Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems

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 is an extended version of a paper that appeared in the proceedings of the IEEE Real-Time Systems Symposium 2009. This paper has been updated with respect to advances made in schedulability analysis, and contains a number of significant additional results.

The paper addresses the problem of priority assignment in multiprocessor real-time systems using global fixed task-priority pre-emptive scheduling.

We prove that Audsley's Optimal Priority Assignment (OPA) algorithm, originally devised for uniprocessor scheduling, is applicable to the multiprocessor case, provided that three conditions hold with respect to the schedulability tests used. Our empirical investigations show that the combination of optimal priority assignment policy and a simple compatible schedulability test is highly effective in terms of the number of tasksets deemed to be schedulable.

We also examine the performance of heuristic priority assignment policies such as Deadline Monotonic, and an extension of the TkC priority assignment policy called DkC that can be used with any schedulability test. Here we find that Deadline Monotonic priority assignment has relatively poor performance in the multiprocessor case, while DkC priority assignment is highly effective.

Keywords: Real-Time, multiprocessor, multicore, optimal priority assignment; heuristic priority assignment; global scheduling; fixed priority; taskset generation; schedulability analysis.

Extended version

This paper is an expanded version of "Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems" (Davis and Burns, 2009a), which appeared in the proceedings of the IEEE Real-Time Systems Symposium (RTSS) 2009. This paper updates and extends that work as follows:

- Research by Guan et al. (2009) also published in RTSS 2009 improved upon the response time analysis (RTA) test of Bertogna and Cirinei (2007) for global fixed priority scheduling used by Davis and Burns (2009a). Here, we integrate these improvements into our research on priority assignment.
- We provide formal proof (Theorem 1) of a key observation made by Guan et al. (2009) concerning the pattern of task execution that results in the worst-case response time of a task under global fixed priority pre-emptive scheduling. Our proof applies to the general case, and is not limited in scope to the sufficient analysis given by Guan et al. (2009).
- We introduce an improved version of the polynomial time schedulability test of Bertogna et al. (2005) by limiting the carry-in interference using the approach of Guan et al. (2009).
- We update our analysis of which schedulability tests are compatible with Audsley's Optimal Priority Assignment (OPA) algorithm (Audsley, 1991, 2001) with respect to these improved schedulability tests, and also the tests of Fisher and Baruah (2006), and Baruah and Fisher (2008).
- We derived a pseudo-schedulability condition that dominates the response time analysis of Guan et al. (2009). We show that this condition is compatible with the OPA algorithm.
- We update our empirical results to cover the improved schedulability tests, and also the pseudo-schedulability condition which provides an upper bound on the performance of the response time analysis of Guan et al. (2009) combined with any priority assignment policy. We also provide results for tasksets with implicit deadlines as well as those for tasksets with constrained deadlines. Further, we add a comparison with the response time schedulability test for global EDF given by Bertogna and Cirenei (2007).
- Finally, in Appendix A, we consider alternative heuristic priority assignment policies based on RM-US$\{\varsigma\}$ (Andersson et al., 2001) and SM-US$\{\varsigma\}$ (Andersson, 2008).

1. Introduction

Real-time embedded systems are found in many diverse application areas including; automotive electronics, avionics, space systems, telecommunications, and consumer electronics. In all of these areas, there is rapid technological progress. Companies building embedded real-time systems are driven by a profit motive. To succeed, they aim to meet the needs and desires of their customers by providing systems that are more capable, more flexible, and more effective than their competition, and by bringing these systems to market earlier. This desire for technological progress has resulted in a rapid increase in both software complexity and processing demands. To address these demands for increased processor performance, silicon vendors no longer concentrate on increasing processor clock speeds, as this approach has lead to problems with high power consumption and the need for excessive heat dissipation. Instead, there is now an increasing trend towards using multiprocessor platforms for high-end real-time applications. In Avionics (Rosenburg, 2009), Automotive Electronics (Leteinturier, 2007), and Space systems, this trend is driven by the capability of multicore devices to significantly reduce size, weight and power requirements.

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 (bin-packing) followed by uniprocessor scheduling. In contrast, global approaches allow tasks to migrate from one processor to another at run-time. Real-time scheduling algorithms can be categorised into three classes based on when priorities can change: *fixed task-priority* (all invocations, or jobs, of a task have the same priority), *fixed-job priority* and *dynamic-priority*.

In this paper, we focus on priority assignment policies for global fixed task-priority pre-emptive scheduling, which for brevity we refer to as *global FP scheduling*.

1.1. Related work

In the context of uniprocessor fixed priority scheduling, there are three fundamental results regarding priority assignment. Serlin (1972) and Liu and Layland (1973) showed that Rate Monotonic priority ordering (RMPO) is optimal for independent *synchronous* periodic tasks (that share a common release time) that have *implicit deadlines* (equal to their periods). Leung and Whitehead (1982) showed that Deadline Monotonic priority ordering (DMPO) is optimal for independent synchronous tasks with *constrained deadlines* (less than or equal to their periods). Audsley (1991, 2001) devised an optimal priority assignment (OPA) algorithm that solved the problem of priority assignment for *asynchronous* tasksets, and for tasks with *arbitrary deadlines* (which may be greater than their periods).

In the context of multiprocessor global FP scheduling, work on priority assignment has focussed on circumventing the so called “Dhall effect”. Dhall and Liu (1978) showed that under global FP scheduling with RMPO, a set of periodic tasks with implicit deadlines and total utilisation just greater than 1 can be unschedulable on *m* processors. For this problem to occur at least one task must have a high utilisation.

Andersson and Jonsson (2000b) designed the TkC priority assignment policy to circumvent the Dhall effect. TkC assigns priorities based on a task’s period (T_i) minus *k* times its worst-case execution time (C_i), where *k* is a real value computed on the basis of the number of processors. Via an empirical investigation, Andersson and Jonsson showed that TkC is an effective priority assignment policy for periodic tasksets with implicit deadlines.

Andersson et al. (2001) gave a utilisation bound for global FP scheduling of periodic tasksets with implicit deadlines using the RM-US{ς} priority assignment policy. RM-US{ς} gives the highest priority to tasks with utilisation greater than a threshold ς. Andersson and Jonsson (2003) showed that the maximum utilisation bound for global FP scheduling of such tasksets is $(\sqrt{2}-1)m \approx 0.41m$, when priorities are defined as a scale invariant function of worst-case execution times and periods.

Bertogna et al. (2005) extended the work of Andersson et al. (2001) to sporadic tasksets with constrained deadlines forming the DM-DS{ς} priority assignment policy. DM-DS{ς} gives the highest priority to at most $m-1$ tasks with densities greater than the threshold ς, and otherwise uses DMPO. Bertogna et al. (2005) provided a density-based schedulability test for DM-DS{ς}. Andersson (2008) proposed a form of Slack Monotonic priority assignment called SM-US{ς}. Using a threshold of $2/(3+\sqrt{5})$, SM-US{ς} has a utilisation bound of $2/(3+\sqrt{5})m \approx 0.382m$ for sporadic tasksets with implicit-deadlines.

More sophisticated schedulability tests for global FP scheduling of sporadic tasksets with constrained and arbitrary deadlines have been developed using analysis of response times and processor load.

Andersson and Jonsson (2000a) gave a simple response time upper bound applicable to tasksets with constrained-deadlines. Baker (2003) 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 into schedulability tests for global EDF (Baker and Baruah, 2009; Bertogna, 2007; Baruah and Baker, 2009; Baruah et al., 2009), and global FP scheduling (Bertogna et al., 2005, 2009; Baker, 2006; Fisher and Baruah , 2006; Baruah and Fisher, 2008; Guan et al., 2009).

Baker’s work was subsequently built upon by Bertogna et al. (2005), and Bertogna and Cirinei (2009). They developed sufficient schedulability tests for: (i) any work conserving algorithm, (ii) global EDF, and (iii) global FP scheduling based on bounding the maximum workload in a given interval. This basic approach was extended to form an iterative schedulability test using the computed slack for each task to limit the amount of carry-in interference and hence to calculate a new value for the slack. Bertogna and Cirinei (2007) 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.

Guan et al. (2009) extended the response time analysis of Bertogna and Cirinei (2007), limiting the amount of carry-in interference using ideas from Baruah (2007).

Global multiprocessor scheduling is intrinsically a much more difficult problem than uniprocessor scheduling due to the simple fact that a task can only use one processor at a time, even when several are free (Liu, 1969). This restriction manifests itself as the critical instant effect (Lauzac et al., 1998), where simultaneous release of all tasks does not necessarily lead to worst-case response times. As a result, to the best of our knowledge, no exact tests are currently known for global FP scheduling of sporadic tasksets. Exact tests are only known for the strictly periodic case (Cucu and Goossens, 2006, 2007).

For an extensive survey of multiprocessor scheduling, the interested reader is referred to (Davis and Burns, 2009b).

1.2. 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 multiprocessor real-time scheduling for sporadic tasksets with constrained deadlines and what may be possible as indicated by feasibility / infeasibility tests. We hypothesise that a key factor in closing this gap is priority assignment. The intuition behind our work is the idea that for fixed priority scheduling, finding an appropriate priority ordering is as important as using an effective schedulability test.

In the simulation chapter of his thesis, Bertogna (2007) showed that for sporadic tasksets with constrained deadlines, the response time test given in (Bertogna and Cirinei, 2007) for global FP scheduling – using DMPO, outperformed all other tests known at the time, including those for global FP, global EDF, and EDZL (Cirinei and Baker 2007, Baker et al., 2008); a minimally dynamic global scheduling algorithm that dominates global EDF. While DMPO is known to be an optimal priority assignment policy for the equivalent uniprocessor case (Leung and Whitehead, 1982), this optimality does not extend to multiprocessors.

In this paper, we prove that Audsley's Optimal Priority Assignment (OPA) algorithm (Audsely, 1991, 2001), originally devised for uniprocessor scheduling, is applicable to the multiprocessor case provided that the schedulability test used meets three simple conditions. These conditions allow us to classify schedulability tests for global FP scheduling into two categories: OPA-compatible and OPA-incompatible. We show via an empirical investigation that optimal priority assignment combined with a polynomial time OPA-compatible schedulability test can be significantly more effective in terms of the number of tasksets deemed schedulable, than using a state-of-the-art, pseudo-polynomial-time OPA-incompatible schedulability test with DMPO. Further, we build on the work of Andersson and Jonsson (2000b), developing heuristic priority assignment policies: D-CMPO and DkC that are applicable to any schedulability test. Our empirical studies show that DkC significantly outperforms DMPO, giving close to optimal results.

1.3. Organisation

The remainder of the paper is organised as follows: Section 2 describes the terminology, notation and system model used. Section 3 describes sufficient tests for global FP scheduling. Section 4 discusses both optimal and heuristic approaches to priority assignment. Section 5 outlines an unbiased method of taskset generation based on techniques developed for the uniprocessor case. Section 6 presents an empirical investigation into the effectiveness of priority assignment policies and sufficient schedulability tests. Finally, Section 7 concludes the paper.

2. System model, terminology and notation

In this paper, we are interested in global FP 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. Before the taskset can be scheduled, a priority assignment policy is used to assigned a unique static priority i, from 1 to n (where n is the lowest priority) to each task. For convenience of notation, each task τ_i is identified by its unique priority i.

