schedulability test in Lemma 7.*

Proof. If we focus on (A), the LHS of Eq. (16) is not larger than that of Eq. (19) because it holds that $I_{k,i}^{\mathrm{WC}^*}(D_k - 1) \leq I_{k,i}^{\mathrm{WC}}(D_k - 1) \leq I_{k,i}^{\mathrm{WC}}(D_k)$. On the other hand, while (B) in Theorem 3 allows at most m tasks satisfying Eq. (17), the corresponding (B) in Lemma 7 allows no task satisfying Eq. (20), and moreover the LHS of Eq. (17) is not larger than that of Eq. (20) since it holds that $I_{k,i}^{\mathrm{WC}^*}(D_k) \leq I_{k,i}^{\mathrm{WC}}(D_k)$. Thus, this theorem holds. □

When we apply the existing ZL schedulability test in Lemma 7, the calculation of the LHS of Eqs. (19) and (20) for a given task τ_k has complexity $O(n)$. We need to calculate the LHS of Eqs. (19) and (20) for all tasks in the worst case, and then the existing test has time-complexity $O(n^2)$. On the other hand, in Theorem 3, the calculation of the LHS of Eqs. (16) and (17) for a given task τ_k has complexity $O(n \cdot \log(n))$, because we need to sort values $(W_i(l) - Z_i(l))$ for all tasks τ_i ($\neq \tau_k$). Further, similar to the existing test, we need to calculate the LHS of Eqs. (16) and (17) for all tasks in the worst case, and then our improved test has time-complexity $O(n^2 \cdot \log(n))$. Thus, the improved ZL schedulability test incurs only a marginal time penalty, when compared to the existing test.

An example of the new and existing ZL schedulability tests (Theorem 3 *and* Lemma 7). For better understanding of the new and existing ZL schedulability tests, we present an example of four tasks executed on two processors: $\tau_1 = \tau_2 = (10, 2, 10)$ and $\tau_3 = \tau_4 = (5, 3, 4)$. As shown in Table 1 (See $\min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k + 1\right\}$), all tasks satisfy Eq. (20), i.e., $4+7+7 \geq 2 \times 9 = 18$ (for τ_1 and τ_2), and $2+2+2 \geq 2 \times 2 = 4$ (for τ_3 and τ_4). Similarly, we can check that all tasks satisfy Eq. (19). Therefore, the task set cannot be deemed schedulable by Lemma 7.

However, once we apply Theorem 3, τ_1 and τ_2 cannot satisfy Eq. (17). That is, the difference between $\min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k + 1\right\}$ and $\min\left\{I_{k,i}^{\mathrm{ZL}}(D_k), D_k - C_k + 1\right\}$ for $k=1$ (or 2) is 4-2, 7-6, and 7-6, respectively for $i=2$ (or 1), 3, and 4. So, only τ_3 and τ_4 are included in $\mathcal{T}_k(l, \mathrm{WC})$ for $k=1$ or 2. Then, as shown in Table 1 (See $\min\left\{I_{k,i}^{\mathrm{WC}^*}(D_k), D_k - C_k + 1\right\}$), the LHS of Eq. (17) for $k=1$ or 2 is $4+6+6=16$, which is strictly less than the RHS of the equation $2 \cdot 9 = 18$, and thus τ_1 and τ_2 cannot satisfy Eq. (17). Here τ_3 and τ_4 satisfy Eq. (17), i.e., $2+2+2 \geq 2 \times 2 = 4$. We conclude there are only two tasks (τ_3 and τ_4) which satisfy Eq. (17), and therefore the task set is deemed schedulable by Theorem 3.

