# FPZL Schedulability Analysis

Robert I. Davis and Alan Burns
Real-Time Systems Research Group, Department of Computer Science, University of York, YO10 5DD, York (UK)
*rob.davis@cs.york.ac.uk*; *alan.burns@cs.york.ac.uk*

*Abstract*— **This paper presents the Fixed Priority until Zero Laxity (FPZL) scheduling algorithm for multiprocessor real-time systems. FPZL is similar to global fixed priority pre-emptive scheduling; however, whenever a task reaches a state of zero laxity it is given the highest priority. FPZL is a minimally dynamic algorithm, in that the priority of a job can change at most once during its execution, bounding the number of pre-emptions. Polynomial time and pseudo-polynomial time sufficient schedulability tests are derived for FPZL. These tests are then improved by computing upper bounds on the amount of execution that each task can perform in the zero laxity state. An empirical evaluation shows that FPZL is highly effective, with a significantly larger number of tasksets deemed schedulable by the tests derived in this paper, than by state-of-the-art schedulability tests for Earliest Deadline until Zero Laxity (EDZL) scheduling.**

***Keywords-real-time; real-time; multiprocessor; scheduling; fixed priority; zero laxity; FPZL***

## I. INTRODUCTION

Approaches to multiprocessor real-time scheduling, can be categorised into two broad classes: partitioned and global. Partitioned approaches allocate each task to a single processor, dividing the multiprocessor scheduling problem into one of task allocation followed by uniprocessor scheduling. In contrast, global approaches allow tasks to migrate from one processor to another at run-time. Each approach has its distinct advantages and disadvantages [17]. Here, we focus on global scheduling techniques with the aim of increasing effectiveness, in terms of the number of tasksets that can be guaranteed schedulable, without compromising efficiency, in terms of the overheads caused by pre-emption and migration.

In this paper, we present a minimally dynamic global scheduling algorithm called FPZL (Fixed Priority until Zero Laxity). FPZL is based on global fixed priority pre-emptive scheduling, which for brevity we refer to as global FP scheduling. Under FPZL, jobs are scheduled according to the fixed priority of their associated task, until a situation is reached where the remaining execution time of a job is equal to the time to its deadline. Such a job has zero laxity and will miss its deadline unless it executes continually until completion. FPZL gives such zero-laxity jobs the highest priority. The schedules produced by FPZL and global FP scheduling are identical until the latter fails to execute a task with zero laxity. Such a task will subsequently miss its deadline. Hence FPZL dominates global FP scheduling, in the sense that all priority ordered tasksets that are schedulable according to global FP scheduling are also schedulable according to FPZL. FPZL is closely related to EDZL [22], [12], [4], [11], [26], [27], and [14] which applies the same zero-laxity rule to global EDF scheduling.

### A. Related Work

During the last ten years, sophisticated schedulability tests have been developed for global FP, and global EDF scheduling of sporadic tasksets with constrained and arbitrary deadlines.

In 2003, Baker [2] developed a fundamental schedulability test strategy, based on considering the minimum amount of interference in a given interval that is necessary to cause a deadline to be missed, and then taking the contra-positive of this to form a sufficient schedulability test. This basic strategy underpins an extensive thread of subsequent research.

Baker's work was built upon by Bertogna et al. [7] in 2005, (see also Bertogna et al. [9]). They developed sufficient schedulability tests for global EDF, and global FP scheduling based on bounding the maximum workload in a given interval. In 2007, Bertogna and Cirinei [8] adapted this approach to iteratively compute an upper bound on the response time of each task, using the upper bound response times of other tasks to limit the amount of interference considered. In 2009, Guan et al. [20] extended the response time analysis given in [8] using ideas from [6].

In 2009 and 2010, Davis and Burns [15], [16] showed that priority assignment is fundamental to the effectiveness of global FP scheduling. They proved that Audsley's optimal priority assignment algorithm [1] is applicable to some of the sufficient tests developed for global FP scheduling, including the deadline-based test of Bertogna et al. [9], but not to others such as the response time tests of Bertogna and Cirinei [8], and Guan et al. [20].

The Earliest Deadline first until Zero Laxity (EDZL) algorithm was introduced in 1994 by Lee [22], who showed that EDZL dominates global EDF scheduling, and is sub-optimal for two processors (see also Cho et al. [14], Park et al. [27]). Here, sub-optimal is used to mean that EDZL can "schedule any feasible set of *ready* tasks". This weak form of optimality is appropriate for online scheduling algorithms, which cannot take account of future arrival times. In 2006, Piao et al. [26] showed that EDZL is completion time

predictable in the sense defined by Ha and Liu [21], (see Section II.A). A simpler proof of predictability was given by Cirinei and Baker [12] in 2007, who also developed a sufficient schedulability test for EDZL based on the strategy of Baker [2].

In 2008, Baker et al. [4] gave an iterative sufficient test for EDZL based on the approach taken by Bertogna et al. [9] for work conserving algorithms and global EDF. The empirical evaluation in [4] shows that this iterative test for EDZL outperforms other tests for EDZL given in [12] and, as expected, similar tests for global EDF.

Also in 2008, Kato and Yamasaki [23], introduced EDCL, a variant of EDZL, which increases job priority on the basis of critical-laxity at the release or completion time of a job. This has the effect of reducing the maximum number of context switches to two per job, the same as EDF, at the expense of slightly inferior schedulability, when compared to EDZL. Kato and Yamasaki [23] also corrected a minor flaw in the polynomial time schedulability test for EDZL given in [12].

In 2009, Kato et al. [28], [24] presented research on RMZL (RMZL and FPZL are different names for essentially the same scheduling algorithm). Papers [28] and [24] were initially published in Japanese, with an English language version of [28] subsequently made available in May 2010 as a technical report [25]. The FPZL schedulability analysis presented in this paper was developed independently and initially published as a technical report in April 2010 [18]. We now make a brief comparison between the RMZL analysis given in [25] and the FPZL analysis presented in this paper.

The FPZL analysis, presented in this paper, is applicable to constrained-deadline tasksets with no restrictions on the priority ordering which may be used; whereas the RMZL analysis given in [25] is limited to implicit-deadline tasksets with task priorities assigned in Rate Monotonic priority order. The FPZL analysis computes which tasks can enter the zero-laxity state and only includes interference from those tasks; by comparison, the RMZL analysis includes zero-laxity interference from every lower priority task. This results in significant pessimism which heavily degrades performance as taskset cardinality increases much beyond the number of processors. The FPZL analysis computes an upper bound on the amount of execution that can take place in the zero-laxity state, while the RMZL analysis assumes that all of a zero-laxity task's execution can take place in the zero-laxity state.

In summary, the FPZL RTA-LC schedulability test presented in this paper is more widely applicable, dominates, and significantly outperforms the RMZL schedulability test given in [25].

### B. Intuition and Motivation

The research described in this paper is motivated by the need to close the large gap that currently exists between the best known approaches to global multiprocessor real-time scheduling for sporadic tasksets with constrained deadlines and what may be possible as indicated by feasibility / infeasibility tests.

Dynamic priority scheduling has the potential to schedule many more tasksets than fixed task or fixed job priority algorithms. However, this theoretical advantage must be balanced against the increased overheads that dynamic changes in priority can bring via an increase in the number of pre-emptions / migrations.

For example, the LLREF scheduling algorithm [13], which is optimal for periodic tasksets with implicit deadlines, and the LRE-TL scheduling algorithm [19] which is optimal for sporadic tasksets with implicit deadlines, divide the timeline into intervals that start and end at task releases/deadlines (referred to as TL-planes in [13]). In each interval, LLREF and LRE-TL ensure that each active task $\tau_i$ executes for at least $U_i t$, where $U_i$ is the task's utilisation, and $t$ is the length of the time interval. Hence every task can in the worst-case execute in every interval between task deadlines, resulting in $n$-1 pre-emptions per job release, where $n$ is the number of tasks. In systems with a large number of tasks, this level of pre-emptions leads to prohibitively high overheads.

Minimally dynamic scheduling algorithms, such as FPZL (and EDZL) offer a potential solution to this problem. Note, by *minimally dynamic*, we mean that the priority of a job changes at most once during its execution, hence bounding the number of pre-emptions / migrations to at most two per job release. By comparison, global FP and global EDF scheduling incur at most one pre-emption / migration per job release.