We are interested in two task models, referred to as *periodic* and *sporadic*. In both models, tasks give rise to a potentially infinite sequence of jobs. In the periodic task model, the jobs of a task arrive strictly periodically, separated by a fixed time interval. In the sporadic task model, 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 τ_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 task arriving to it completing execution.

It is assumed unless otherwise stated that all tasks have constrained deadlines ($D_i \le T_i$). 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 subsumed into the worst-case execution time of each task.

2.1. Feasibility, schedulability and optimality

A taskset is referred to as *feasible* if there exists a scheduling algorithm that can schedule the taskset without any deadlines being missed. Further, we refer to a taskset as being *global FP feasible* if there exists a priority ordering under which the taskset is schedulable using global FP scheduling.

In systems using global FP scheduling, it is useful to separate the two concepts of priority assignment and schedulability testing. The priority assignment problem is one of determining the relative priority ordering of a set of tasks. Given a taskset with some priority ordering, then the schedulability testing problem involves determining if the taskset is schedulable with that priority ordering. Clearly the two concepts are closely related. For a given taskset, there may be many priority orderings that are unschedulable, and just a few that are schedulable.

A schedulability test *S* can be classified as follows. Test *S* is said to be *sufficient* if all of the priority ordered tasksets that it deems schedulable are in fact schedulable. Similarly, test *S* is said to be *necessary* if all of the priority ordered tasksets that it deems unschedulable are in fact unschedulable. Finally, test *S* is referred to as *exact* if it is both sufficient and necessary.

The concept of an *optimal priority assignment policy* can be defined with respect to a specific schedulability test *S*:

Definition 1: *Optimal priority assignment policy*: A priority assignment policy *P* is referred to as *optimal* with respect to a schedulability test *S* and a given task model, if and only if there are no tasksets that are compliant with the task model that are deemed schedulable by test *S* using another priority assignment policy, that are not also deemed schedulable by test *S* using policy *P*.

We note that the above definition is applicable to both sufficient (and not necessary) schedulability tests and exact schedulability tests.

An optimal priority assignment policy for an exact schedulability test facilitates classification of all global FP feasible tasksets compliant with a particular task model. For example, for periodic tasksets, Cucu and Goossens (2006, 2007) showed that exact schedulability can be determined by simulating the schedule over an interval related to the hyperperiod[1] of the taskset. For this exact test the only known optimal priority assignment policy involves checking all *n*! possible priority orderings (Cucu, 2008). Combining the two, it is theoretically possible, but computational intractable, to determine if any given periodic taskset is global FP feasible.

Using an optimal priority assignment policy for a sufficient (but not necessary) test *S* we cannot classify all global FP feasible tasksets, due to the fact that the test is not exact. However, optimal performance is still provided with respect to the limitations of the test itself. For example, the set *Y* of all tasksets that are deemed schedulable by a sufficient test *S* using its optimal priority assignment policy is a superset of the set *Z* of all tasksets that are deemed schedulable by test *S* using any other priority assignment policy. Further due to fact that the test is not exact, *Y* is a strict subset of the set *G* containing all global FP feasible tasksets ($G \supset Y \supseteq Z$).

The concept of *comparability* relates to the priority ordered tasksets that are deemed schedulable by different schedulability tests. There are three possibilities:

1. *Dominance*: Schedulability test *S* is said to *dominate* test *V*, if all of the priority ordered tasksets that are schedulable according to test *V* are also schedulable according to test *S*, and priority ordered tasksets exist that are schedulable according to test *S*, but not according to test *V*.
2. *Equivalence*: Schedulability tests *S* and *V* are equivalent if all of the priority ordered tasksets that are schedulable according to test *S* are also schedulable according to test *V*, and vice-versa.
3. *Incomparable*: Priority ordered tasksets exist that are schedulable according to test *S,* but not according to test *V* and vice-versa.

3. Schedulability tests

In this section, we outline two sufficient schedulability tests for global fixed priority scheduling of sporadic tasksets. The first was developed by Bertogna et al (2009) and uses deadline analysis. The second was developed by Bertogna and Cirinei (2007) and uses response time analysis. Subsequently, the response time test was improved by Guan et al. (2009), using ideas from (Baruah, 2007) which limit the amount of carry-in interference.

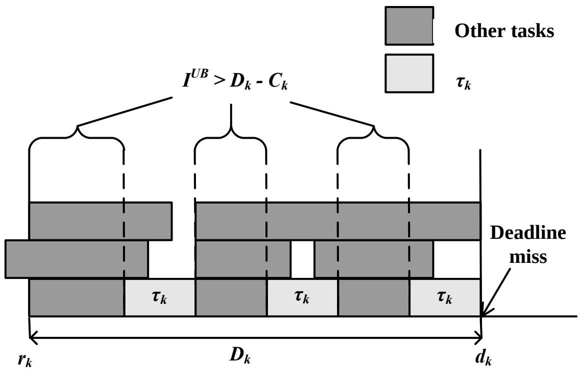

Figure 1: Problem window

All of these schedulability tests are based on the fundamental strategy derived by Baker (2003), the outline of which is as follows:

1. Consider an interval referred to as the *problem window*, at the end of which a deadline is missed, for example the interval of length D_k from the arrival to the deadline of some job of task τ_k, see Figure 1.

[1] The hyperperiod of a taskset is the least common multiple of the task periods.

2. Establish a condition *necessary* for the job to miss its deadline, for example, all m processors executing other tasks for more than $D_k - C_k$ during the interval.
3. Derive an upper bound I^{UB} on the maximum interference in the interval due to other tasks.
4. Form a necessary unschedulability test; i.e. an inequality between I^{UB} and the amount of execution necessary for a deadline miss, then negate this inequality to form a sufficient schedulability test.

Bertogna et al. (2009) derived a simple sufficient test using the above approach, by considering the maximum amount of interference that could occur in the problem window due to each higher priority task. This maximum interference occurs when the first job of the higher priority task in the problem window starts executing at the start of the problem window, and completes at its deadline, with all subsequent jobs executing as early as possible – see Figure 2.

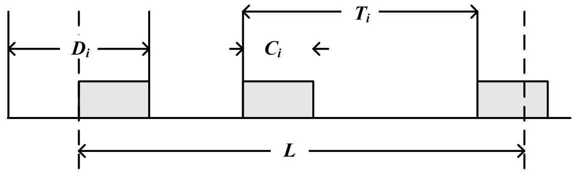

Figure 2: Interference in an interval

Bertogna et al. (2009) showed that $W_i^D(L)$ is an upper bound on the workload of task τ_i in an interval of length L:

$$W_i^D(L) = N_i^D(L)C_i + \min(C_i, L + D_i - C_i - N_i^D(L)T_i) \quad (1)$$

where $N_i^D(L)$ is the maximum number of jobs of task τ_i that contribute all of their execution time in the interval.

$$N_i^D(L) = \left\lfloor \frac{L + D_i - C_i}{T_i} \right\rfloor \quad (2)$$

If task τ_k is schedulable, then an upper bound on the interference due to a higher priority task τ_i in an interval of length D_k is given by[2]:

$$I_i^D(D_k, C_k) = \min(W_i^D(D_k), D_k - C_k + 1) \quad (3)$$

Note, the '+1' term in Equation (3) is a result of the approach to time representation[3] used in (Bertogna et al., 2009) and also in this paper.

A sufficient schedulability test for each task τ_k is then given by the following inequality:

$$D_k \geq C_k + \left\lfloor \frac{1}{m} \sum_{i \in hp(k)} I_i^D(D_k, C_k) \right\rfloor \quad (4)$$

where $hp(k)$ refers to the set of tasks with priorities higher than k. Note we have re-written Equation (4) in a different form from that presented by Bertogna et al. (2009) for ease of comparison with the schedulability test given by Bertogna and Cirinei (2007). We refer to Equation (4) as the "DA test".

Bertogna and Cirinei (2007) extended the method described above to iteratively compute an upper bound response time R_k^{UB} for each task, using the upper bound response times of higher priority tasks to limit the amount of interference considered. This extended approach applies the same logic as (Bertogna et al., 2009), while recognising that the latest time that a task can execute is when it completes with its worst-case response time rather than at its deadline.

Below, we give the schedulability test for this method. Note we have simplified the equations given by Bertogna and Cirinei (2007) to remove the slack terms and use upper bound response times directly. This is possible for global FP scheduling as the response times computed are unaffected by lower priority tasks[4].

Taking upper bound response times into account, an upper bound $W_i^R(L)$ on the workload of task τ_i in an interval of length L is given by:

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

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

$$N_i^R(L) = \left\lfloor \frac{L + R_i^{UB} - C_i}{T_i} \right\rfloor \quad (6)$$

If task τ_k is schedulable, then an upper bound on the interference due to a higher priority task τ_i in an interval of length R_k^{UB} is given by:

$$I_i^R(R_k^{UB}, C_k) = \min(W_i^R(R_k^{UB}), R_k^{UB} - C_k + 1) \quad (7)$$

An upper bound on the response time of each task τ_k can then be found via the following fixed point iteration (Theorem 7 in (Bertogna and Cirinei, 2007)).

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \in hp(k)} I_i^R(R_k^{UB}, C_k) \right\rfloor \quad (8)$$

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 τ_k is unschedulable. We refer to Equation (8) as the "RTA test".

Guan et al. (2009) extended the response time analysis of Bertogna and Cirinei (2007), limiting the amount of 'carry-in' interference (see Figure 3) using ideas from (Baruah, 2007).

An observation following from the mathematical analysis of Guan et al. (2009) concerns the pattern of task execution that results in the largest response time upper bound R_k^{UB}, for a job of task τ_k under global FP scheduling, computed using the sufficient response time

[2] D_k and C_k are included as parameters to show the similarity with subsequent response time equations, and to make clear the dependency on these values. This dependency is relevant to the discussion of optimal priority assignment in Section 4

[3] 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.

[4] Bertogna and Cirinei (2007) also investigated global EDF scheduling and the slack terms are necessary in that case.

analysis given in (Guan et al., 2009). We re-state this observation in Theorem 1 and provide a formal proof that it holds for exact response times.

To determine the exact worst-case response time of task τ_k, we potentially need to consider all possible patterns of job releases for high priority tasks. We note that as the times at which jobs of higher priority tasks execute are unaffected by the releases times and execution times of jobs of lower priority tasks, we can independently consider all possible alignments of a release of a job of task τ_k with respect to those patterns of higher priority execution. (For a constrained deadline task τ_k, we need only consider one job of τ_k).

Let us assume that all possible patterns of job releases of higher priority tasks (i.e. those in the set *hp*(*k*)) occur along a single discrete timeline. We can classify points on this timeline as belonging to the set $\Psi(k)$ as follows: time point $t \in \Psi(k)$ if and only if all *m* processors are busy executing tasks in *hp*(*k*) in the interval $[t, t+1]$, and during the preceding time interval $[t-1, t]$ at least one processor was not occupied by a higher priority task. In the following, we use *t* to refer to time points that are members of the set $\Psi(k)$ ($t \in \Psi(k)$) and *x* to refer to time points that are not members of the set ($x \notin \Psi(k)$).

Theorem 1: (Release time leading to the worst-case response time of task τ_k under global FP scheduling). For a sporadic task system scheduled under global FP scheduling on a multiprocessor, and a timeline including all valid patterns of job releases for higher priority tasks, there exists a time $t \in \Psi(k)$ (i.e. a time *t* when all *m* processors are busy executing tasks in *hp*(*k*) during the interval $[t, t+1]$, and during the preceding time interval $[t-1, t]$ at least one processor was not occupied by a higher priority task), such that release of task τ_k at time *t* results in the worst-case response time.