Table 1
An example of calculating $I_{k,i}^{\mathrm{WC}}(D_k)$, $I_{k,i}^{\mathrm{ZL}}(D_k)$, and $I_{k,i}^{\mathrm{WC}^*}(D_k)$ in the new and existing ZL schedulability tests (Theorem 3 and Lemma 7)

$\min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k + 1\right\}$	$i=1$	$i=2$	$i=3$	$i=4$
$k=1$ $(D_k - C_k + 1 = 9)$	·	4	7	7
$k=2$ $(D_k - C_k + 1 = 9)$	4	·	7	7
$k=3$ $(D_k - C_k + 1 = 2)$	2	2	·	2
$k=4$ $(D_k - C_k + 1 = 2)$	2	2	2	·
$\min\left\{I_{k,i}^{\mathrm{ZL}}(D_k), D_k - C_k + 1\right\}$				
$k=1$ $(D_k - C_k + 1 = 9)$	·	2	6	6
$k=2$ $(D_k - C_k + 1 = 9)$	2	·	6	6
$k=3$ $(D_k - C_k + 1 = 2)$	2	2	·	2
$k=4$ $(D_k - C_k + 1 = 2)$	2	2	2	·
$min\left\{I_{k,i}^{\mathrm{WC}^*}(D_k), D_k - C_k + 1\right\}$				
$k=1$ $(D_k - C_k + 1 = 9)$	·	4	6	6
$k=2$ $(D_k - C_k + 1 = 9)$	4	·	6	6
$k=3$ $(D_k - C_k + 1 = 2)$	2	2	·	2
$k=4$ $(D_k - C_k + 1 = 2)$	2	2	2	·

4.4. Summary of existing tests for specific zero-laxity based algorithms

We have presented an improved ZL schedulability test, which can be applied to any work-conserving zero-laxity based scheduling algorithm. However, once we focus on individual ZL-based algorithms, schedulability can be improved by using characteristics of the individual algorithms. Now we briefly summarize the cutting edge of schedulability tests for individual ZL-based algorithms: EDZL (Baker et al., 2008) and FPZL (Davis and Burns, 2011).

The schedulability test for EDZL has been developed in Baker et al. (2008). In EDZL, tasks with positive laxity are prioritized according to their deadlines (i.e., EDF). For positive-laxity tasks, since a task with later deadline cannot interfere with another task with earlier deadline, the maximum interference of τ_i on τ_k occurs when the deadlines of two tasks are aligned as shown in Fig. 2 with $\epsilon = 0$. Thus, the interference bound function for positive-laxity task τ_i on τ_k during an interval of length l is the same as $I_{k,i}^{\mathrm{ZL}}(l)$. Therefore, if we replace $I_{k,i}^{\mathrm{WC}}(l)$ with $I_{k,i}^{\mathrm{ZL}}(l)$, Lemma 7 can be a schedulability test for EDZL. Such replacement can reduce interference because $I_{k,i}^{\mathrm{ZL}}(l)$ is always smaller than or equal to $I_{k,i}^{\mathrm{WC}}(l)$, and thus improves schedulability compared to the original lemma.

The schedulability test for FPZL has been presented in Davis and Burns (2011). In FPZL, a task with lower fixed-priority can interfere with tasks with higher fixed-priority only when the task with lower fixed-priority has zero or negative laxity. In turn, a lower fixed-priority task with positive laxity cannot interfere with a higher fixed-priority task. We can improve schedulability of FPZL by utilizing this property in an iterative manner. In the first round,

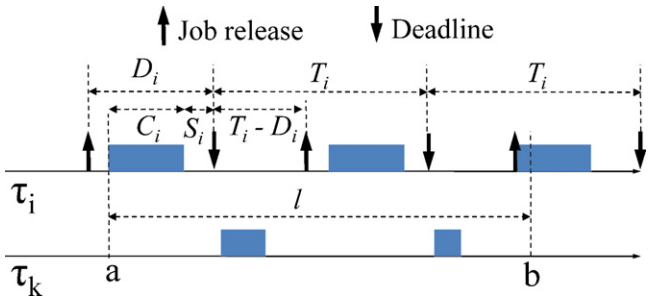

Fig. 3. The situation where the maximum interference occurs under any work-conserving algorithm when the slack value (S_i) is applied.