### C. Organisation

The remainder of the paper is organised as follows: Section II describes the terminology, notation and system model used. Section III describes sufficient tests for global FP scheduling. These tests are used in Section IV to derive polynomial time and pseudo-polynomial time sufficient schedulability tests for FPZL. Section V shows how the schedulability tests for FPZL can be improved by bounding the amount of execution that each task can perform in the zero-laxity state. Section VI presents an empirical investigation into the effectiveness of FPZL and its schedulability tests. Finally, Section VII concludes with a summary and suggestions for future research.

## II. System Model, Terminology and Notation

In this paper, we are interested in FPZL scheduling of an application on a homogeneous multiprocessor system comprising $m$ identical processors. The application or taskset is assumed to comprise a static set of $n$ tasks ( $\tau_1 ... \tau_n$ ), where each task $\tau_i$ is assigned a unique priority $i$, from 1 to $n$ (where $n$ is the lowest priority).

Tasks are assumed to comply with the *sporadic* task model. In this model, tasks give rise to a potentially infinite sequence of jobs. Each job of a task may arrive at any time

once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task.

Each task $\tau_i$ is characterised by its relative *deadline* $D_i$, *worst-case execution time* $C_i$, and minimum inter-arrival time or *period* $T_i$. The *utilisation* $U_i$ of each task is given by $C_i / T_i$. A task's *worst-case response time* $R_i$ is defined as the longest time from a job of the task arriving to it completing execution.

It is assumed unless otherwise stated that all tasks have constrained deadlines ( $D_i \leq T_i$ ). The tasks are assumed to be independent and so cannot be blocked from executing by another task other than due to contention for the processors. Further, it is assumed that once a task starts to execute it will not voluntarily suspend itself.

Intra-task parallelism is not permitted; hence, at any given time, each job may execute on at most one processor. As a result of pre-emption and subsequent resumption, a job may migrate from one processor to another. The cost of pre-emption, migration, and the run-time operation of the scheduler is assumed to be either negligible, or subsumed into the worst-case execution time of each task. Notwithstanding this assumption, we are interested in scheduling schemes that will have low overheads in practice (i.e. a minimum number of pre-emptions / migrations).

Under global FP scheduling, at any given time, the *m* highest priority ready jobs are executed. Under FPZL scheduling, if a job reaches zero laxity then it is given the highest priority and will execute until completion. The laxity of a job is given by the elapsed time to its deadline less its remaining execution time.

Schedulability analysis may identify certain tasks as being able to enter the *zero-laxity state*. We refer to these tasks as *zero-laxity tasks*. An upper bound on the maximum amount of execution that a job of task $\tau_i$ can perform in the zero-laxity state is denoted by $Z_i^{UB}$. Under FPZL, at any given time, at most *m* tasks may be in the zero-laxity state without a deadline being missed.

Finally, when discussing the schedulability of a given task $\tau_k$, we use the term *interference* to refer to the execution of other tasks, at a priority higher than *k*, that can potentially delay the completion of task $\tau_k$.

### A. Predictability

In 1994, Ha and Liu [21] defined the concept of scheduling algorithm *predictability*. A scheduling algorithm is referred to as *predictable* if the response times of jobs cannot be increased by decreases in their execution times, with all other parameters remaining constant. Predictability is a fundamental requirement for scheduling algorithms as in real systems task execution times are almost always variable up to some worst-case time.

Ha and Liu [21] proved that all priority driven (i.e. fixed job priority and fixed task priority) pre-emptive scheduling algorithms for multiprocessor systems are predictable. However, FPZL is a dynamic priority algorithm, and as such it is necessary to prove the predictability of FPZL before it can be considered useful. Proof that the FPZL scheduling algorithm is predictable with respect to decreases in task execution times follows from the logic used in Theorem 1 of [4] to prove that EDZL is predictable. Noting that under FPZL, the choice to schedule different jobs from the two sets of jobs considered in Theorem 1 of [4] can only be due to the zero-laxity rule. This is because the jobs belong to the same task and therefore have the same priority. They differ only in their execution times.

## III. SCHEDULABILITY TESTS FOR GLOBAL FP

In this section, we recapitulate two sufficient schedulability tests for global FP scheduling of sporadic tasksets. These tests are described in more detail in [16].

### A. Deadline Analysis for global FP

In [9], Bertogna et al. showed that if task $\tau_k$ is schedulable under global FP scheduling in an interval of length *L*, then an upper bound on the interference over the interval due to a higher priority task $\tau_i$ with or without a carry-in job is given by the following equation[1]. (Note a *carry-in job* is a job that is released prior to the start of the interval, and causes interference within that interval).

$$I_i^D(L, C_k) = \min(W_i^D(L), L - C_k + 1) \quad (1)$$

where $W_i^D(L)$ is an upper bound on the workload of task $\tau_i$ in an interval of length *L*, given by:

$$W_i^D(L) = N_i^D(L)C_i + \min(C_i, L + D_i - C_i - N_i^D(L)T_i) \quad (2)$$

and $N_i^D(L)$ is the maximum number of jobs of task $\tau_i$ that contribute all of their execution time in the interval:

$$N_i^D(L) = \lfloor (L + D_i - C_i) / T_i \rfloor \quad (3)$$

In [20], Guan et al. showed that if task $\tau_k$ is schedulable in an interval of length *L*, then an upper bound on the interference over the interval due to a higher priority task $\tau_i$ *without* a carry-in job is given by:

$$I_i^{NC}(L, C_k) = \min(W_i^{NC}(L), L - C_k + 1) \quad (4)$$

where:

$$W_i^{NC}(L) = N_i^{NC}(L)C_i + \min(C_i, L - N_i^{NC}(L)T_i) \quad (5)$$

and

$$N_i^{NC}(L) = \lfloor L / T_i \rfloor \quad (6)$$

The difference between the two interference terms given by (1) and (4) is:

$$I_i^{DIFF-D}(L, C_k) = I_i^D(L, C_k) - I_i^{NC}(L, C_k) \quad (7)$$

Building on the work of Guan et al. [20], Davis and Burns [16] showed that the worst-case scenario for a task $\tau_k$ under global FP scheduling occurs when it is released at the start of an interval where there are at most *m*-1 carry-in jobs of higher priority tasks (Theorem 1 of [16]). This observation,

[1] Note we adopt the approach to time representation used in [9]. Time is represented by non-negative integer values, with each time value *t* viewed as representing the whole of the interval [*t*, *t*+1). This enables mathematical induction on clock ticks and avoids confusion with respect to end points of execution.

lead to the introduction in [16] of an improved version of the polynomial-time, deadline-based test of Bertogna [9]:

DA-LC test for global FP scheduling [16]: *A sporadic taskset is schedulable, if for every task $\tau_k$ in the taskset, the inequality given by (8) holds:*

$$D_k \geq C_k + \left\lfloor \frac{1}{m}\left( \sum_{\forall i \in hp(k)} I_i^{NC}(D_k, C_k) + \sum_{i \in MD(k,m-1)} I_i^{DIFF-D}(D_k, C_k) \right) \right\rfloor \quad (8)$$

where $MD(k, m\text{-}1)$ is the subset of the $min(k, m\text{-}1)$ tasks with the largest values of $I_i^{DIFF-D}(D_k, C_k)$ from the set of tasks $hp(k)$ with priorities higher than $k$.

### B. Response Time Analysis for global FP

In [8], Bertogna and Cirinei showed that if task $\tau_k$ is schedulable under global FP scheduling in an interval of length $L$, then an upper bound on the interference in that interval due to a higher priority task $\tau_i$ with or without a carry-in job is given by:

$$I_i^R(L, C_k) = \min(W_i^R(L), L - C_k + 1) \quad (9)$$

where, $W_i^R(L)$ is an upper bound on the workload of task $\tau_i$ in an interval of length $L$, taking into account the upper bound response time ( $R_i^{UB}$ ) of task $\tau_i$ :

$$W_i^R(L) = N_i^R(L)C_i + \min(C_i, L + R_i^{UB} - C_i - N_i^R(L)T_i) \quad (10)$$

and $N_i^R(L)$ is given by:

$$N_i^R(L) = \left\lfloor (L + R_i^{UB} - C_i)/T_i \right\rfloor \quad (11)$$

In [20], Guan et al. showed that if task $\tau_i$ does not have a carry-in job, then the interference term is given by (4). The difference between the two interference terms, (9) and (4), is then given by:

$$I_i^{DIFF-R}(L, C_k) = I_i^R(L, C_k) - I_i^{NC}(L, C_k) \quad (12)$$

Further, Guan et al. [20] showed that for their analysis, the worst-case occurs when at most $m$-1 higher priority tasks with carry-in jobs contribute interference, thus deriving an improved version of the response time test of Bertogna and Cirinei [8]:

RTA-LC test for global FP scheduling [20]: *A sporadic taskset is schedulable, if for every task $\tau_k$ in the taskset, the upper bound response time $R_k^{UB}$ computed via the fixed point iteration given in (13) is less than or equal to the task's deadline:*

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m}\left( \sum_{\forall i \in hp(k)} I_i^{NC}(R_k^{UB}, C_k) + \sum_{i \in MR(k,m-1)} I_i^{DIFF-R}(R_k^{UB}, C_k) \right) \right\rfloor \quad (13)$$

where $MR(k, m\text{-}1)$ is the subset of the $min(k, m\text{-}1)$ tasks with the largest values of $I_i^{DIFF-R}(R_k^{UB}, C_k)$, given by (12), from the set of tasks $hp(k)$. Iteration starts with $R_k^{UB} = C_k$, and continues until the value of $R_k^{UB}$ converges or until $R_k^{UB} > D_k$, in which case task $\tau_k$ is unschedulable.