Proof: We show that for every time $x \notin \Psi(k)$, there exists a time $t \in \Psi(k)$ such that the response time of a job of task τ_k released at time *t* is at least as large as the response time of a job of task τ_k released at time *x*. There are two possibilities to consider:

Case 1: During the time interval $[x, x+1]$ all *m* processors are busy executing higher priority tasks, and have been occupied in this way since some time $t \in \Psi(k)$. As all *m* processors are occupied during the interval $[t, x]$, we may move the release time of the job of task τ_k back from time *x* to time *t* without changing its finishing time. The response time of the job is therefore greater when it is released at time *t* than when it is released at time *x*.

Case 2: During the interval $[x, x+1]$ not all *m* processors are busy executing higher priority tasks. The next time at which they will be occupied in this way is some time $t \in \Psi(k)$. As not all *m* processors are occupied during the interval $[x, t]$ the job of task τ_k must be executing during this interval. Moving the release time of the job forward from time *x* to time *t* must therefore increase its finishing time by at least *t*-*x*. Hence the job's response time is at least as great when released at time *t* as it is when released at time *x*

□

Corollary 1: Theorem 1 implies that in the worst-case scenario for task τ_k, at most *m*-1 higher priority tasks can contribute workload due to jobs that are released strictly prior to the start of the interval of interest (so called *carry-in* jobs) see Lemma 1 in (Guan et al., 2009).

Recall that Bertogna and Cirinei (2007) showed that if task τ_k is schedulable, then an upper bound on the interference due to a higher priority task τ_i with a carry-in job, in an interval of length R_k^{UB} is given by $I_i^R(R_k^{UB}, C_k)$ (Equation (7)), see Figure 3 below.

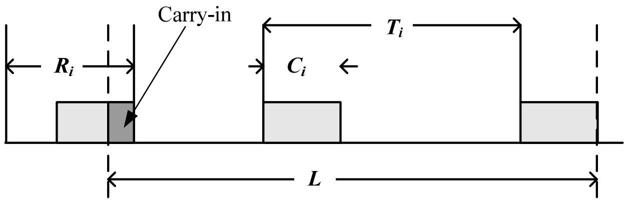

Figure 3: Interference in an interval with carry-in

However if task τ_i does not have a carry-in job, then Guan et al. (2009) showed that the worst-case interference occurs in the scenario shown in Figure 4, and is given by:

$$I_i^{NC}(L, C_k) = \min(W_i^{NC}(L), L - C_k + 1) \tag{9}$$

where:

$$W_i^{NC}(L) = N_i^{NC}(L)C_i + \min(C_i, L - N_i^{NC}(L)T_i) \tag{10}$$

and

$$N_i^{NC}(L) = \left\lfloor \frac{L}{T_i} \right\rfloor \tag{11}$$

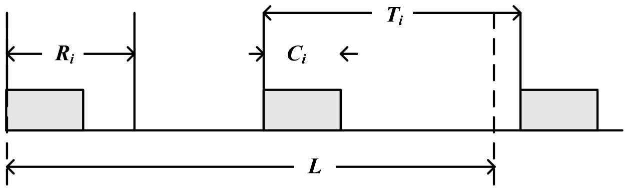

Figure 4: Interference in an interval: no carry-in

The difference between the two interference terms is:

$$I_i^{DIFF-R}(R_k^{UB}, C_k) = I_i^R(R_k^{UB}, C_k) - I_i^{NC}(R_k^{UB}, C_k) \tag{12}$$

Using this result, Guan et al. (2009) improved upon the response time test of Bertogna and Cirinei (2007) as follows:

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

where *Max*(*k*, *m*-1) is the set of *m*-1 tasks in *hp*(*k*) that have the largest values of $I_i^{DIFF}(R_k^{UB}, C_k)$. We refer to the schedulability test given by Equation (13) as the "RTA-LC test" (Response Time Analysis with Limited

Carry-in). We note that the RTA-LC test reduces to the RTA test if the $I_i^{DIFF-R}(R_k^{UB}, C_k)$ term is included for all of the higher priority tasks, rather than just those with the *m*-1 largest values.

The observation by Guan et al. (2009), restated in Theorem 1, means that the technique of limiting interference due to carry-in jobs can be applied to an interval of length D_k, and hence to the DA test of Bertogna et al. (2009), giving the following sufficient schedulability test for each task τ_k:

$$D_k \geq C_k + \left\lfloor \frac{1}{m}\left(\sum_{i \in hp(k)} I_i^{NC}(D_k, C_k) + \sum_{i \in Max(k,m-1)} I_i^{DIFF-D}(D_k, C_k) \right) \right\rfloor \quad (14)$$

where:

$$I_i^{DIFF-D}(D_k, C_k) = I_i^{D}(D_k, C_k) - I_i^{NC}(D_k, C_k) \quad (15)$$

and $I_i^{D}(D_k, C_k)$ and $I_i^{NC}(D_k, C_k)$ are given by Equations (3) and (9) respectively. We refer to the schedulability test given by Equation (14) as the "DA-LC test" (Deadline Analysis with Limited Carry-in). We note that the DA-LC test reduces to the DA test if the $I_i^{DIFF-D}(D_k, C_k)$ term is included for all of the higher priority tasks, rather than just those with the *m*-1 largest values.

3.1. An upper bound on the RTA-LC test

In this section, we derive a pseudo-schedulability condition, called the C-RTA condition that dominates the RTA-LC test. It is important to note that the C-RTA condition is *not* a schedulability test. It may deem some priority ordered tasksets schedulable that are in fact unschedulable.

The C-RTA[5] condition is formed from Equation (13) by using the smallest possible value that the response time upper bound of each higher priority task could take (i.e. by using C_i instead of R_i^{UB} in Equations (5) and (6)). We observe that as Equation (5) is monotonically non-decreasing in R_i^{UB} and the minimum possible value for R_i^{UB} is C_i, the C-RTA condition dominates the RTA-LC test. Thus the C-RTA condition forms an upper bound on task schedulability under the RTA-LC test. We will return to this point in Sections 4 and 6.

3.2. Complexity and comparability

We note that the RTA and RTA-LC tests (and the C-RTA condition) are pseudo-polynomial in complexity, $O(n^2 D_{\max})$ for a taskset of cardinality *n*, with longest deadline $D_{\max}$, while the DA and DA-LC tests are polynomial in complexity, $O(n^2)$.

The following comparability relationships hold between the various schedulability tests, shown in Figure 5:

(i) the RTA-LC test given by Equation (13) dominates the RTA test, Equation (8);
(ii) the DA-LC test given by Equation (14) dominates the DA test, Equation (4);
(iii) the RTA-LC test dominates the DA-LC test;
(iv) the RTA test dominates the DA test;
(v) the DA-LC and RTA tests are incomparable.

Figure 5 also depicts the set of priority ordered tasksets deemed schedulable by the C-RTA condition. Note this includes some unschedulable tasksets.

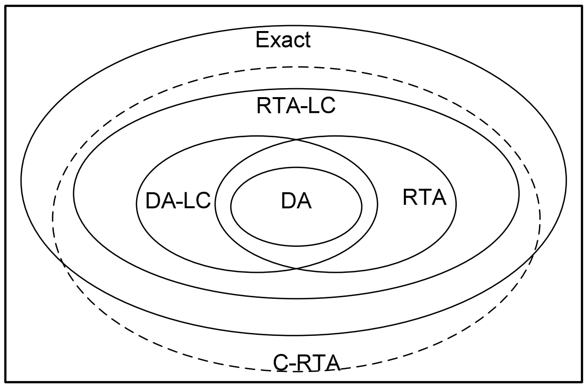

Figure 5: Comparability relationships between schedulability tests

The comparability relationships between the various schedulability tests are illustrated by the priority ordered taskset given in Table 1.

Table 1: Taskset used to illustrate the comparability relationships between schedulability tests

Task (priority)	*C*	*D=T*
τ_1	3	10
τ_2	3	10
τ_3	4	10
τ_4	4	10
τ_5	1	See text

Table 2 gives the schedulability of the taskset in Table 1, according to each of the schedulability tests, assuming a two processor system, and a range of values for the deadline (and period) of the lowest priority task, τ_5. This simple example illustrates the four dominance relationships (i) to (iv) stated above, as well as the incomparability of the RTA and DA-LC tests.

Table 2: Schedulability of the taskset in Table 1 according to the four schedulability tests

	Deadline of τ_5		
Schedulability test	**10**	**12**	**15**
RTA-LC	✓	✓	✓
RTA	✗	✗	✓
DA-LC	✓	✗	✗
DA	✗	✗	✗

We observe the following interesting equivalence of the RTA and RTA-LC tests.

Theorem 2: The upper bound response times computed by

[5] The C-RTA condition is the same with or without limiting carry-in interference, hence we drop the "-LC".

the RTA test (Equation (8)) and the RTA-LC test (Equation (13)) are the same for the 2m highest priority tasks in any taskset with cardinality $n \geq 2m$.

Proof: We consider two cases:

Case 1: Tasks with priorities from 1 to m. According to both the RTA and RTA-LC tests, the m highest priority tasks all have response time upper bounds equal to their worst-case execution times $(R_k^{UB} = C_k)$. This can be seen by considering the behaviour of Equations (8) and (13) starting with an initial value of $R_k^{UB} = C_k$. The maximum value of $I_i^R(C_k, C_k)$ for each higher priority task τ_i is 1 (see Equation (7)), and hence with at most m-1 higher priority tasks, the maximum value of the floor function in Equations (8) and (13) is $\lfloor (m-1)/m \rfloor = 0$. Hence the fixed point iteration immediately terminates, returning a value of $R_k^{UB} = C_k$.

Case 2: Tasks with priorities from m+1 to 2m. Let us consider the interference from each of the m highest priority tasks τ_i, where $1 \leq i \leq m$, on some lower priority task τ_k, where $m+1 \leq k \leq 2m$. From Case 1, we know that $R_i^{UB} = C_i$. Substituting these values into Equations (5) and (6) reduces them to Equations (10) and (11) respectively. Hence, for each of the m highest priority tasks τ_i, we have $I_i^{DIFF-R}(L, C_k) = I_i^R(L, C_k) - I_i^{NC}(L, C_k) = 0$ for any value of L. Therefore, when task τ_k has a priority in the range m+1 to 2m, at most (2m-1) – m = m-1 tasks with priorities higher than k have a non-zero I_i^{DIFF-R} term that could contribute to Equation (13). Limiting the number I_i^{DIFF-R} terms included to the largest m-1 of them therefore excludes no non-zero terms, and so Equation (13) reduces to Equation (8). Hence the upper bound response times computed by Equation (13) and Equation (8) are the same for each task τ_k with priority from m+1 to 2m

□

Corollary 2: The RTA test (Equation (8)) and the RTA-LC test (Equation (13)) are equivalent for any priority ordered taskset with cardinality $n \leq 2m$. (This is why we needed a taskset of cardinality 5 (see Table 1) to highlight the differences between the RTA and RTA-LC tests, assuming a two processor system).

4. Priority assignment

Andersson and Jonsson (2000a) made the following observation about periodic tasksets:

"For fixed priority pre-emptive global multiprocessor scheduling, there exist task sets for which the response time of a task depends not only on T_i and C_i of its higher-priority tasks, but also on the relative priority ordering of those tasks".