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

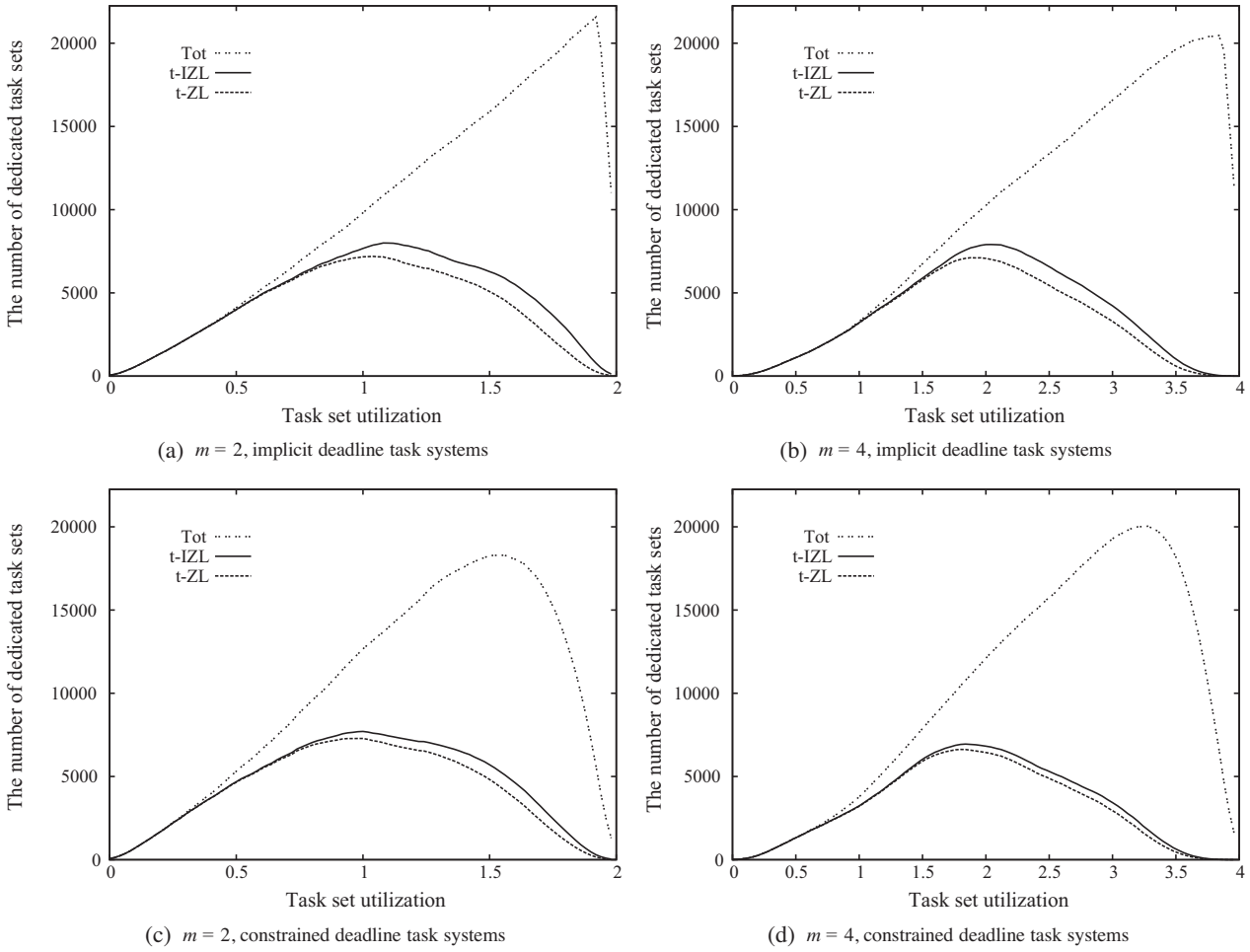

Fig. 4. Schedulability tests for t-ZL and t-IZL.