We note that using the RTA-LC test, task schedulability needs to be determined in priority order, highest priority first, as upper bounds on the response times of higher priority tasks are required for computation of the interference term $I_i^R(R_k^{UB})$. The RTA-LC test is pseudo-polynomial in complexity, and dominates the DA-LC test.

## IV. Schedulability tests for FPZL

In this section, we derive polynomial time and pseudo-polynomial time sufficient schedulability tests for FPZL. These tests are applicable to sporadic tasksets with constrained deadlines, and are independent of the priority assignment policy used. They are based on the tests described in the previous section for global FP scheduling.

### A. Deadline Analysis for FPZL

Schedulability under FPZL differs from that under global FP scheduling in two important aspects:

1. Under FPZL, up to $m$ tasks may be deemed unschedulable according to analysis of their response times; and yet, due to the zero-laxity rule, the tasks will not miss their deadlines.
2. Tasks that enter the zero-laxity state have an additional impact on the schedulability of other tasks.

We now derive the maximum interference on a higher priority task $\tau_k$, in an interval of length $L$, that could possibly be caused by a lower priority task $\tau_j$ executing in the zero-laxity state.

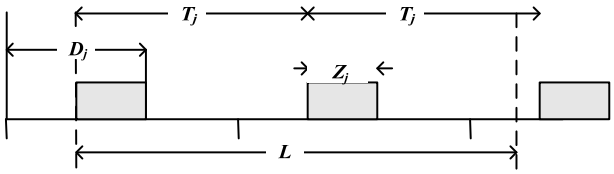


Figure 1: Zero-laxity interference in an interval

Figure 1 illustrates the worst-case scenario. This occurs when the first job of $\tau_j$ in the interval starts executing in the zero-laxity state, i.e. at the highest priority, at the start of the interval, and completes at its deadline. As zero-laxity execution can only occur immediately prior to a task's deadline, in the worst case, subsequent zero-laxity execution of the jobs of $\tau_j$ occurs at minimum intervals of $T_i$. Thus an upper bound on the amount of zero-laxity workload due to task $\tau_j$ in an interval of length $L$ is given by:

$$W_j^Z(L) = N_j^Z(L)Z_j^{UB} + \min(Z_j^{UB}, L - N_j^Z(L)T_j) \quad (14)$$

where $N_j^Z(L)$ is the maximum number of jobs of task $\tau_j$ that contribute all of their zero-laxity execution in the interval

$$N_j^Z(L) = \left\lfloor L/T_j \right\rfloor \quad (15)$$

Note, $Z_j^{UB}$ ( $\leq C_j$ ) is an upper bound on the amount of execution that any job of task $\tau_j$ can perform in the zero-laxity state.

If task $\tau_k$ is schedulable in an interval of length $L$, then an upper bound on the interference in that interval due to a lower priority task $\tau_j$ executing in the zero-laxity state is given by:

$$I_j^Z(L, C_k) = \min(W_j^Z(L), L - C_k + 1) \quad (16)$$

Davis and Burns [16] showed that the worst-case scenario for global FP scheduling of a task $\tau_k$ occurs when there are at most *m*-1 higher priority tasks with carry-in jobs. We observe that as execution of a lower priority task in the zero-laxity state can simply be modelled as the execution of a virtual higher priority task, then the same observation applies to our analysis of FPZL. We therefore need to consider the interference from each lower priority zero-laxity task with and without a carry-in job. The interference on task $\tau_k$ due to a lower priority zero-laxity task $\tau_j$, in an interval of length *L*, is the same irrespective of whether $\tau_j$ is considered as having a carry-in job or not. Hence task $\tau_j$ cannot contribute any additional carry-in interference (its $I_j^{DIFF-D}$ term is effectively zero) and so does not need to be included when determining the *m*-1 tasks that contribute the largest amounts of additional carry-in interference.

We now consider the interference from a higher priority task $\tau_i$ capable of entering the zero-laxity state. In this case, the maximum interference with a carry-in job occurs when the first job of $\tau_i$ in the interval starts executing at the start of the interval, and completes at its deadline, with all subsequent jobs executing as early as possible, see Figure 2 below. We observe that this is the same scenario that leads to the worst-case interference from a higher priority task which does not enter the zero-laxity state but completes at its deadline, and is given by (1). Similarly, zero-laxity execution cannot increase the amount of interference from a higher priority task with no carry-in job, given by (4). This is an important observation. It means that when calculating interference from higher priority tasks, we do not need to know if they are zero-laxity tasks.

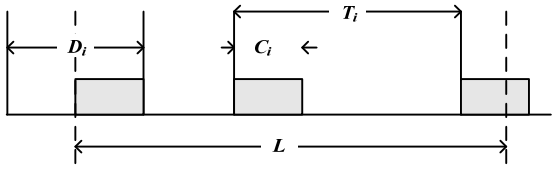


Figure 2: Interference in an interval

Under FPZL, each task $\tau_k$ is therefore schedulable without entering the zero-laxity state if the following inequality holds:

$$D_k \geq C_k + \left\lfloor \frac{1}{m} \left( \begin{array}{l} \sum_{\forall i \in hp(k)} I_i^{NC}(D_k, C_k) + \\ \sum_{i \in MD(k,m-1)} I_i^{DIFF-D}(D_k, C_k) + \\ \sum_{\forall j \in lpzl(k)} I_j^{Z}(D_k, C_k) \end{array} \right) \right\rfloor \quad (17)$$

where $I_i^{NC}(D_k, C_k)$ is given by (4), $I_i^{DIFF-D}(D_k, C_k)$ is given by (7), $I_j^{Z}(D_k, C_k)$ is given by (16), and *lpzl*(*k*) is the set of zero-laxity tasks with lower priorities than *k*.

If the inequality in (17) does not hold, then the task is a zero-laxity task. Under FPZL, at most *m* tasks can be zero-laxity tasks without a deadline being missed.

We note that the zero-laxity status of each task is unknown until its schedulability is checked via (17), hence task schedulability needs to be checked in priority order, lowest priority first.

Algorithm 1 presents the DA-LC schedulability test for FPZL. For now we make the pessimistic assumption that a zero-laxity task completes all of its execution in the zero-laxity state, hence 'Compute $Z_k^{UB}$' can be assumed to set $Z_k^{UB} = C_k$. Section V describes the calculation of less pessimistic upper bounds on zero-laxity execution.

```
1   countZL = 0
2   for (each priority level k, lowest first) {
3       Determine schedulability of τ_k using (17)
4       if ( τ_k is not schedulable) {
5           mark τ_k as a 'zero-laxity' task
6           countZL = countZL + 1
7           Compute Z_k^UB
8       }
9   }
10  if (countZL > m)
11      return unschedulable
12  else
13      return schedulable
```

Algorithm 1: DA-LC schedulability test for FPZL

The DA-LC schedulability test for FPZL is a polynomial time test requiring $O(n^2)$ operations (assuming that 'Compute $Z_k^{UB}$' takes linear time). Note that identifying the tasks with the *m*-1 largest values of $I_i^{DIFF-D}(D_k, C_k)$ can be achieved using a linear time selection algorithm [10].

We note that the DA-LC schedulability test for FPZL reduces to the DA-LC test for global FP scheduling for any taskset that the latter finds schedulable. Hence, the DA-LC test for FPZL dominates the DA-LC test for global FP scheduling.