Andersson and Jonsson (2000a) concluded that even if an exact schedulability test were known[6], then it would not be possible to use Audsley's OPA algorithm (Audsley, 1991, 2001) to determine the optimal priority ordering. While this is undoubtedly true, we believe that it has also lead to a common misconception that the OPA algorithm cannot be applied to schedulability tests for global FP scheduling.

[6] Note, such tests are now known for periodic tasksets (Cucu and Goossens 2006, 2007).

In this section, we show that the OPA algorithm is applicable to the multiprocessor case provided that the schedulability test used meets three simple conditions. These conditions allow us to classify schedulability tests for global FP scheduling into two categories: OPA-compatible and OPA-incompatible. First we provide an overview of the OPA algorithm (Audsley, 1991, 2001) originallyderived for uniprocessor systems.

4.1. Optimal priority assignment

The pseudo code for the OPA algorithm, using some compatible schedulability test S is given below.

```
Optimal Priority Assignment Algorithm
for each priority level k, lowest first
{
   for each unassigned task τ
   {
       if τ is schedulable at priority k
       according to schedulability test S
       with all unassigned tasks assumed to
       have higher priorities
       {
          assign τ to priority k
          break (continue outer loop)
       }
   }
    return unschedulable
}
return schedulable
```

For n tasks, the algorithm performs at most $n(n+1)/2$ schedulability tests and is guaranteed to find a priority assignment that is schedulable according to schedulability test S, if one exists. This is a significant improvement over inspecting all $n!$ possible priority orderings. Note that the OPA algorithm does not specify the order in which tasks should be tried at each priority level.

For a schedulability test S to be compatible with the OPA algorithm, it must comply with three conditions stated below. These conditions refer to properties or attributes of the tasks which make up the taskset. Task properties are referred to as *independent* if they have no dependency on the priority assigned to the task. For example in the sporadic task model used in this paper, the worst-case execution time, deadline, and minimum inter-arrival time are all independent properties of a task, while the worst-case response time depends on the task's priority and so is a *dependent* property.

Condition 1: The schedulability of a task τ_k may, according to test S, depend on any independent properties of tasks with priorities higher than k, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 2: The schedulability of a task τ_k may, according to test *S*, depend on any independent properties of tasks with priorities lower than *k*, but not on any properties of those tasks that depend on their relative priority ordering.

Condition 3: When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test *S*, if it was previously schedulable at the lower priority. (As a corollary, the task being assigned the lower priority cannot become schedulable according to test *S*, if it was previously unschedulable at the higher priority).

We now prove the following theorem about the applicability of the OPA algorithm to global FP scheduling. Theorem 3 and its proof make no assumptions about the schedulability test *S*, save that it complies with Conditions 1-3.

Theorem 3: The Optimal Priority Assignment (OPA) algorithm is an *optimal priority assignment policy* (see Definition 1) for any global FP schedulability test *S* compliant with Conditions 1-3.

Proof: We prove that for every taskset *X* that is schedulable according to test *S* with some arbitrary priority ordering Q, the OPA algorithm is able to generate a priority ordering *P* that is also schedulable according to test *S*.

In the proof, we will show that when applied to taskset *X*, each iteration *k* of the OPA algorithm, from priority level *n* down to 1, is able to find a task that is schedulable according to test *S*. Thus the OPA algorithm is able to generate a complete priority ordering *P* for taskset *X* that is schedulable according to test *S*.

For the purposes of the proof, we refer to priority ordering Q as Q_n. Over the *n* iterations, we will transform Q_n into $Q_{n-1} \dots Q_0$, where Q_0 is identical to the priority ordering *P* generated by the OPA algorithm. The transformation will be such that after each iteration *k* from *n* down to 1, the transformed priority ordering Q_{k-1} remains schedulable according to test *S*, and the tasks at priority levels *k* and below are the same in Q_{k-1} as in *P*.

We now introduce a concise notation to aid in the discussion of tasks and groups of tasks within a priority ordering:

- $Q_k(i)$ is the task at priority level *i* in priority ordering Q_k.
- $hep(i, Q_k)$ is the set of tasks with priority higher than or equal to *i* in priority ordering Q_k.
- $hp(i, Q_k)$ is the set of tasks with priority strictly higher than *i* in priority ordering Q_k.
- $lep(i, Q_k)$ is the set of tasks with priority lower than or equal to *i* in priority ordering Q_k.
- $lp(i, Q_k)$ is the set of tasks with priority strictly lower than *i* in priority ordering Q_k.

In the proof that follows, we use *k* to represent both the iteration of the OPA algorithm, i.e. the priority level examined (so initially, *k* = *n*), and also the index for the transformed priority ordering. Note that priority ordering *P* is built up over the *n* iterations, with a task assigned at priority *k* on each iteration of the OPA algorithm. Unassigned tasks are assumed to have higher priority than the priority level currently being examined.

The proof proceeds by iterating over values of *k* from *n* to 1: At the start of each iteration *k*, all tasks in priority ordering Q_k are known to be schedulable according to test *S*.

As the tasks with lower priority than *k* are the same in both Q_k and *P* ($lp(k, Q_k) = lp(k, P)$ and initially $lp(n, Q_n) = lp(n, P) = \phi$), then it follows that $hep(k, Q_k)$ = $hep(k, P)$. Given Condition 1 and the fact that Q_k is a schedulable priority ordering according to test *S*, on iteration *k* the OPA algorithm is guaranteed to find a task in the set of unassigned tasks (i.e. $hep(k, P)$ = $hep(k, Q_k)$) that is schedulable at priority *k* according to test *S*. We note that one such task is $Q_k(k)$. The task chosen by the OPA algorithm is designated $P(k)$.

There are two cases that need to be considered:

1. $P(k)$ is the same as $Q_k(k)$, in which case no transformation is necessary to form priority ordering Q_{k-1} ($Q_{k-1} = Q_k$) and hence Q_{k-1} is trivially a schedulable priority ordering.
2. The OPA algorithm chose a different schedulable task; in other words $P(k)$ is the task at some higher priority level *i* in Q_k, i.e. $Q_k(i)$. In this case, we transform Q_k into Q_{k-1} by moving task $Q_k(i)$ down in priority from priority level *i* to priority level *k* and the tasks in Q_k at priority levels $i+1$ to *k* up one priority level, as illustrated in Figure 6.

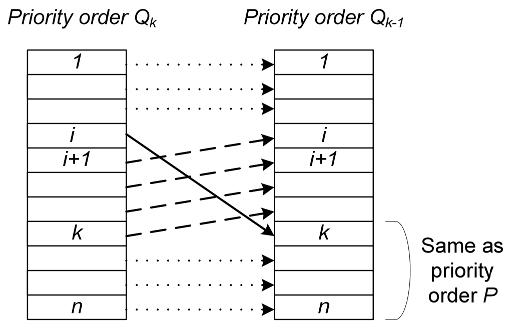

Figure 6

Comparing the tasks in priority order Q_{k-1} with their counterparts in Q_k. There are effectively four groups of tasks to consider:

1. $hp(i, Q_{k-1})$: These tasks are assigned the same priorities in both Q_k and Q_{k-1}, given Condition 2, all of these tasks remain schedulable.
2. $hp(k, Q_{k-1}) \cap lep(i, Q_{k-1})$: These tasks retain the same partial order but are shifted up one priority level in Q_{k-1}. This shift in priority can be achieved by repeatedly swapping the priorities of task $P(k)$ and the task immediately below it in the priority order, until task $P(k)$ reaches priority *k*. Hence, given Condition 3, all the tasks increasing in priority, i.e.

those in the set $hp(k, Q_{k-1}) \cap lep(i, Q_{k-1})$, remain schedulable.

3. Task $Q_{k-1}(k) = Q_k(i) = P(k)$: The tasks of lower priority than k are the same in both Q_k and *P*, hence $hep(k, P) = hep(k, Q_k)$. The OPA algorithm selected task $P(k)$ from the set of tasks $hep(k, Q_k)$ on the basis that it is schedulable at priority *k* with the set of tasks $hep(k, Q_k) - \{Q_k(i)\}$ = $hp(k, Q_{k-1})$ at higher priorities. Given Condition 1, task $Q_{k-1}(k) = P(k)$ is schedulable at priority *k*, irrespective of the priority order of the tasks in $hp(k, Q_{k-1})$ and therefore it remains schedulable in priority order Q_{k-1}.
4. $lp(k, Q_{k-1})$: These tasks are assigned the same priorities in both Q_k and Q_{k-1}. Given Condition 1 and the fact that $hep(k, Q_{k-1}) = hep(k, Q_k)$, all of the tasks in $lp(k, Q_{k-1})$ remain schedulable according to test *S*.

The above analysis shows that every task in Q_{k-1} remains schedulable according to test *S*. A total of *n* iterations of the above process (for *k* = *n* down to 1) correspond to iteration of the OPA algorithm over all *n* priority levels. On each iteration the OPA algorithm is able to identify a task that is schedulable according to test *S* and therefore generate a priority ordering *P* that is schedulable according to test *S*

□

The proof of Theorem 3 shows that compliance with Conditions 1-3 is sufficient for schedulability test *S* to be OPA-compatible. We now show that each of the three conditions is also necessary.

Theorem 4: (Non-optimality of the OPA algorithm for schedulability tests that are OPA-incompatible). For a schedulability test *S* that is non-compliant with one or more of Conditions 1-3, the OPA algorithm may fail to generate a priority ordering that is deemed schedulable by test *S*, when such an ordering exists.

Proof:

Necessity of Condition 1: The OPA algorithm does not specify the priority ordering of unassigned tasks; therefore when determining the schedulability of a task *A* at priority *k*, schedulability test *S* is effectively free to choose any arbitrary priority ordering for the unassigned (higher priority) tasks. Let us assume that with the arbitrary priority ordering chosen for the unassigned tasks, task *A* is deemed schedulable at priority *k*, and it is therefore assigned to that priority level. If Condition 1 does *not* hold, then a different priority ordering later established by the OPA algorithm for the higher priority tasks can result in task *A* becoming unschedulable at priority *k* according to test *S*. In this case, the priority ordering found by the OPA algorithm is erroneous; it is not in fact schedulable according to test *S*. However, a different choice of priorities for the higher priority tasks would result in task *A* being schedulable, and thus a schedulable priority ordering existed, yet it was not found by the OPA-algorithm.

Necessity of Condition 2: If Condition 2 does not hold then the schedulability of a task according to test *S* is dependent on the priority order of lower priority tasks. In this case, the OPA algorithm could place tasks at priorities lower than *k* in an order that results in no task being schedulable at priority level *k*. Yet, if the lower priority tasks were placed in a different priority order, then a task could be found that was schedulable at priority *k* according to test *S*. In this case, the OPA-algorithm fails to find a priority ordering that is schedulable according to test *S* when such a priority ordering exists.

Necessity of Condition 3: If Condition 3 does not hold, then two tasks *A* and *B* may both be schedulable according to test *S* when assigned the lowest priority; however task *B* may be unschedulable when assigned the next highest priority. Let us assume that the OPA algorithm arbitrarily chooses to assign task *A* to the lowest priority. In this case, no tasks are found that are schedulable at the next highest priority. Thus the OPA-algorithm fails to find a priority ordering that is schedulable according to test *S*, even though one exists; with task *B* at the lowest priority

□

Condition 2 holds for all of the schedulability tests considered in this paper. These tests deal with pre-emptive scheduling of independent tasks, hence the schedulability of higher priority tasks is independent of lower priority tasks. We note that Condition 2 is important when considering non-pre-emptive scheduling and task models which permit access to mutually exclusive shared resources.