we calculate Eq. (19) for all τ_k, and record whether each task has positive laxity (i.e., violating Eq. (19) means the task has positive laxity). If Lemma 7 holds, the task set is schedulable. If not, we continue on the next round. In the round, if τ_i has lower fixed-priority than τ_k and τ_i has positive laxity, the interference of τ_i on τ_k (i.e., $\min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k\right\}$) does not count when the LHS of Eq. (19) is calculated. Here we update positive-laxity tasks using the newly calculated interference, and if there exists any updated task, we repeat the next round. If not, the iteration halts. By removing interference from lower fixed-priority tasks with positive laxity, we find more schedulable task sets FPZL.

A technique has been developed for reducing interference (Baker et al., 2008; Bertogna et al., 2009), and this technique can be also applied to the tests for ZL-based algorithms. For example, when we apply Lemma 7 for any work-conserving ZL-based algorithm, the LHS of Eq. (19) means the amount of total interference of all other tasks $\{\tau_i\}_{i \neq k}$ on τ_k during an interval of length D_k. Thus, a task τ_k can be blocked by other tasks during at most $\sum_{i \neq k} \min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k\right\}/m$, and then τ_k can finish its execution within $C_k + \sum_{i \neq k} \min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k\right\}/m$. Here we can define the slack S_k of τ_k as the minimum time interval between the finishing time and the deadline of τ_k, and the slack S_k can be calculated by $D_k - C_k - \sum_{i \neq k} \min\left\{I_{k,i}^{\mathrm{WC}}(D_k), D_k - C_k\right\}/m$. This slack value can reduce interference of τ_i on τ_k as shown in Fig. 3 (Compare the figure with Fig. 1). Then, we apply this slack value to Lemma 7 in an iterative manner. In each round, τ_k updates its slack value when Eq. (19) is calculated, and in the next round, the newly updated slack values of each task reduce interference of the task on other tasks. The iteration halts when there are no more updated slack values. This idea of using slack values is also easily applied to the schedulability tests for EDZL (Baker et al., 2008) and FPZL (Davis and Burns, 2011).

5. Performance evaluation

This section evaluates the performance of the improved ZL schedulability test.

We generate task sets based on a technique proposed earlier (Baker, 2005), which has also been used in many previous studies (e.g., see Bertogna et al., 2009; Andersson et al., 2008). We have two input parameters: (a) the number of processors m (2 or 4) and (b) the task system (constrained or implicit deadline), and (c) individual task utilization (C_i/T_i) distribution (bimodal with parameter[2]: 0.1, 0.3, 0.5, 0.7, or 0.9, or exponential with parameter[3]: 0.1, 0.3, 0.5, 0.7, or 0.9). For each task, T_i is uniformly chosen in $[1, T_{\max} = 1000]$, C_i is chosen based on the bimodal or exponential parameter, and D_i is uniformly chosen in $[C_i, T_i]$ for constrained deadline task systems or D_i is equal to T_i for implicit deadline task systems.

For each combination of (a), (b) and (c), we repeat the following procedure (from Bertogna et al., 2009) and generate 100,000 task

[2] For a given bimodal parameter p, a value for C_i/T_i is uniformly chosen in $[0, 0.5)$ with probability p, and in $[0.5, 1)$ with probability $1 - p$.

[3] For a given exponential parameter $1/\lambda$, a value for C_i/T_i is chosen according to the exponential distribution whose probability density function is $\lambda \cdot \exp(-\lambda \cdot x)$.

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

sets, thus resulting in 1,000,000 task sets for any given m and task system.