We observe that a task $\tau_k$ may reach zero laxity, but still be schedulable at its fixed priority *k*, and so complete at its deadline. Such a task cannot have any impact on the execution of higher priority tasks, even though it is given the highest priority by the FPZL algorithm once it reaches zero laxity. This is because the maximum possible interference from higher priority tasks and lower priority zero-laxity tasks has already been accounted for and yet task $\tau_k$ was found to be schedulable, assuming execution at priority *k*. Hence there cannot be any time *t* between the release and the deadline of a job of $\tau_k$ at which it has zero laxity and there are *m* jobs with higher priorities are ready to execute, (otherwise $\tau_k$ would be unschedulable). Thus execution of task $\tau_k$ at the highest priority once it reaches zero laxity cannot interfere with the execution of any higher priority tasks. As there can be no interference on higher priority tasks, we do not classify a task that can reach zero-laxity but is nevertheless schedulable as a zero-laxity task, nor do we regard it as entering the zero-laxity state.

### B. Response Time Analysis for FPZL

Building on the work of Bertogna and Cirinei [8] and Guan et al. [20] (i.e. (13)), we now derive a sufficient schedulability test for FPZL which computes an upper bound $R_k^{UB}$ on the response time of each task $\tau_k$.

If task $\tau_k$ is schedulable under FPZL with a response time bounded by $R_k^{UB}$, then an upper bound on the interference in an interval of length $R_k^{UB}$ due to a lower priority task $\tau_j$ executing in the zero-laxity state can be obtained by substituting $R_k^{UB}$ for the length of the interval $L$ in (16).

We observe that again the maximum interference on task $\tau_k$ due to a lower priority zero-laxity task $\tau_j$ is the same irrespective of whether $\tau_j$ is considered as having a carry-in job or not. Hence task $\tau_j$ cannot contribute any additional carry-in interference and so does not need to be included when determining the $m$-1 tasks that contribute the largest amounts of additional carry-in interference.

In the previous section, we showed that using deadline analysis the maximum interference on task $\tau_k$ from a higher priority task $\tau_i$ capable of entering the zero-laxity state can be determined using (1) assuming a carry-in job, and using (4) assuming no carry-in job. Thus we showed that the maximum interference from a higher priority task is independent of whether or not that task is a zero-laxity task.

We now consider the situation when response time analysis ((9) and (4)) is used to compute the interference due a higher priority zero-laxity task. As the upper bound response time $R_i^{UB}$ of a zero-laxity task is equal to its deadline $D_i$, we find that (9) reduces to (1). Hence the maximum interference from a higher priority task is again independent of whether or not that task is a zero-laxity task. An upper bound on the worst-case response time of a task $\tau_k$ that is schedulable under FPZL without entering the zero-laxity state can therefore be found using the fixed point iteration given by (18).

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m} \left( \begin{array}{l} \sum_{\forall i \in hp(k)} I_i^{NC}(R_k^{UB}, C_k) + \\ \sum_{i \in MR(k,m-1)} I_i^{DIFF-R}(R_k^{UB}, C_k) + \\ \sum_{\forall j \in lpzl(k)} I_j^{Z}(R_k^{UB}, C_k) \end{array} \right) \right\rfloor \quad (18)$$

where $I_i^{NC}(R_k^{UB}, C_k)$ is given by (4), $I_i^{DIFF-R}(R_k^{UB}, C_k)$ is given by (12), and $I_j^{Z}(R_k^{UB}, C_k)$ is given by (16).

Iteration starts with $R_k^{UB} = C_k$, and continues until the value of $R_k^{UB}$ converges in which case $\tau_k$ is schedulable, or until $R_k^{UB} > D_k$. If $R_k^{UB} > D_k$, then the task is a zero-laxity task. Recall that under FPZL, at most $m$ tasks may be zero-laxity tasks without a deadline being missed.

Using (18), we can construct a sufficient schedulability test for FPZL based on upper bound response times; however, this is not entirely straightforward. Equation (18) depends on the response times of higher priority tasks (via (10)), and also on which lower priority tasks are zero-laxity tasks. Thus it would appear that we cannot compute task schedulability in increasing or decreasing priority order. This problem can however be solved by backtracking as shown in Algorithm 2.

Algorithm 2 initially assumes that there are no zero-laxity tasks and starts computing task response times in priority order, highest priority first (lines 6 and 7). Then, whenever a task $\tau_k$ is encountered where (18) results in a value of $R_k^{UB} > D_k$, the task is marked as a zero-laxity task and its upper bound response time is set to its deadline (lines 8 and 9). We note that provided that the taskset is schedulable under FPZL, then this is the correct upper bound response time, as the zero-laxity rule will prevent the task from actually missing its deadline.

```
1   countZL = 0
2   Initialize all R_k^UB = C_k and Z_k^UB = 0
3   repeat = true
4   while (repeat) {
5       repeat = false
6       for (each priority level k, highest first) {
7           Determine R_k^UB according to (18)
8           if ( R_k^UB > D_k ) {
9               R_k^UB = D_k
10              Compute Z_k^UB
11              if ( τ_k not marked as a ZL task) {
12                  mark τ_k as a ZL task
13                  repeat = true
14                  countZL = countZL + 1
15                  if(countZL > m) {
16                      repeat = false
17                      break (exit for loop)
18                  }
19              }
20          }
21          [if ( R_k^UB or Z_k^UB differ from prev. values)
22              repeat = true]
23      }
24  }
25  if (countZL > m)
26      return unschedulable
27  else
28      return schedulable
```

Algorithm 2: RTA-LC schedulability test for FPZL

The discovery of a zero-laxity task effectively invalidates the upper bound response times calculated for all higher priority tasks. These values could be too small, and therefore the process of upper bound response time calculation needs to be repeated (line 13). However, if more than $m$ zero-laxity tasks have been found, then even the zero-laxity rule cannot prevent deadline misses and the taskset is deemed unschedulable. In this case, the algorithm can exit immediately (lines 15-17).

We note that lines 21-22 are not required when a simple fixed value of $Z_k^{UB} = C_k$ is used for the zero-laxity execution time of task $\tau_k$. However, when the computed value of $Z_k^{UB}$ depends on the response times of higher priority tasks then this additional convergence check is required. This point is discussed in detail in Section V.

We note that the upper bound response time for a task $\tau_i$ is monotonically non-decreasing in the amount of zero-laxity execution time of each of the tasks with lower priority than *i*. Hence, the calculation of $R_i^{UB}$ can be made more efficient on subsequent iterations of the 'while' loop (line 4) by using as an initial value, the value of $R_i^{UB}$ computed on the previous iteration.

The 'while' loop (lines 4-24) continues to iterate until either *m*+1 zero-laxity tasks are found, in which case the taskset is unschedulable under FPZL, or there are *m* or fewer zero-laxity tasks and the upper bound response times have been re-calculated since the final zero-laxity task was found. In this case, the taskset is schedulable.

Under the assumption that 'Compute $Z_k^{UB}$' sets $Z_k^{UB} = C_k$, the RTA-LC schedulability test for FPZL requires at most $O(mn)$ response time calculations (i.e. (18)), each of which is pseudo-polynomial in complexity. This can be seen by noting that when 'Compute $Z_k^{UB}$', sets $Z_k^{UB} = C_k$, lines 21-22 are not required, and so the 'while' loop (line 4) only repeats when 'repeat' is set to true on line 13. This can only happen at most *m* times, as a result of finding a zero-laxity task, before the taskset is declared unschedulable. Hence the maximum number of times that a response time can be computed (line 7) is $O(mn)$. By comparison, the RTA-LC test for global FP scheduling requires $O(n)$ such response time calculations.

We note that the RTA-LC schedulability test for FPZL reduces to the RTA-LC test for global FP scheduling for any taskset that the latter finds schedulable. Hence, the RTA-LC test for FPZL dominates the RTA-LC test for global FP scheduling. Further, the RTA-LC test for FPZL also dominates the DA-LC test for FPZL.

## V. BOUNDING ZERO LAXITY EXECUTION TIME

So far, we have made the potentially pessimistic assumption that a task that can reach the zero-laxity state does so without having started to execute. Hence, we used an upper bound on the zero-laxity execution time of $Z_k^{UB} = C_k$. In this section, we derive a more effective upper bound and use this bound to improve the schedulability tests derived in Section IV.

Calculation of an improved bound relies on the concept of DC-Sustainability. A schedulability test for task $\tau_k$ is referred to as *DC-Sustainable* if it is sustainable [5] with respect to simultaneous and equal changes in both the execution time and the deadline of the task. Below we give a formal definition of DC-Sustainability.