We note that Theorem 3 depends on emergent properties of the schedulability test, and not on the specific properties of the underlying task model. It is therefore applicable to both periodic and sporadic task models.

Conditions 1-3 enable us to classify global FP schedulability tests as either OPA-compatible or OPA-incompatible.

Theorem 5: Any exact schedulability test for a general periodic taskset is OPA-incompatible.

Proof: It suffices to show that Condition 1 does not hold for any exact test. Consider the following synchronous periodic taskset with four tasks, two copies of task *A* = {1, 2, 3} and two copies of task *B* = {2, 4, 4}, executing on a two processor system. (The parameters are the task's worst-case execution time, deadline, and period respectively). Task *B* is schedulable at the lowest priority, with the other tasks in priority order *A*, *A*, *B*, but not schedulable when they are in priority order *A*, *B*, *A* or *B*, *A*, *A*. This can be seen by examining the schedule over the hyperperiod. In effect, both the exact schedulability and the exact response time of task *B* at the lowest priority level are dependent on the relative priority ordering of the higher priority tasks. As all exact schedulability tests must by definition provide an identical classification of all priority ordered tasksets as schedulable or unschedulable it follows that all exact schedulability tests for periodic tasksets are OPA-incompatible □

Theorem 6: The RTA test (Bertogna and Cirinei, 2007) (Equation (8)) for global FP scheduling of sporadic tasksets is OPA-incompatible.

Proof: It suffices to show that Condition 1 does not hold for the RTA test. The workload $W_i^R(L)$ (Equation (5)) used to determine schedulability via these tests depends on the response times of higher priority tasks, which in turn depend on the relative priority ordering of those tasks. This can be seen by considering the following example comprising four tasks: two copies of task *A* = {10, 20, 20}, task *B* = {10, 20, 100}, and task *C* = {20, 55, 55}, executing on a two processor system. With priority order *A*, *A*, *B*, *C* the taskset is deemed schedulable by the test with upper bounds on task response times of 10, 10, 20, and 55 respectively. However, if the priority order is instead *A*, *B*, *A*, *C*, then the copy of task A at priority 3 has an increased upper bound response time of 20 (it was 10 at priority 2). This increases its workload and interference on task *C* which is then deemed unschedulable
□

Theorem 7: The RTA-LC test (Guan et al., 2009) (Equation (13)) for global FP scheduling of sporadic tasksets is OPA-incompatible.

Proof: Follows directly from the proof of Theorem 6, noting that the RTA-LC and RTA tests are equivalent for the 4 task, 2 processor counter example used (Corollary 2)
□

Theorem 8: The response time test of Andersson and Jonsson (2000a) (Equation (16) below) for global FP scheduling of sporadic tasksets is OPA-compatible:

$$R_k^{ub} \leftarrow C_k + \frac{1}{m} \sum_{\forall i \in hp(k)} \left(\left\lceil \frac{R_k^{ub}}{T_i} \right\rceil C_i + C_i \right) \tag{16}$$

Proof: It suffices to show that Conditions 1-3 hold.

Inspection of Equation (16) shows that the upper bound response time R_k^{ub} computed for task τ_k depends on the set of higher priority tasks, but not on their relative priority ordering, hence Condition 1 holds.

R_k^{ub} (Equation (16)) has no dependency on the set of tasks with lower priority than *k*, hence Condition 2 holds.

Consider two tasks *A* and *B* initially at priorities *k* and *k*+1 respectively. The upper bound response time of task *B* cannot increase when it is shifted up one priority level to priority *k*, as the only change in the response time computation (Equation (16)) is the removal of task *A* from the set of tasks that have higher priority than task *B,* hence Condition 3 holds
□

Theorem 9: The DA test (Bertogna et al., 2009) (Equation (4)) for global FP scheduling of sporadic tasksets is OPA-compatible:

Proof: Follows the same logic as the proof of Theorem 8, with the upper bound response time given by the right hand side of Equation (4) rather than Equation (16)
□

Theorem 10: The DA-LC test (Equation (15)) for global FP scheduling of sporadic tasksets is OPA-compatible.

Proof: Follows the same logic as the proof of Theorem 8, with the upper bound response time given by the right hand side of Equation (14) rather than Equation (16)
□

Theorem 11: The C-RTA condition is OPA-compatible.

Proof: It suffices to show that Conditions 1-3 hold.

Inspection of Equation (13) and its component equations shows that the upper bound response time R_k^{UB} computed for task τ_k depends on the set of higher priority tasks, and their parameters (C_i, D_i, T_i) but not on their upper bound response times (as C_i is substituted for R_i^{UB} in Equations (5) and (6)) or their relative priority ordering, hence Condition 1 holds.

Equation (13) has no dependency on the set of tasks with priorities lower than *k*, hence Condition 2 holds.

Consider two tasks *A* and *B* initially at priorities *k* and *k*+1 respectively. The upper bound response time of task *B* cannot increase when it is shifted up one priority level to priority *k*, as the only change in the response time computation (Equation (13)) is the removal of task *A* from the set of tasks that have higher priority than task *B,* hence Condition 3 holds
□

As the C-RTA condition is both OPA-compatible and dominates the RTA-LC schedulability test, it follows that any taskset that is deemed unschedulable according to the OPA-algorithm using the C-RTA condition is guaranteed to also be unschedulable according to the RTA-LC test for every possible priority assignment. We can therefore use the C-RTA condition combined with the OPA algorithm to provide an upper bound on the potential performance of the RTA-LC test with optimal priority assignment.

We observe that the processor load based schedulability test for global FP scheduling given by Fisher and Baruah, (2006) (see Theorem 2, Equation (9) of (Fisher and Baruah, 2006)) is OPA-compatible as the schedulability of each task depends only on the set of higher priority tasks, and not on their relative priority order. Note that the simpler form of that test, given by Corollary 1 of (Fisher and Baruah, 2006)), is applicable only to tasksets using DMPO.

The processor load based test given by Baruah and Fisher (2008) (see Corollary 1 of (Baruah and Fisher, 2008)) is also OPA-compatible; however, Baruah and Fisher proved that DMPO is the optimal priority assignment policy with respect to that test.

Finally, the processor load based test given by Baruah (2009) (see Theorem 3 in (Baruah, 2009)) is applicable only to tasksets in DMPO.

4.2. Heuristic priority assignment

In this section, we investigate heuristic priority assignment policies.

In his thesis, Bertogna (2007) evaluates the effectiveness of a number of different schedulability tests. Bertogna's experiments show that using DMPO the RTA test for global FP scheduling outperforms all other then known schedulability tests for constrained deadline sporadic tasksets, including those for EDF and EDZL. Despite this, and the optimality of DMPO in the equivalent uniprocessor case, we are sceptical about the effectiveness of DMPO in the multiprocessor case.

The intuition for an alternative priority assignment policy can be obtained by re-arranging Equation (4):

$$D_k - C_k \geq \left\lfloor \frac{1}{m} \sum_{\forall i \in hp(k)} I_i^D(D_k, C_k) \right\rfloor \tag{17}$$

For large m, the term on the right hand side grows relatively slowly with each additional higher priority task. This suggests that $D_i - C_i$ monotonic priority ordering (D-CMPO) might be a useful heuristic.

Andersson and Jonsson (2000b) investigated a similar priority ordering, called TkC, for implicit deadline tasksets. TkC assigns priorities based on the value of $T_i - kC_i$, where k is a real value computed on the basis of the number of processors, as follows:

$$k = \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{2m} \tag{18}$$

Extending this approach to tasksets with constrained deadlines, we form the DkC priority assignment policy which orders tasks according to the value of $D_i - kC_i$, where k is again computed according to Equation (18).

The performance of the three heuristic priority assignment policies: DMPO, D-CMPO, and DkC is examined empirically in Section 6.

We also developed heuristic priority assignment algorithms based on the DM-DS$\{\varsigma\}$ (Bertogna et al., 2005) and SM-US$\{\varsigma\}$ (Andersson, 2008) priority assignment policies. These algorithms, although more complex, were found in general to be no more effective than the DkC policy. (Appendix A gives details of these policies and their performance).

It is interesting to note that D-CMPO and DkC have some similarities with recent work on dynamic priority scheduling: The LEDLm algorithm proposed by Easwaran et al. (2008), partially schedules jobs on the basis of longest remaining execution time first. This has the effect of maximising the potential for concurrency by retaining a large number of incomplete jobs with short remaining execution times; the idea being that such jobs are easier to schedule than a smaller number of jobs with longer remaining execution times. D-CMPO and DkC incorporate an element of this effect, as by comparison with DMPO, they assign higher priorities to tasks with longer execution times.

5. Taskset generation

Empirical investigations into the effectiveness of priority assignment policies and schedulability tests require a means of generating tasksets. A taskset generation algorithm should be unbiased (Bini and Buttazzo, 2005), and ideally, it should allow tasksets to be generated that comply with a specified parameter setting. That way the dependency of priority assignment policy / schedulability test effectiveness on each taskset parameter can be examined by varying that parameter, while holding all other parameters constant, avoiding any confounding effects.

A (naïve) unbiased method of generating tasksets of cardinality n and target utilisation (Ut) is as follows.

1. Select n task utilisation values U_i at random from a uniform distribution over the range [0,1].
2. Discard the taskset if the total utilisation U is not within some small percentage of Ut, and generate a new taskset by returning to step 1.

We note that this naive approach is not viable in practice due to the number of tasksets that need to be discarded. The UUnifast algorithm of Bini and Buttazzo (2005) (pseudo code given below), was devised to give the same unbiased distribution. Note, pow(x, y) raises x to the power y, and rand() returns a random number in the range [0,1] from a uniform distribution.

```
UUnifast(n,Ut)
{
  SumU = Ut;
  for (i = 1 to n-1)
  {
     nextSumU = SumU * pow(rand(), 1/(n-i));
     U[i] = SumU – nextSumU;
     sumU = nextSumU;
  }
  U[n] = SumU;
}
```

To the best of our knowledge, UUnifast has not previously been used in the context of multiprocessors, as the basic algorithm cannot generate tasksets with total utilisation $U > 1$ without the possibility that some tasks will have utilisation $U_i > 1$. Instead, researchers have used an approach to taskset generation based on generating an initial taskset of cardinality m+1 at random and then repeatedly adding tasks to it to generate further tasksets until the total utilisation exceeds the available processing resource (Bertogna, 2007; Bertogna et al., 2009; Bertogna and Cirinei, 2007; Baker et al., 2008). This approach has the disadvantage that it effectively combines two variables, utilisation and taskset cardinality, and does not necessarily result in an unbiased distribution of task utilisation values.

In the remainder of this section, we show how the UUnifast algorithm can be adapted to generate the tasksets needed to study multiprocessor systems. Inspection of the UUnifast algorithm shows that it is scale invariant. We can therefore use it to generate tasksets with $U > 1$ as follows:

- The UUnifast method, with parameters n, and Ut (which may be > 1), is used to generate task utilisation values in the range [0, Ut].
- If a task utilisation value U_i is generated that is > 1, then the values produced so far, that is U_1 to U_i, are discarded. If the total number of discarded partial tasksets exceeds some *limit*, then the algorithm exits and reports that it has failed, otherwise it re-starts generating utilisation values at U_1.
- Once a sequence of n valid utilisation values are generated, the algorithm completes, reporting success.

We refer to the above algorithm as UUnifast-Discard.