- Initially, we generate a set of $m+1$ tasks.
- In order to exclude unschedulable sets, we check whether the generated task set can pass a necessary feasibility condition (Baker and Cirinei, 2006; Baruah et al., 2009).
- If it fails to pass the feasibility test, we discard the generated task set and return to step 1. Otherwise, we include this set for evaluation. Then, this set serves as a basis for the next new set; we create a new set by adding a new task into this old set and return to step 2.

We evaluate the performance of two schedulability tests: (1) the existing ZL schedulability test in Lemma 7, and (2) the improved schedulability test in Theorem 3. These tests are respectively annotated as 't-ZL', and 't-IZL', in the figures.

In Fig. 4 we plot the number of task sets proven schedulable by each test, with total set utilization ($U_{sys} \stackrel{\text{def.}}{=} \sum_{\tau_i \in \mathcal{T}} C_i/T_i$) in $[U_{sys} - 0.01 \cdot m, U_{sys} + 0.01 \cdot m)$. Here 'Tot' means the number of task sets with each total set utilization.

Fig. 4(a) and 4(b) shows schedulability test results of implicit deadline task systems for $m=2$ and $m=4$. Here we can easily observe that 't-IZL' outperforms 't-ZL'. For $m=2$, while 't-ZL' deems 409,430 task sets schedulable among 1,000,000 task sets, 't-IZL' deems 465,117 task sets schedulable, which include 13.6% additional task sets compared to 't-ZL'. For $m=4$, 't-IZL' deems 12.8% more task sets schedulable than 't-ZL'.

Fig. 4(c) and 4(d) plot performance results of constrained deadline task systems for the cases of $m=2$ and $m=4$. Similar to implicit deadline task systems, 't-IZL' also finds a considerable portion of task sets schedulable, while 't-ZL' fail to find them schedulable.

6. Conclusion

This paper presents the ZL policy, as an initial step towards considering urgency and parallelism for assigning job priorities in multiprocessor environments. We show that, although simple, the ZL policy is nevertheless effective in scheduling constrained deadline task systems. Further, existing algorithms can be easily extended with the ZL policy, and the improved ZL schedulability test developed in this paper can be used for such modified algorithms.

In multiprocessor platforms, it is clear that the ZL policy significantly improves schedulability by prioritizing tasks with $D(t) - C(t) = 0$. However, there are few studies on how to simultaneously consider parallelism and urgency more effectively than the ZL policy. One direction of future work is to investigate such effective policies, and these studies can lay the foundation towards solving an important open problem, which is development of an optimal multiprocessor scheduling algorithm for arbitrary deadline task systems with a periodic release model (Liu and Layland, 1973). Another direction of future work is to identify some properties of our ZL schedulability analysis (e.g., sustainability Burns and Baruah, 2008) and extend the analysis towards other systems (e.g., time-triggered embedded systems Kopetz, 2008).

Acknowledgements

This work was supported in part by the IT R&D Program of MKE/KEIT [2011-KI002090, Development of Technology Base for Trustworthy Computing], Basic Research Laboratory (BRL) Program (2009-0086964), Basic Science Research Program (2011-0005541), and the Personal Plug&Play DigiCar Research Center (NCRC, 2011-0018245) through the National Research Foundation of Korea (NRF) funded by the Korea Government (MEST), and KAIST-Microsoft Research Collaboration Center.

This work was also partially funded by the Portuguese Science and Technology Foundation (FCT), the European Commission (ARTISTDesign), the ARTEMIS-JU (RECOMP), and the Luso-American Development Foundation (FLAD).