*Definition*: A schedulability test *S* for a task $\tau_k$ is *DC-Sustainable* if the following two conditions hold:

*Condition 1*: If task $\tau_k$ is deemed schedulable by test *S* with some paired deadline and execution time values $D'_k = D_k - v$, $C'_k = C_k - v$ where $0 \le v \le C_k$ then test *S* is guaranteed to deem task $\tau_k$ schedulable for all deadline and execution time pairs $D'_k = D_k - w$, $C'_k = C_k - w$ where $v \le w \le C_k$.

*Condition 2*: If task $\tau_k$ is deemed unschedulable by test *S* with some paired deadline and execution time values $D'_k = D_k - v$, $C'_k = C_k - v$ where $0 \le v \le C_k$ then test *S* is guaranteed to deem task $\tau_k$ unschedulable for all deadline and execution time pairs $D'_k = D_k - w$, $C'_k = C_k - w$ where $0 \le w \le v$.

*Theorem 1*: Equation (17) is a DC-Sustainable schedulability test for task $\tau_k$.

*Proof*: We can re-write (17) as follows:

$$D'_k - C'_k \ge \left\lfloor \frac{1}{m} \left( \begin{array}{l} \sum_{\forall i \in hp(k)} I_i^{NC}(D'_k, C'_k) + \\ \sum_{i \in MD(k,m-1)} I_i^{DIFF-D}(D'_k, C'_k) + \\ \sum_{\forall j \in lpzl(k)} I_j^{Z}(D'_k, C'_k) \end{array} \right) \right\rfloor \quad (19)$$

Consider the behavior of (19) for paired deadline and execution time values $D'_k = D_k - w$, $C'_k = C_k - w$ as *w* takes different values in the range $0 \le w \le C_k$. The RHS of (19) gives an upper bound on the interference from higher priority tasks and lower priority tasks executing in the zero-laxity state in an interval of length $D'_k = D_k - w$. By inspecting (1), (2), (3), (4), (5), (6), (7), (14), (15), and (16) it can be seen that this interference is monotonically non-decreasing with respect to the length of the interval $D'_k$. We must however also consider the dependence of equations (1), (4) and (16) on $C'_k$, which also varies with *w*. $C'_k$ appears in the second term in the *min*( ) function of each of these equations in the expression $D'_k - C'_k + 1$. This expression is unchanged by varying *w*. The RHS of (19) is therefore monotonically non-increasing with respect to increasing values of *w*.

In the case of Condition 1, as the LHS of (19) is unchanged and the RHS is monotonically non-increasing for increasing values of *w*: $0 \le w \le C_k$ then it follows that, given that (19) holds for *w*=*v*, it must also hold for all values of *w*: $v \le w \le C_k$.

In the case of Condition 2 as the LHS of (19) is unchanged and the RHS is monotonically non-decreasing for decreasing values of *w*: $0 \le w \le C_k$ then it follows that, given that (19) does not hold for *w*=*v*, then it cannot hold for any value of *w*: $0 \le w \le v$ □

We now prove that (18) is also a DC-Sustainable schedulability test for task $\tau_k$. Below, we re-write (18), using the variable *q* to indicate the fixed point iteration.

$$R_k^{q+1} \leftarrow C'_k + \left\lfloor \frac{1}{m} \left( \begin{array}{l} \sum_{\forall i \in hp(k)} I_i^{NC}(R_k^q, C'_k) + \\ \sum_{i \in MR(k,m-1)} I_i^{DIFF-R}(R_k^q, C'_k) + \\ \sum_{\forall j \in lpzl(k)} I_j^{Z}(R_k^q, C'_k) \end{array} \right) \right\rfloor \quad (20)$$

Recall that iteration begins with $R_k^0 = C'_k$ (the execution time of task $\tau_k$), and ends when either $R_k^{q+1} = R_k^q$ or when $R_k^{q+1} > D'_k$, in which case task $\tau_k$ is unschedulable.

Let $R_k^{UB}(D,C)$ be the response time upper bound given by (20) for task $\tau_k$ ($D$, $C$) with deadline $D$ and execution time $C$. Similarly, let $R_k^{UB}(D+x,C+x)$ be the response time upper bound given by (20) for task $\tau_k$ ( $D+x$ , $C+x$ ) with deadline $D+x$ and execution time $C+x$ .

*Lemma 1*: If $\tau_k$ ($D$, $C$) is schedulable according to (20) then $R_k^{UB}(C+x) \geq R_k^{UB}(C)+x$ . Further, if $\tau_k$ ($D$, $C$) is not schedulable according to (20) then neither is $\tau_k$ ( $D+x$ , $C+x$ ).

*Proof*: Let $R_k^q(C)$ be the value computed by the $q$th iteration of (20) for task $\tau_k$ ($D$, $C$). Similarly, let $R_k^q(C+x)$ be the value computed by the $q$th iteration of (20) for task $\tau_k$ ( $D+x$ , $C+x$ ).

We prove the Lemma by induction, showing that on each iteration $q$, until either convergence or the deadline of $\tau_k$ ($D$, $C$) is exceeded, then $R_k^q(C+x) \geq R_k^q(C)+x$ .

Initial condition: in each case iteration starts with an initial value corresponding to the execution time of $\tau_k$ , hence $R_k^0(C) = C$ and $R_k^0(C+x) = C+x$ , so $R_k^0(C+x) \geq R_k^0(C)+x$ .

Inductive step: assume that $R_k^q(C+x) \geq R_k^q(C)+x$ , and consider the values computed for $R_k^{q+1}(C+x)$ and $R_k^{q+1}(C)$ on iteration $q$+1. The floor function (second term on the RHS of (20)) contains three summation terms; together, these terms give an upper bound on the interference from higher priority tasks and lower priority tasks executing in the zero-laxity state in an interval of length $R_k^q$ . Inspection of (4), (5), (6), (9), (10), (11), (12), (14), (15), and (16) shows that this interference term is no smaller for input values $R_k^q(C+x) \geq R_k^q(C)+x$ , and $C'_k = C+x$ (the latter is used in (4), (9) and (16)) than it is for input values $R_k^q(C)$ and $C'_k = C$ , hence once the value of $C'_k$ is added (first term on the RHS of (20)), we have $R_k^{q+1}(C+x) \geq R_k^{q+1}(C)+x$ .

We note that if the fixed point iteration for $\tau_k$ ($D$, $C$) converges on $R_k^{UB}(D,C) = R_k^{q+1}(C)$ , then the smallest possible value of $R_k^{UB}(D+x,C+x)$ is $R_k^{q+1}(C)+x$ . Further, if $\tau_k$ ($D$, $C$) is unschedulable, then it follows that $R_k^{q+1}(C) > D$ which implies that $R_k^{q+1}(C+x) > D+x$ and therefore $\tau_k$ ( $D+x$ , $C+x$ ) must also be unschedulable □

*Theorem 4*: Equation (20), and hence (18) is a DC-Sustainable schedulability test for task $\tau_k$ .

*Proof*: We can choose an execution time of $C'_k = 0$ and a deadline of $D'_k = D_k - C_k$ for task $\tau_k$ . With these parameters, $\tau_k$ is deemed schedulable by Equation (20). We then consider all possible deadline and execution time pairs $D'_k = D_k - w$ , $C'_k = C_k - w$ for $w$ from 1 to $C_k$ (recall that execution times are represented by non-negative integers). Let $v$ be the largest value of $w$, if any, for which $\tau_k$ is unschedulable. Lemma 1 tells us that for all smaller values of $w$, $\tau_k$ will also be unschedulable. Proof that Conditions 1 and 2 in the definition of DC-Sustainability hold follows directly from the observation that task schedulability is therefore monotonically decreasing with respect to decreasing values of $w$ □

We now show how a bound on the zero laxity execution time of each zero-laxity task can be derived. Let us assume that we are using the DA-LC schedulability test (Algorithm 1) or the RTA-LC schedulability test (Algorithm 2) for FPZL, and that task $\tau_k$ has been identified as a zero-laxity task by (17) or (18). We know that task $\tau_k$ cannot be guaranteed to complete all of its execution within its deadline, without entering the zero-laxity state. However, if we can show that $\tau_k$ is guaranteed to complete $C'_k = C_k - v$ units of execution time by an effective deadline of $D'_k = D_k - v$ , when executing at priority $k$, then that proves that the task can execute for at most $v$ units of time in the zero-laxity state. (Note for the reasons described at the end of Section VI.A, a job of task $\tau_k$ may reach zero laxity at or before $D_k - v$ from its release, but cannot cause interference on higher priority tasks until after $D_k - v$, otherwise $C_k - v$ units of execution time at priority $k$ would not be schedulable by $D_k - v$ . We therefore need only consider task $\tau_k$ as entering the zero-laxity state at $D_k - v$ , with zero-laxity execution time $v$).