Theorem 12: The tasksets produced by UUnifast-Discard are unbiased, i.e. uniformly distributed (Bini and Buttazzo, 2005), with task utilisations in the range [0, min(Ut,1)] which sum to Ut.

Proof: We prove the theorem via a geometric argument**.** Each taskset can be represented by a point on an n-1 dimensional plane in n-dimensional space, where the co-ordinates of the point are the utilisation values of each task in the taskset i.e. $(U_1, U_2, U_3 .. U_n)$. A uniform distribution of tasksets is required over the valid region of the plane.

UUnifast produces tasksets that are uniformly distributed over a finite convex region Z of the n-1 dimensional plane defined by $\sum U_i = Ut$, $U_i \le Ut$ and $U_i \ge 0$. (See Figure 9 in (Bini and Buttazzo, 2005) for a graphical illustration).

For UUnifast-Discard, there are two cases to consider:
Case 1: $Ut \le 1$: No tasksets are discarded; hence the distribution of tasksets remains uniform and unbiased.
Case 2: $Ut > 1$: Let Y be the convex finite region of the n-1 dimensional plane defined by $\sum U_i = Ut$, $U_i \le 1$ and $U_i \ge 0$. Note that Y is a subset of Z and so the distribution of tasksets produced by UUnifast over the region Y is also uniform and unbiased. Now, all tasksets generated with any $U_i > 1$ are discarded by UUnifast-Discard. This corresponds to removal of all of those tasksets that are in region Z but not in region Y. Further, none of the tasksets in region Y are discarded, hence the distribution of tasksets over region Y remains uniform and unbiased
□

An unbiased distribution of task utilisation values is exactly what is required to study the effectiveness of multiprocessor schedulability tests. Unfortunately, there is a drawback to the UUnifast-Discard approach. As the target utilisation requested increases towards n/2, then the number of valid tasksets (with all $U_i \le 1$) becomes a vanishingly small proportion of those generated. While this is clearly a limitation in theory, in practice, we contend that many commercial real-time systems using multiprocessors will have significantly more tasks than processors. In any case, we can simply set a pragmatic discard limit for UUnifast-Discard and investigate as much of the problem space as possible within this limit.

Figure 7 shows the maximum taskset utilisation that UUnifast-Discard is able to generate, using discard limits of 10,100, 1000, and 10,000 respectively, plotted against taskset cardinality. This graph was generated empirically, by running the UUnifast-Discard algorithm for increasing values of target utilisation for each value of taskset cardinality until the algorithm reached the discard limit and was thus unable to generate the 100 tasksets required by the experiment in the permitted number of iterations (given by the discard limit multiplied by the number of tasksets required).

Figure 7 shows that, UUnifast-Discard, with a discard limit of 1000, can be used to generate tasksets with a target utilisation of up to 8, (suitable for investigation of 8 processor systems) provided that the taskset cardinality exceeds 14. Lower utilisation levels of 7.5 and 6.7 are possible with 12 and 10 tasks respectively. (Note that the behaviour of the UUnifast-Discard algorithm is independent of the number of processors).

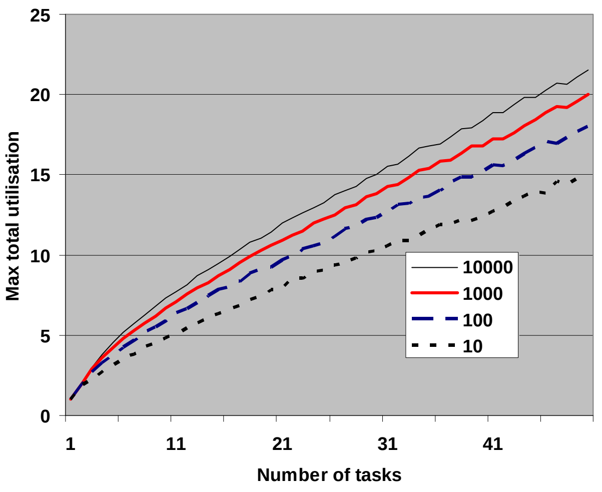

Figure 7

As we will see in the next section, the scope of this taskset generation method is sufficient to examine the effectiveness of schedulability tests over a wide range of interesting parameter values.

6. Empirical investigation

In this section, we present the results of an empirical investigation, examining the effectiveness of different priority assignment policies when used in conjunction with two sufficient schedulability tests: the DA-LC test (Equation (14)), which is OPA-compatible, and the RTA-LC test (Equation (13)), which is OPA-incompatible. The priority assignment policies studied are DMPO, D-CMPO, DkC and the OPA algorithm (DA-LC test only).

We also used the C-RTA condition combined with the OPA algorithm to determine an upper bound on the potential performance of the RTA-LC test for any priority ordering. It is however important to remember that the C-RTA condition is not a schedulability test, it can only tell us which tasksets are definitely unschedulable. Some of the tasksets that it apparently deems schedulable may in fact be unschedulable. Further, the optimism inherent in the C-RTA condition means that the actual performance of the RTA-LC test assuming optimal priority assignment is likely to be some way below the bound shown in the graphs.

We also compared the performance of the tests for global FP scheduling with the iterative response time test for global EDF scheduling given by Bertogna and Cirinei (2007). We refer to this test as EDF-RTA. EDF-RTA is arguably the most effective schedulability test currently available for global EDF scheduling. (We note that by combining a number of incomparable schedulability tests for global EDF scheduling including EDF-RTA, a slightly large number of schedulable tasksets can be detected than using EDF-RTA alone, see (Bertogna, 2009) for further details).

Note the figures in this section are best viewed online in colour.

6.1. Parameter generation

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

- Task utilisations were generated using the UUnifast-Discard algorithm, using a discard limit of 1000.
- Task periods were generated according to a log-uniform distribution[7] 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. The log-uniform distribution was used as it generates an equal number of tasks in each time band (e.g. 1-10ms, 10-100ms etc.), thus providing reasonable correspondence with real systems.
- Task execution times were set based on the utilisation and period selected: $C_i = U_i T_i$.
- Task deadlines were assigned according to a uniform random distribution, in the range $[C_i, T_i]$.

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 various combinations of priority assignment policy and schedulability test. The graphs plot the percentage of tasksets generated that were deemed schedulable in each case.

6.2. Experiment 1 (Priority assignment)

In this experiment we investigated the impact of each of the priority assignment policies on the percentage of tasksets deemed schedulable by the different schedulability tests. Figures 7 to 10 show this data for 2, 4, 8, and 16 processors respectively. In each case, the number of tasks was set to 5 times the number of processors.

From the graphs, we can see that the priority assignment policy used has a significant impact on overall performance, and that the more processors there are, the larger this impact becomes. There are four broad solid lines on each graph depicting the performance of the DA-LC test for DMPO (lowest performance with respect to this schedulability test), D-CMPO, DkC, and OPA (highest performance / optimal with respect to this schedulability test). The uppermost (thin) solid line on each graph represents the C-RTA condition combined with OPA. This line upper bounds the potential performance of the state-of-the-art RTA-LC schedulability test combined with optimal priority assignment. Note the lines on the graphs appear in the order given in the legend.

In the 16 processor case (Figure 11), using DMPO, approx. 50% of the tasksets are unschedulable according to the DA-LC test at a utilisation level of 4.4 (= 0.28*m*); however, using the OPA algorithm, approx. 50% of the tasksets are schedulable according to the same test at a utilisation level of 9.6 (= 0.6*m*). Hence, in this case, optimal priority assignment effectively enables 118% better utilisation of the processing resource than DMPO. D-CMPO is more effective than DMPO, and the DkC priority assignment policy is notably significantly more effective again. Note, the performance of DkC and D-CMPO are identical in the 2 processor case (Figure 8) as $k = 1$ in Equation (18). Comparison between the four figures shows that the difference between OPA and DMPO becomes considerably larger as the number of processors increases.

It is clear from the graphs that the difference in performance between the DA-LC test (solid lines) and the RTA-LC test (dashed lines) is much less significant than the difference between the best and the worst priority assignment policies studied.

It is noticeable that the EDF-RTA test for global EDF scheduling results in performance that is similar to that of the DA-LC test for global FP scheduling using DMPO in the 16 processor case (Figure 11), and generally inferior in the case of fewer processors. (The two tests are incomparable).

It is clear from the graphs that the performance of the DA-LC test combined with the OPA algorithm is relatively close to the upper bound on the potential performance of the RTA-LC test with optimal priority assignment, given by the C-RTA condition.

[7] The log-uniform distribution of a variable *x* is such that *ln* (*x*) has a uniform distribution.

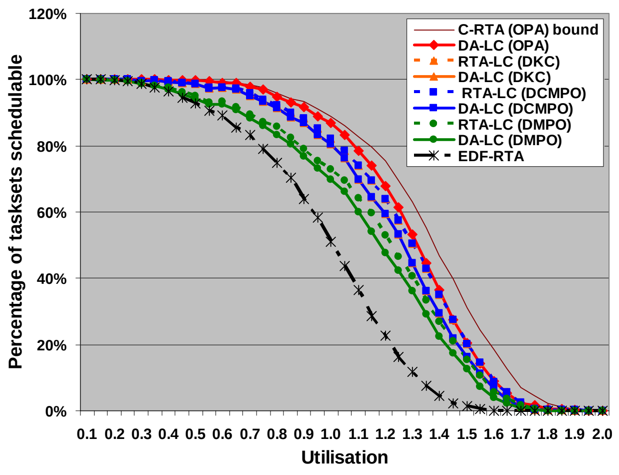

Figure 8: (2 processors, 10 tasks)

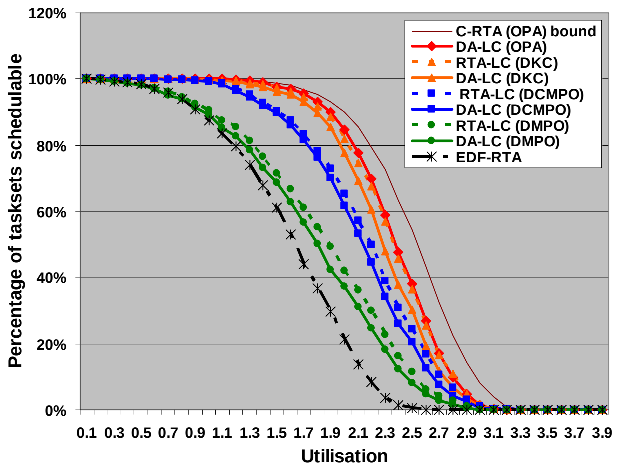

Figure 9: (4 processors, 20 tasks)

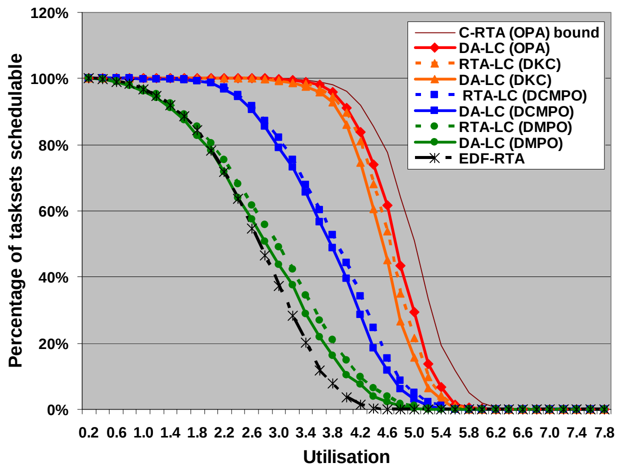

Figure 10: (8 processors, 40 tasks)

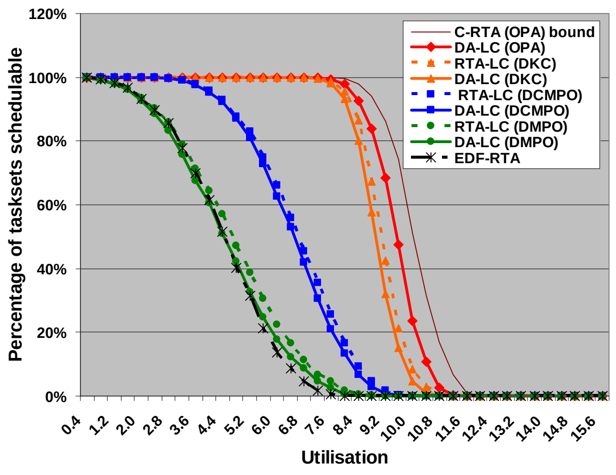

Figure 11: (16 processors, 80 tasks)

We repeated the same experiment for tasksets with implicit deadlines, using exactly the same parameter settings for taskset generation, save that task deadlines were set equal to their periods. In this case, DMPO reduces to Rate Monotonic priority ordering (Serlin, 1972; Liu and Layland, 1973), and DkC reduces to TkC (Andersson and Jonsson, 2000a).