References

Anderson, J.H., Srinivasan, A., 2000. Early-release fair scheduling. In: Proceedings of Euromicro Conference on Real-Time Systems, pp. 35–43.
Anderson, J.H., Bud, V., Devi, U., 2005. An EDF-based scheduling algorithm for multiprocessor soft real-time systems. In: Proceedings of Euromicro Conference on Real-Time Systems, pp. 199–208.
Andersson, B., Bletsas, K., 2008. Sporadic multiprocessor scheduling with few preemptions. In: Proceedings of Euromicro Conference on Real-Time Systems, pp. 243–252.
Andersson, B., Tovar, E., 2006. Multiprocessor scheduling with few preemptions. In: Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 322–334.
Andersson, B., Baruah, S., Jonsson, J., 2001. Static-priority scheduling on multiprocessors. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 193–202.
Andersson, B., Bletsas, K., Baruah, S., 2008. Scheduling arbitrary-deadline sporadic task systems on multiprocessor. In: Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 197–206.
Baker, T.P., 2005. Comparison of empirical success rates of global vs. paritioned fixed-priority and edf scheduling for hand real time, Tech. Rep. Technical Report TR-050601, Dept. of Computer Science, Florida State University, Tallahasee.
Baker, T.P., Cirinei, M., 2006. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 178–190.
Baker, T.P., Cirinei, M., Bertogna, M., 2008. EDZL scheduling analysis. Real-Time Systems 40, 264–289.
Baruah, S., Mok, A., Rosier, L., 1990. Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 182–190.
Baruah, S., Cohen, N.K., Plaxton, C.G., Varvel, D.A., 1996. Proportionate progress: a notion of fairness in resource allocation. Algorithmica 15 (6), 600–625.
Baruah, S., Bonifaci, V., Marchetti-Spaccamela, A., Stiller, S., 2009. Implementation of a speedup-optimal global EDF schedulability test. In: Proceedings of Euromicro Conference on Real-Time Systems, pp. 259–268.
Bertogna, M., Cirinei, M., 2007. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 149–160.
Bertogna, M., Cirinei, M., Lipari, G., 2005. Improved schedulability analysis of EDF on multiprocessor platforms. In: Proceedings of Euromicro Conference on Real-Time Systems, pp. 209–218.
Bertogna, M., Cirinei, M., Lipari, G., 2009. Schedulability analysis of global scheduling algorithms on multiprocessor platforms. IEEE Transactions on Parallel and Distributed Systems 20, 553–566.
Burns, A., Baruah, S., 2008. Sustainability in real-time scheduling. Journal of Computing Science and Engineering 2 (1), 74–97.
Cheng, B.-C., Stoyenko, A., Marlowe, T., Baruah, S., 1997. LSTF: A new scheduling policy for complex real-time tasks in multiple processor systems. Automatica 33 (5), 921–926.
Cho, S., Lee, S.-K., Ahn, S., Lin, K.-J., 2002. Efficient real-time scheduling algorithms for multiprocessor systems. IEICE Trans. on Communications E85–B (12), 2859–2867.
Cho, H., Ravindran, B., Jensen, E.D., 2006. An optimal real-time scheduling algorithm for multiprocessors. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 101–110.
Davis, R.I., Burns, A., 2011. FPZL schedulability analysis. In: Proceedings of IEEE Real-Time Technology and Applications Symposium, pp. 245–256.
Easwaran, A., Shin, I., Lee, I., 2008. Towards optimal multiprocessor scheduling for arbitrary deadline tasks. In: Proceedings of the Work-in-Progress Session of IEEE Real-Time Systems Symposium, pp. 1–4.
Easwaran, A., Shin, I., Lee, I., 2009. Optimal virtual cluster-based multiprocessor scheduling. Real-Time Systems 43 (1), 25–59.
Funaoka, K., Kato, S., Yamasaki, N., 2008. Work-conserving optimal real-time scheduling on multiprocessors. In: Proceedings of Euromicro Conference on Real-Time Systems, pp. 13–22.
Kopetz, H., 2008. On the design of distributed time-triggered embedded systems. Journal of Computing Science and Engineering 2 (4), 340–356.
Lee, S.K., 1994. On-line multiprocessor scheduling algorithms for real-time tasks. In: IEEE Region 10's Ninth Annual International Conference, pp. 607–611.
Lee, J., Easwaran, A., Shin, I., 2010. LLF schedulability analysis on multiprocessor platforms. In: Proceedings of IEEE Real-Time Systems Symposium, pp. 25–36.
Lee, J., Easwaran, A., Shin, I., Lee, I., 2010. Multiprocessor real-time scheduling considering concurrency and urgency. ACM SIGBED Review 7 (1).
Lee, J., Easwaran, A., Shin, I., 2011. Maximizing contention-free executions in multiprocessor scheduling. In: Proceedings of IEEE Real-Time Technology and Applications Symposium, pp. 235–244.