Due to the DC-Sustainability of the single task schedulability tests given by (17) and (18), each of these equations can be used as the basis of a binary search to determine the smallest value of $v$ ($0 \leq v \leq C_k$) such that task $\tau_k$ is guaranteed to complete $C'_k = C_k - v$ units of execution time by a deadline $D'_k = D_k - v$ , thus computing an upper bound $Z_k^{UB} = v$ on the amount of time that a job of task $\tau_k$ can spend executing in the zero-laxity state. The initial minimum value of $v$ for the search is $v = 0$ which is known to result in un-schedulability, as $\tau_k$ is a zero-laxity task, while the initial maximum value is $v = C_k$ which is deemed to result in schedulability, as it is equivalent to $\tau_k$ having zero execution time.

In the DA-LC test, a binary search based on (17) can be used to 'Compute $Z_k^{UB}$' (line 8 of Algorithm 1), for each zero-laxity task, improving the effectiveness of the test. Similarly, in the RTA-LC test, a binary search based on (18) can be used to 'Compute $Z_k^{UB}$' (line 10 of Algorithm 2) for each zero-laxity task. However, in this case, a further convergence check (lines 21-22) is required as the zero-laxity execution times computed by the binary searches are dependent on the response times of higher priority tasks, and vice-versa. We note that Algorithm 2 will either find more than $m$ zero-laxity tasks or converge on unchanging values for the response times and zero-laxity execution times. Such convergence is guaranteed because the response times of higher priority tasks are monotonically non-decreasing with respect to increases in the zero laxity execution time of lower priority tasks, and similarly, the zero laxity execution times of lower priority tasks computed by binary search are monotonically non-decreasing with respect to increases in the response times of higher priority tasks.

## VI. EMPIRICAL INVESTIGATION

In this section, we present the results of an empirical investigation, examining the effectiveness of the schedulability tests for FPZL. We also conducted scheduling simulations which form necessary but not sufficient schedulability tests, thus providing upper bounds on the potential performance of the various algorithms.

### A. Taskset parameter generation

The taskset parameters used in our experiments were randomly generated as follows:

- Task utilisations were generated using the UUnifast-Discard algorithm [15], giving an unbiased distribution.
- Task periods were generated according to a log-uniform distribution with a factor of 1000 difference between the minimum and maximum possible task period. This represents a spread of task periods from 1ms to 1 second, as found in most hard real-time applications.
- Task execution times were set based on the utilisation and period selected: $C_i = U_i T_i$ .
- To generate constrained-deadline tasksets, task deadlines were assigned according to a uniform random distribution, in the range $[C_i, T_i]$ . For implicit-deadline tasksets, deadlines were set equal to periods.

In each experiment, the taskset utilisation (x-axis value) was varied from 0.025 to 0.975 times the number of processors in steps of 0.025. For each utilisation value, 1000 valid tasksets were generated and the schedulability of those tasksets determined using the various schedulability tests for different scheduling algorithms. The graphs plot the percentage of tasksets generated that were deemed schedulable in each case. The lines on all of the graphs appear in the order given in the legend. (The graphs are best viewed online in colour).

### B. Scheduling simulation

We used a simulation of global FP, FPZL, global EDF and EDZL scheduling to provide an upper bound on the potential performance of each scheduling algorithm, and hence to evaluate the quality of the schedulability tests. (Further details of the simulation are given in [18]) The simulation deemed a taskset schedulable by a given algorithm if it did not find a deadline miss during the time interval simulated, or any unavoidable deadline miss for any job that had execution time remaining at the end of the interval. Thus the simulation provides a necessary but not sufficient schedulability test. Any taskset failing the simulation, with a deadline miss, is guaranteed to be unschedulable, while tasksets that pass the simulation may or may not be schedulable. We note that in the case of constrained-deadline sporadic tasksets, to the best of our knowledge, no tractable exact tests exist for any of the algorithms studied. Thus upper bounds on performance derived via simulation are one of the few ways in which the performance potential of each algorithm can be explored.

### C. Schedulability test effectiveness

We investigated the performance of the FPZL DA-LC schedulability test using Audsley's OPA algorithm [1] to assign priorities. (When no task was found to be schedulable at a given priority then a heuristic assignment was made, selecting the task with the smallest proportion of execution time in the zero-laxity state). We compared the performance of the FPZL DA-LC test to that of the equivalent tests for global FP scheduling, and to schedulability tests for global EDF [9] (the "EDF-RTA" test) and EDZL scheduling [4] (the "EDZL-I test"). Also shown on the graphs are results for the necessary infeasibility test of Baker and Cirinei [3] (labelled "LOAD*"). This line gives the total number of tasksets at each utilisation level that we cannot be certain are infeasible (i.e. unschedulable by any algorithm). Further, the narrow lines on the graphs indicate an upper bound on the performance of each algorithm found via simulation. In the case of global FP and FPZL scheduling, these upper bounds assume Deadline minus Computation time Monotonic Priority Ordering (DCMPO) [15],[16], which was found in the simulation studies to be significantly more effective than Deadline Monotonic Priority Ordering (DMPO). Note it is not possible to simulate optimal priority assignment as simulation of all possible priority orderings is completely intractable.

Figures 3, 4, and 5 show the results of experiments for systems with 2, 4, and 8 processors and 10, 20, and 40 constrained-deadline tasks respectively.

From Figure 5 for the 8 processor case, we can see that the EDF-RTA test for global EDF scheduling and the DA-LC test for global FP scheduling using DMPO have the lowest performance, with approximately 50% of the generated tasksets schedulable at a utilisation of 2.7 (=0.34$m$) and 2.8 (=0.35$m$) respectively. The EDZL-I test performs significantly better with 50% of the tasksets schedulable at a utilisation of approx. 3.4 (=0.43$m$). Using optimal priority assignment significantly improves the performance of global FP scheduling, with 50% of the tasksets schedulable at a utilisation of approximately 4.7 (=0.59$m$) according to the DA-LC test. Finally, the DA-LC test for FPZL, using Audsley's OPA algorithm and a binary search to bound zero laxity execution time (marked FPZL-LZ on the graph) has the highest performance, with 50% of tasksets deemed schedulable at a utilisation of approx. 4.9 (=0.61$m$); a modest improvement over global FP scheduling.

The simulation results in Figure 5 show that both global EDF and global FP scheduling with DMPO have relatively poor performance potential. This is because these algorithms typically favour executing tasks with short deadlines first. This has the effect of reducing the amount of available concurrency, in terms of the number of ready tasks, which makes the remaining tasks more difficult to schedule.

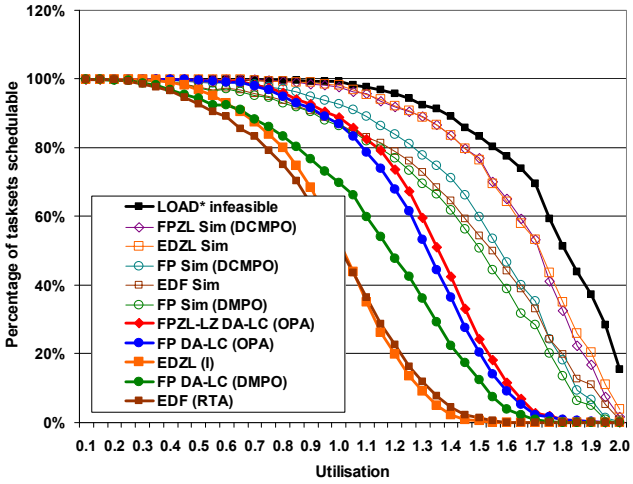


Figure 3: (2 processors, 10 tasks, D≤T)

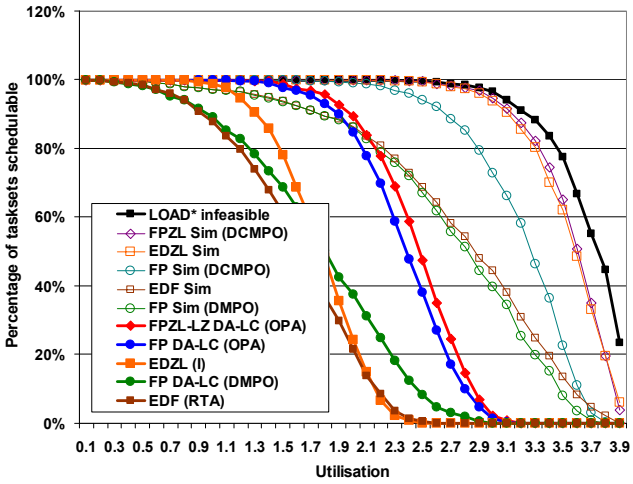


Figure 4: (4 processors, 20 tasks, D≤T)