Figures 11 to 14 illustrate the impact of each of the priority assignment policies on the percentage of tasksets deemed schedulable by the different schedulability tests for 2, 4, 8, and 16 processors respectively.

In the two processors case (Figure 12) DMPO, D-CMPO and DkC priority assignment policies have similar performance[8], and it is the effectiveness of the schedulability tests which dominates the results obtained for these policies. Using optimal priority assignment, the DA-LC test admits slightly fewer tasksets as compared to the RTA-LC test using DkC priority assignment.

As the number of processors is increased, from 2 up to 16 (in Figure 15) the difference between priority assignment policies dominates the results. For 16 processors, there is a large difference between the utilisation level at which. 50% of the tasksets are deemed schedulable according to the DA-LC test using DMPO (approx. 9.2 = 0.58m), versus using the optimal priority assignment algorithm (approx. 12 = 0.75m). This difference corresponds to an effective increase in usable processing capacity of around 30%.

It is interesting to compare the graphs for implicit-deadline tasksets (Figures 11 to 14) with their counterparts for constrained-deadline tasksets (Figures 7 to 10). We conclude from this comparison, that the selection of OPA or DkC priority ordering rather than D-CMPO or DMPO brings about a greater improvement in the number of tasksets deemed schedulable in the constrained deadline case. Further the improvement obtained by using optimal priority assignment rather than the DkC heuristic is more

[8] DkC and D-CMPO are in fact identical as k=1 for two processors.

pronounced in the implicit deadline case.

Finally, as the assumption that all higher priority tasks have worst-case response times equal to their execution times becoming more optimistic for tasks with longer deadlines and tasksets with higher utilisation, we hypothesize that the upper bound on the performance of the RTA-LC test with optimal priority assignment, given by the C-RTA condition, becomes more optimistic in the implicit-deadline case.

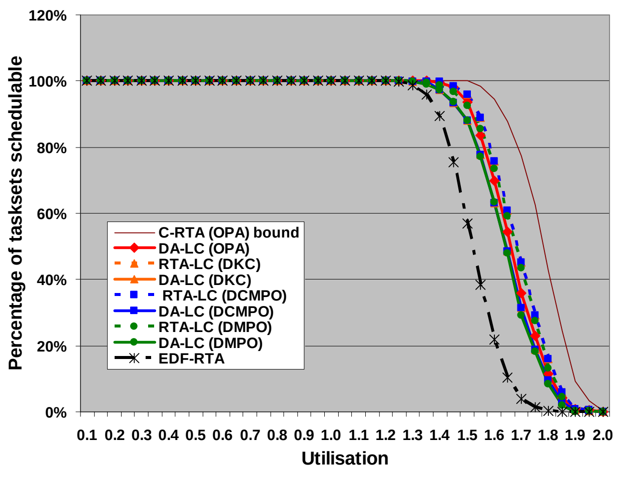

Figure 12: (2 processors, 10 tasks)

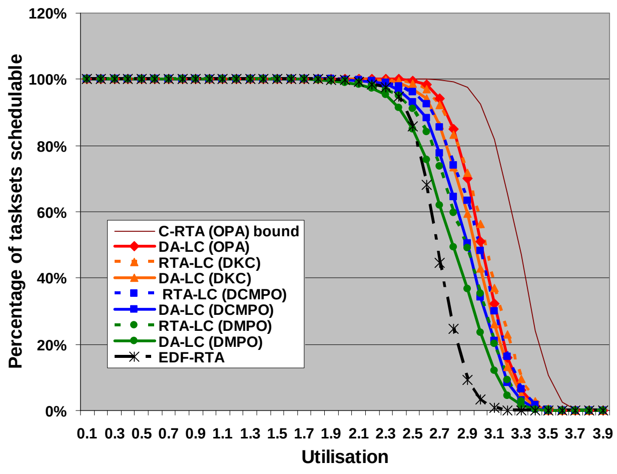

Figure 13: (4 processors, 20 tasks)

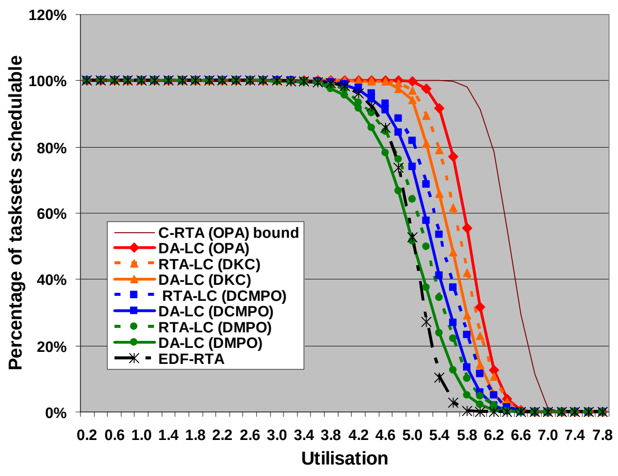

Figure 14: (8 processors, 40 tasks)

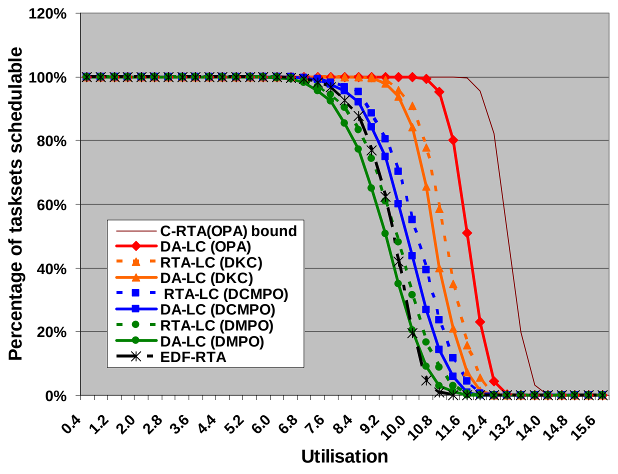

Figure 15: (16 processors, 80 tasks)

We repeated our experiments for smaller (2) and larger (20) numbers of tasks per processor. In each case, although the data points changed, the relationships between the effectiveness of the different methods and the conclusions that can be drawn from them remained essentially the same.

We note that overall, (2, 4, 8, and 16 processors, implicit and constrained deadline tasksets) the method with the best performance is the polynomial DA-LC test combined with optimal priority assignment. This confirms our hypothesis that finding an appropriate priority ordering is as important as using an effective schedulability test. Further, this test also significantly outperforms the EDF-RTA test for global EDF scheduling in all of the cases studied.

6.3. Experiment 2 (Number of tasks)

In this experiment we investigated the effect of varying the number of tasks. Figure 16 shows the percentage of tasksets that were schedulable on an 8 processor system, for taskset cardinalities of 9, 10, 12, 16, 24, and 40, using the DA-LC test with optimal priority assignment. Figure 17 shows similar data for tasksets of cardinality 40, 80, 120, 160, and 200. We repeated this experiment for tasksets with implicit deadlines, and also for the RTA-LC test using DkC priority assignment with similar results.

There are some data points missing from the right hand side of Figure 16. This is because the UUnifast-Discard algorithm, was unable to generate tasksets with cardinality 9 and utilisation greater than 6.6 (or cardinality 10 and utilisation greater than 6.8) using a discard limit of 1000; however, despite this the trends are still clearly visible.

In Figure 16, the percentage of schedulable tasksets decreases as the number of tasks is increased from 9 towards 40, with all other parameters held constant. It would appear from this data alone that tasksets with a larger number of tasks are more difficult to schedule. Figure 17 shows what happens as we continue to increase the number of tasks from 40 to 200 (25 times the number of processors). Now as the number of tasks increases, the tasksets appear to become easier to schedule. This behaviour can be explained as a combination of two effects: With a small number of tasks, tasksets are relatively easy to schedule as the impact of each high utilisation, high interference task is limited to effectively occupying one processor (see Equations (3) and (7)). In the extreme, any valid taskset with m tasks or less is trivially schedulable on an m processor system. As taskset cardinality increases from m to $2m$ we therefore expect fewer tasksets to be schedulable at any given utilisation. At the other extreme, with increasing taskset cardinality ($n >> m$), the average density (C_k / D_k) of each task τ_k becomes small. This means that the amount of pessimism in the schedulability tests, due to the assumption that when τ_k executes all other processors are idle is reduced. Hence, as n increases beyond $10m$ so the number of schedulable tasksets increases.

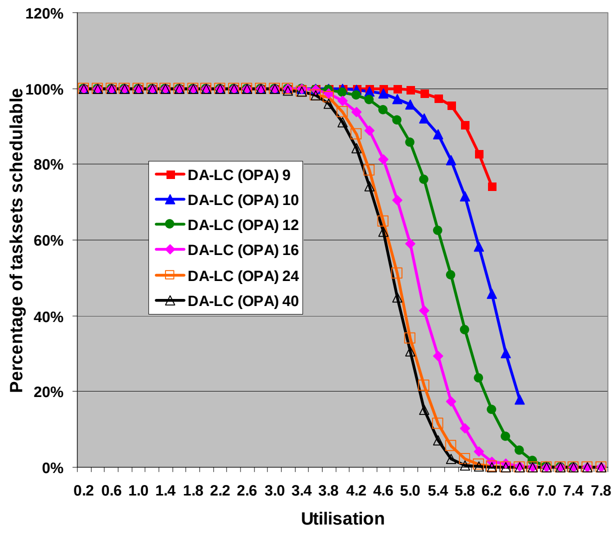

Figure 16: (taskset cardinality from 9 to 40)

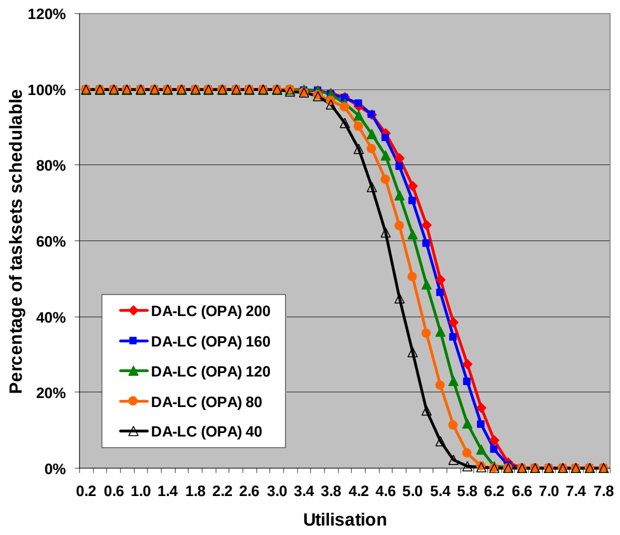

Figure 17: (taskset cardinality from 40 to 200)

The fact that on an m processor system, any valid set of m tasks is schedulable, illustrates the incomparability of global FP scheduling on m processors of speed 1, with respect to fixed priority pre-emptive scheduling on a similar uniprocessor of speed m. The m-speed uniprocessor can trivially schedule a single task of utilisation greater than one, whereas the m processors of speed 1 cannot. Similarly, the m processors can schedule any set of m tasks with co-prime periods and individual task utilisations equal to 1, whereas the m-speed uniprocessor cannot.