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002

Leung, J.Y.-T., 1989. A new algorithm for scheduling periodic, real-time tasks. Algorithmica 4, 209–219.
Levin, G., Funk, S., Sadowski, C., Pye, I., Brandt, S., 2010. DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In: Proceedings of Euromicro Conference on Real-Time Systems, pp. 3–13.
Liu, C., Layland, J., 1973. Scheduling algorithms for multi-programming in a hard-real-time environment. Journal of the ACM 20 (1), 46–61.
Mok, A., 1983. Fundamental design problems of distributed systems for the hard-real-time environment, Ph.D. thesis, Massachusetts Institute of Technology.
Park, M., Han, S., Kim, H., Cho, S., Cho, Y., 2005. Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor. IEICE Transaction on Information and Systems E88-D, 658–661.
Srinivasan, A., Baruah, S., 2002. Deadline-based scheduling of periodic task systems on multiprocessors. Information Processing Letters 84 (2), 93–98.
Stavrinides, G.L., Karatza, H.D., 2011. Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques. Simulation Modelling Practice and Theory 19 (1), 540–552.

Jinkyu Lee received B.S. and M.S. degrees in computer science in 2004 and 2006, respectively, from KAIST (Korea Advanced Institute of Science and Technology), South Korea. He also received a Ph.D. in Computer Science from KAIST in 2011. His research interests include real-time embedded systems and computer networks. He won the best student paper award from the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011.

Arvind Easwaran received a Ph.D. from the University of Pennsylvania, USA, in 2008 on Advances in Hierarchical Real-Time Systems: Incrementality, Optimality, and Multiprocessor Clustering. From January 2009 to October 2010, he was a Invited Scientist in CISTER lab, at the Polytechnic Institute of Porto, Portugal. Since November 2010, he is working as a R&D Scientist in Honeywell Aerospace, Advanced Technology. He was nominated for the best paper award in Euromicro Conference on Real-Time Systems (ECRTS) in 2008 and won the best student paper award in IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) 2011. His research interests lie in real-time embedded systems.

Insik Shin is an assistant professor in Dept. of Computer Science at KAIST, South Korea, since 2008. He received a Ph.D. from University of Pennsylvania in 2006. He has been a post-doctoral research fellow at Malardalen University, Sweden, and a visiting scholar at University of Illinois, Urbana-Champaign until 2008. His research interests lie in cyber-physical systems and real-time embedded systems. He is currently a member of Editorial Boards of Journal of Computer Science and Engineering. He has served various program committees in real-time embedded systems. He received Best Paper Award from IEEE Real-Time Systems Symposium (RTSS) in 2003 and Best Student Paper Award from IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) in 2011.

Insup Lee is Cecilia Fitler Moore Professor of Computer and Information Science and Director of PRECISE Center at the University of Pennsylvania. He holds a secondary appointment in the Department of Electrical and Systems Engineering. His research interests include cyber-physical systems, real-time and embedded systems, runtime assurance and verification, formal methods and tools, trust management, and high-confidence medical systems. He received a PhD in Computer Science from the University of Wisconsin, Madison. He is IEEE Fellow and received IEEE TC-RTS Outstanding Technical Achievement and Leadership Award in 2008.

Please cite this article in press as: Lee, J., et al., Zero-laxity based real-time multiprocessor scheduling. J. Syst. Software (2011), doi:10.1016/j.jss.2011.07.002