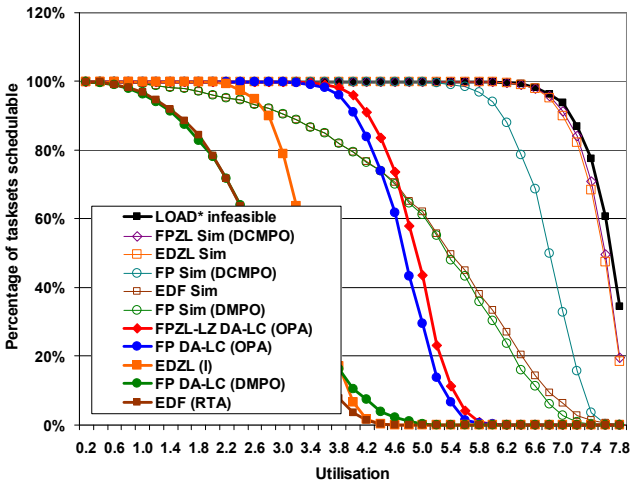


Figure 5: (8 processors, 40 tasks, D≤T)

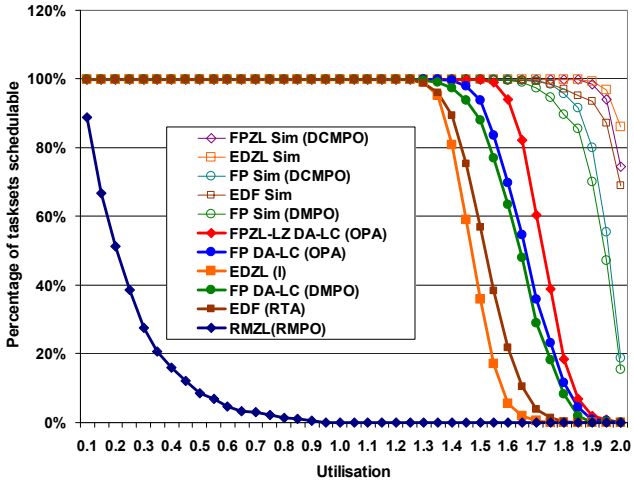


Figure 6: (2 processors, 10 tasks, D=T)

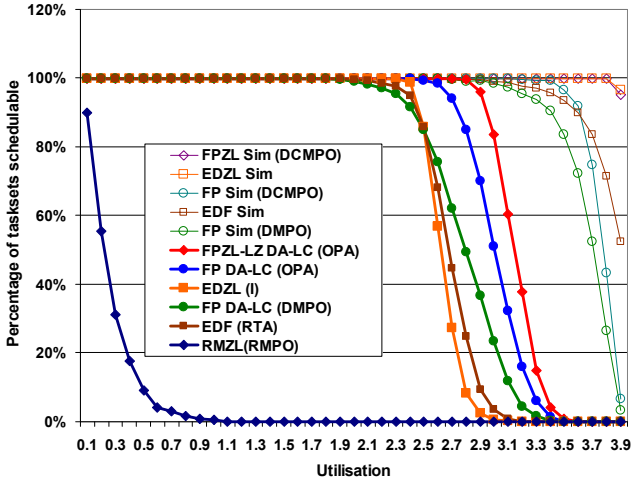


Figure 7: (4 processors, 20 tasks, D=T)

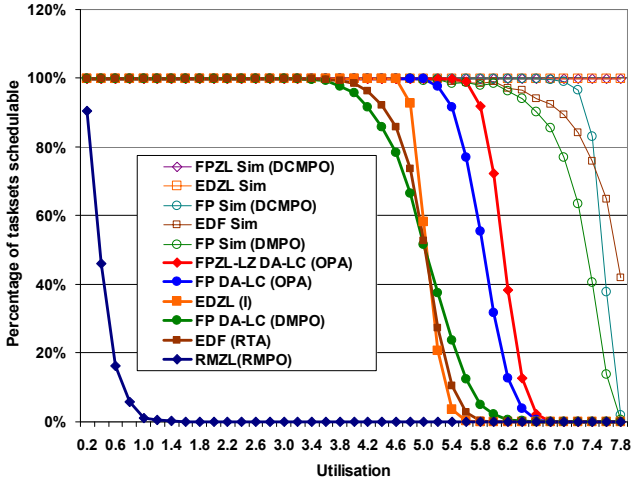


Figure 8: (8 processors, 40 tasks, D=T)

By contrast, using DCMPO greatly improves the performance potential of global FP scheduling, particularly when there are a large number of processors and tasks. The simulation results show that EDZL and FPZL with DCMPO priority ordering have similar performance potential, which as the number of processors and tasks increases becomes close to the bound given by the LOAD* infeasibility test.

Figures 6, 7, and 8 show the results of the same experiment, repeated for implicit-deadline tasksets. These graphs show that the performance of the schedulability tests for FPZL significantly exceed that of the tests for global FP, global EDF and EDZL, with an increased gap between FPZL and global FP scheduling using OPA, compared to the constrained deadline case. This increase in the relative performance of FPZL is due to the calculation of a less pessimistic bound on the amount of zero-laxity execution time having an increased effect in the implicit deadline case.

The performance of the RMZL schedulability test [25] is also shown in Figures 6, 7, and 8. The cause of significant pessimism in the RMZL test can be clearly seen in the results of a further experiment. This experiment compares the performance of the RMZL test with that of an equivalent FPZL RTA schedulability test[2] for implicit-deadline tasksets with varying cardinality ($n$ = 6, 8, 12, 20, and 36) on a 4 processor system. The results are shown in Figure 9.

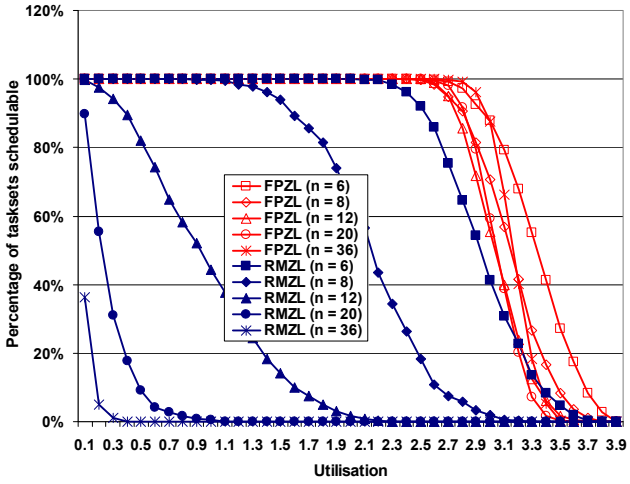


Figure 9: (4 processors, variable number of tasks, D=T)

From Figure 9, it is clear that the performance of the RMZL schedulability test deteriorates rapidly as the number of tasks is increased. This is due to the pessimism inherent in including an interference contribution from every lower priority task, rather than only from those that can enter the zero-laxity state. By comparison, the performance of the FPZL schedulability test is much less sensitive to the number of tasks, and shows similar behaviour to that of the response time test for global FP scheduling examined in [15] and [16].

[2] To obtain a simple and direct comparison, we used an FPZL schedulability test based on response time analysis, but not including the improvements derived by Guan et al. [20] and assuming that the entire execution of every zero-laxity task takes place in the zero-laxity state.

We also examined the relative improvements in the FPZL schedulability tests obtained by (i) computing a more effective bound on the zero-laxity execution time, (ii) using the pseudo-polynomial time RTA-LC tests rather than the polynomial time DA-LC tests. These appear in [18].

## VII. Conclusions and Future Work

The motivation for our work was the desire to improve upon current state-of-the-art global scheduling methods for hard real-time systems in terms of practical techniques that enable the efficient use of processing capacity.

The intuition behind our work was that dynamic priority scheduling has the potential to schedule many more tasksets than fixed task or fixed job priority algorithms, and yet this theoretical advantage has to be tempered by the need to avoid prohibitively large overheads due to a high number of pre-emptions. This led us to consider minimally dynamic scheduling algorithms which permit each job to change priority at most once during its execution. One such algorithm is EDZL. We applied the zero-laxity rule from EDZL to global FP scheduling, forming the FPZL scheduling algorithm. The number of context switches with FPZL is at most two per zero-laxity task, and one per ordinary task. As there are at most $m$ zero-laxity tasks, the increase in overheads compared to global FP scheduling is tightly bounded.

The key contributions of this paper are as follows:

- The derivation of effective polynomial time and pseudo-polynomial time sufficient schedulability tests for FPZL.
- Improvements to these tests, bounding the amount of execution that may take place in the zero-laxity state.

The main conclusions that can be drawn from our empirical investigations are as follows:

- The zero-laxity rule employed by FPZL appears to have a large impact on taskset schedulability, compared to the performance of global FP scheduling, as shown by the simulation results. The performance potential of FPZL using DCMPO was found to be broadly similar to that of EDZL, and significantly better than that of global FP or global EDF scheduling.
- Using Audsley's OPA algorithm to assign task priorities, the polynomial time schedulability test for FPZL results in a modest improvement over the equivalent test for global FP scheduling in the case of constrained-deadline tasksets, with an increased improvement for implicit-deadline tasksets.
- The schedulability tests for FPZL derived in this paper, and the schedulability tests for global FP scheduling, appear to significantly outperform tests for global EDF and EDZL. Even so, there remains a large gap between the sufficient schedulability tests for FPZL and what might be possible as shown by the simulation results.

Given the similarities between FPZL and EDZL, it is interesting to consider why the schedulability tests for FPZL significantly outperform those for EDZL. All of these schedulability tests are sufficient, and so suffer from a degree

of pessimism in terms of the computed interference. The advantage that the schedulability tests for FPZL have over those for EDZL is that this pessimism is restricted to tasks with higher priorities and lower priority zero-laxity tasks. With the schedulability tests for EDZL (and EDF), there is pessimism attributable to the calculation of interference from *all* other tasks. Further, the techniques derived in this paper, reduce the amount of interference considered due to tasks executing in the zero-laxity state, by bounding the amount of execution that takes place in that state. Nevertheless, the tests for FPZL have an additional element of pessimism compared to similar tests for global FP scheduling due to the inclusion of zero-laxity tasks in the interference term. This may account for the fact that the difference in performance between the schedulability tests for FPZL and global FP scheduling is not as large as the difference in the potential performance of the two algorithms as shown by simulation.

In future, we intend to investigate priority assignment policies for FPZL, including how task priorities should be assigned when it is inevitable that there will be some zero-laxity tasks. We also intend to look at variants of FPZL that reduce the number of scheduling points, based on the idea of critical-laxity and EDCL [23].

Semi-partitioned scheduling algorithms, where a small number of tasks are permitted to migrate from one processor to another, offer an alternative approach to achieving enhanced schedulability without excessive overheads, based on partitioning rather than global scheduling. In future, it would be interesting to compare the performance of FPZL with that of semi-partitioned scheduling algorithms.

## ACKNOWLEDGEMENTS

This work was funded by the EPSRC Tempo project (EP/G055548/1) and the EU funded ArtistDesign Network of Excellence. The authors would like to thank Shinpei Kato for providing a translation of [28], and for discussions about the RMZL schedulability test.

## REFERENCES

[1] N.C. Audsley, “On priority assignment in fixed priority scheduling”, Information Processing Letters, 79(1): 39-44, May 2001.

[2] T.P. Baker. “Multiprocessor EDF and deadline monotonic schedulability analysis”. In Proc. RTSS, pp. 120–129, 2003.

[3] T.P. Baker, M. Cirinei, “A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks”, In proc. Work-In-Progress (WIP) session of RTSS 2006.

[4] T.P.Baker, M. Cirinei, M. Bertogna, “EDZL scheduling analysis”. Real-Time Systems. 40:3, 264-289, 2008

[5] S.K. Baruah., A. Burns, “Sustainable Scheduling Analysis”. In Proc. RTSS, pp. 159-168, 2006.

[6] S.K. Baruah, “Techniques for Multiprocessor Global Schedulability Analysis”. In Proc. RTSS, pp. 119-128, 2007.

[7] M. Bertogna, M. Cirinei, G. Lipari, “New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors”. In Proc. 9th International Conf. on Principles of Distributed Systems, pp. 306-321, Dec. 2005.

[8] M. Bertogna, M. Cirinei, “Response Time Analysis for global scheduled symmetric multiprocessor platforms”. In Proc. RTSS, pp. 149-158, 2007.

[9] M. Bertogna, M. Cirinei, G. Lipari. “Schedulability analysis of global scheduling algorithms on multiprocessor platforms”. IEEE Transactions on parallel and distributed systems, 20(4): 553-566. April 2009.

[10] M. Blum, R.W. Floyd, V. Pratt, R. Rivest and R. Tarjan, "Time bounds for selection," Journal of Computer and System Sciences. 7 (1973) pp. 448-461.

[11] Y-H Chao, S-S Lin, K-J Lin, “Schedulability issues for EDZL scheduling on real-time multiprocessor systems”, Information Processing Letters, Volume 107, Issue 5, pp. 158-164, 16 August 2008

[12] M. Cirinei, T. P. Baker. “EDZL scheduling analysis”. In Proc. ECRTS, pp. 9–18, 2007.

[13] H. Cho, B. Ravindran, E.D. Jensen, “An Optimal Real-Time Scheduling Algorithm for Multiprocessors”. In Proc. RTSS pp. 1001-110, 2006.

[14] S. Cho, S-K. Lee, A. Han, K-J Lin, “Efficient real-time scheduling algorithms for multiprocessor systems”. IEICE Transactions on Communications Vol. E85-B No. 12, pp.2859–2867, 2002.

[15] R.I. Davis, A. Burns, “Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems”. In Proc. RTSS, pp. 398-409, 2009.

[16] R.I. Davis, A. Burns, “Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems”. Real-Time Systems (to appear). DOI 10.1007/s11241-010-9106-5. Available from http://www-users.cs.york.ac.uk/~robdavis/

[17] R.I. Davis, A. Burns, “A Survey of Hard Real-Time Scheduling for Multiprocessor Systems”, ACM Computing Surveys (to appear). Available from http://www-users.cs.york.ac.uk/~robdavis/

[18] R.I. Davis, A. Burns, “FPZL Schedulability Analysis”, Technical Report YCS-2010-452, Dept. of Computer Science, University of York, April 2010. Available from http://www-users.cs.york.ac.uk/~robdavis/

[19] S. Funk, V. Nadadur, “LRE-TL: An Optimal Multiprocessor Algorithm for Sporadic Task Sets”. In Proc. RTNS, pp. 159-168, 2009.

[20] N. Guan, M. Stigge, W.Yi, G. Yu, “New Response Time Bounds for Fixed Priority Multiprocessor Scheduling”. In Proc. RTSS, pp 387-397 2009.

[21] R. Ha, J. W.-S. Liu, “Validating timing constraints in multiprocessor and distributed real-time systems”. In Proc. of the International conference on Distributed Computing Systems, pp. 162–171, 1994.

[22] S.K. Lee, “On-line multiprocessor scheduling algorithms for real-time tasks”, In Proc. IEEE Region 10’s Ninth Annual International Conference, pp. 607–611, 1994.

[23] S. Kato, N. Yamasaki, “Global EDF-based scheduling with efficient priority promotion”. In Proc. of RTCSA pp. 197–206, 2008.

[24] S. Kato, N. Yamasaki. "Real-Time Scheduling Module for Linux Kernel", IPSJ Transactions on Advanced Computing Systems, Vol. 2, No. 1 (ACS25), pp. 75-86, 2009. (in Japanese).

[25] S. Kato, A. Takeda, N. Yamasaki. "Global Rate-Monotonic Scheduling with Priority Promotion", Technical Report CMU-ECE-TR10-05, May, 2010.

[26] Piao X, Han S, Kim H, Park M, Cho Y, Cho S “Predictability of earliest deadline zero laxity algorithm for multiprocessor real time systems”. In: Proc. of the international symposium on object and component-oriented real-time distributed computing, 2006.

[27] M. Park, S. Han, H. Kim, S. Cho, Y. Cho, “Comparison of deadline-based scheduling algorithms for periodic real-time tasks on multiprocessor”. IEICE Transactions on Information Systems Vol. E88-D No. 3, pp. 658–661, 2005.

[28] A. Takeda, S. Kato N. Yamasaki. "Real-Time Scheduling based on Rate Monotonic for Multiprocessors", IPSJ Transactions on Advanced Computing Systems, Vol. 2, No. 1 (ACS25), pp. 64-74, 2009. (in Japanese)