7. Summary and conclusions

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

In this paper we addressed the problem of priority assignment for global FP scheduling of constrained-deadline sporadic tasksets. We were drawn to this area of research by the work of Bertogna et al. (2009) which showed that the best schedulability tests then available for global FP scheduling using Deadline Monotonic Priority Ordering (DMPO) outperformed the best tests then known for both global EDF and EDZL.

The intuition behind our work was the idea that in fixed priority scheduling, finding an appropriate priority assignment is as important as using an effective schedulability test. While DMPO is an optimal priority assignment policy for uniprocessors, this result is known not to transfer to the multiprocessor case. Indeed, our results show that DMPO cannot even be considered a good heuristic for multiprocessors.

The key contributions of this paper are as follows:

o Formal proof of a key observation concerning the pattern of task execution that results in the worst-case response time under global fixed priority scheduling (Theorem 1).

- Application of the approach of Guan et al. (2009) to limiting carry-in interference to the polynomial time schedulability test of Bertogna et al. (2009), forming the DA-LC test.
- The observation that although Audsley's Optimal Priority Assignment algorithm (Audsely, 1991, 2001) cannot be applied to any exact schedulability test for global FP scheduling of periodic tasksets, this does not necessarily preclude its use with sufficient schedulability tests.
- Proof that Audsley's OPA algorithm is the optimal priority assignment policy with respect to any global FP schedulability test for periodic or sporadic tasksets that complies with three simple conditions.
- Classification of schedulability tests for global FP scheduling as either OPA-compatible or OPA-incompatible based on these conditions. The deadline-based sufficient test of Bertogna et al. (2009) (DA test), the DA-LC test developed in this paper, and the response time test of Andersson and Jonsson (2000a) for sporadic tasksets are all OPA-compatible; while any exact test for periodic tasksets, the response time test of Bertogna and Cirinei (2007) (RTA test) and the improved version of this test given by Guan et al. (2009) (RTA-LC test) for sporadic tasksets are OPA-incompatible.
- Extension of the TkC (Andersson and Jonsson, 2000b) priority assignment policy to constrained deadline tasksets forming the DkC priority assignment policy. This heuristic policy can be used in conjunction with any schedulability test.
- Adaptation of the UUnifast algorithm (Bini and Buttazzo, 2005) to the multiprocessor case, forming the UUnifast-Discard algorithm. UUnifast-Discard generates tasksets with specific parameter settings, facilitating an empirical study of schedulability test effectiveness without the problem of confounding variables.
- An empirical study showing that by using the OPA algorithm rather than DMPO, the DA-LC test can schedule significantly more tasksets. In fact, this combination of optimal priority assignment and a polynomial time schedulability test outperformed the pseudo-polynomial RTA-LC test combined with various heuristic priority assignment policies, including DMPO, D-CMPO, and DkC. It also significantly outperformed the EDF-RTA test for global EDF scheduling.
- Deriving a pseudo-schedulability condition (C-RTA) which dominates the RTA-LC test yet is OPA-compatible. This condition combined with the OPA algorithm provides an upper bound on the potential performance of the RTA-LC test with optimal priority assignment. The gap between the DA-LC test with optimal priority assignment and this upper bound was found to be relatively small for constrained-deadline tasksets, with a larger gap (possibly due to optimism in the bound) for implicit-deadline tasksets.

Our studies showed that the improvements that an appropriate choice of priority assignment brings are very large when viewed in terms of the proportion of processing capacity that can be usefully deployed. For example, in the 16 processor case, for tasksets with constrained deadlines, the utilisation level at which 50% of the tasksets were schedulable increased from 0.28*m* (for the DA-LC test with DMPO) to 0.6*m* (for the DA-LC test with optimal priority assignment). This represents an effective increase in the usable processing resource of over 100%. This level of improvement is of great value to engineers designing and implementing hard real-time systems based on multiprocessor platforms, as it enables more effective use to be made of processing resources while still ensuring that deadlines are met. We conclude that priority assignment is an important factor in determining the schedulability of tasksets under global fixed priority pre-emptive scheduling.

The OPA algorithm requires a polynomial number of schedulability tests ($n(n+1)/2$) to solve the problem of optimal priority assignment for any OPA-compatible global FP schedulability test. To the best of our knowledge, the complexity of optimal priority assignment for exact schedulability tests for periodic tasksets under global FP scheduling remains an open problem. For sporadic tasksets, no exact test is known and the complexity of optimal priority assignment is also an open problem.

The research reported in this paper suggests that the most effective combination of schedulability test and priority assignment policy currently available for global fixed priority scheduling is the DA-LC test (Equation (14)) introduced in this paper, combined with the optimal priority assignment (OPA) algorithm. As well as being highly effective, this approach has the additional advantage that it is polynomial $O(n^3)$ in complexity and therefore highly efficient.

Finally, the upper bound provided by the C-RTA condition indicates that there remains some scope to improve upon these results if a way can be found to combine optimal priority assignment with the state-of-the-art response time test (RTA-LC test). Initial work in this area can be found in (Davis and Burns, 2010).

7.1. Acknowledgements

The authors would like to thank Enrico Bini and Paul Emberson for their contributions to the discussions about the applicability of the UUnifast algorithm to the multiprocessor case, and also Yang Chang for his insightful review of an early draft. This work was funded in part by the EPSRC Tempo project (EP/G055548/1) and

the EU funded ArtistDesign Network of Excellence.

Appendix A: Other heuristic priority assignment policies

In this section, we examine the performance of two heuristic priority assignment algorithms, derived from RM-US{ς} (Andersson et al., 2001) and SM-US{ς} (Andersson, 2008).

A.1 Implicit-deadline tasksets

The RM-US{ς} priority assignment policy, was derived by Andersson et al. (2001) with the aim of addressing the "Dhall effect" (Dhall and Liu, 1978) for implicit-deadline tasksets using global FP scheduling. RM-US{ς} assigns the highest priority[9] to tasks with utilisation greater than some threshold ς. The remaining tasks are then assigned priorities in Rate Monotonic priority order.

Lundberg (2002) showed that setting the threshold used in RM-US{ς} to 0.375 results in the following utilisation bound which is the maximum possible bound for this class of algorithm:

$$U \le 0.375m \tag{A.1}$$

The SM-US{ς} priority assignment policy was derived by Andersson (2008) with the aim of improving upon the above bound for RM-US{ς}. SM-US{ς} again assigns the highest priority to tasks with utilisation greater than some threshold ς; the remaining tasks are then assigned priorities in Slack Monotonic priority order, (where the slack of task τ_k is defined as $D_k - C_k$). Andersson (2008) showed that using a threshold of $2/(3+\sqrt{5})$ results in the following utilisation bound for SM-US{ς}:

$$U \le 2/(3+\sqrt{5}) \tag{A.2}$$

Figure 18 illustrates the impact of the RM-US{ς} and SM-US{ς} priority assignment policies, on the percentage of implicit-deadline tasksets deemed schedulable by the DA-LC schedulability test. This data is for tasksets generated according to the parameters described in Section 6.1, with the exception that all task deadlines were equal to their periods. The thresholds used were 0.375 for RM-US{ς} and $2/(3+\sqrt{5})$ for SM-US{ς}.

[9] Note that RM-US considers fewer than *m* tasks assigned priorities based on their utilisation, and as the first *m* priority levels in an *m* processor system are essentially equivalent, makes no distinction between their priorities.

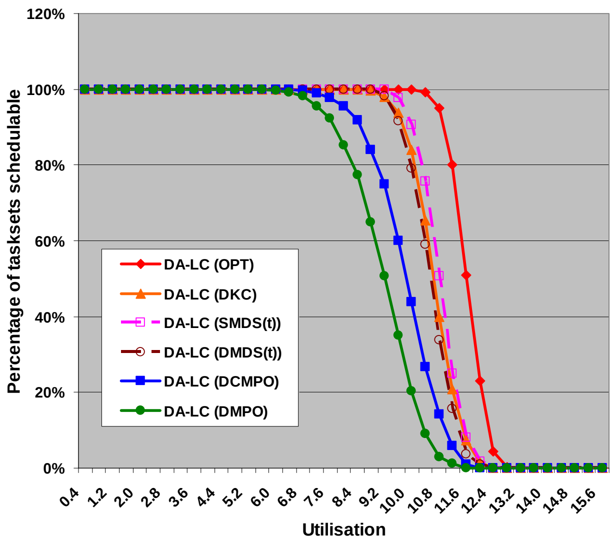

Figure 18: (16 processors, 80 tasks)

From Figure 18, we observe that the performance of the RM-US{0.375} and SM-US{$2/(3+\sqrt{5})$} priority assignment policies is very similar to that of DkC (i.e. TkC) for implicit-deadline tasksets. This similarity in performance was also observed in the cases of 2, 4, and 8 processors.

A.2 Constrained-deadline tasksets

Bertogna et al. (2005) extended the RM-US{ς} priority assignment policy to constrained-deadline tasksets, forming the DM-DS{ς} policy. DM-DS{ς} assigns the highest priorities to at most *m*-1 tasks with densities ($\delta_k = C_k / D_k$) greater than some threshold ς. Bertogna et al. showed that using a threshold of 1/3 results in the following density bound for global FP scheduling of constrained-deadline tasksets using DM-DS{1/3} priority assignment:

$$\sum_{\forall k} \delta_k \le \frac{m+1}{3} \tag{A.3}$$

The SM-US{ς} priority assignment policy can also be extended to the constrained deadline case, by simply assigning the highest priority to those tasks with density (rather than utilisation) greater than some threshold ς. We refer to this policy as SM-DS{ς}

Figure 19 below shows the results of essentially the same experiment as Figure 18; however, this time using constrained-deadline tasksets, with task deadlines chosen according to a uniform random distribution, in the range $[C_i, T_i]$. Here, we see that the performance of the DM-DS{1/3} and SM-DS{$2/(3+\sqrt{5})$} priority assignment policies is significantly worse than that of DkC. Further, we found that the relative performance of DM-DS{1/3} and SM-DS{$2/(3+\sqrt{5})$} was variable, depending on the number of processors. In the case of two processors, the performance of both DM-DS{1/3} and SM-DS{$2/(3+\sqrt{5})$} was significantly worse than that of

DMPO.

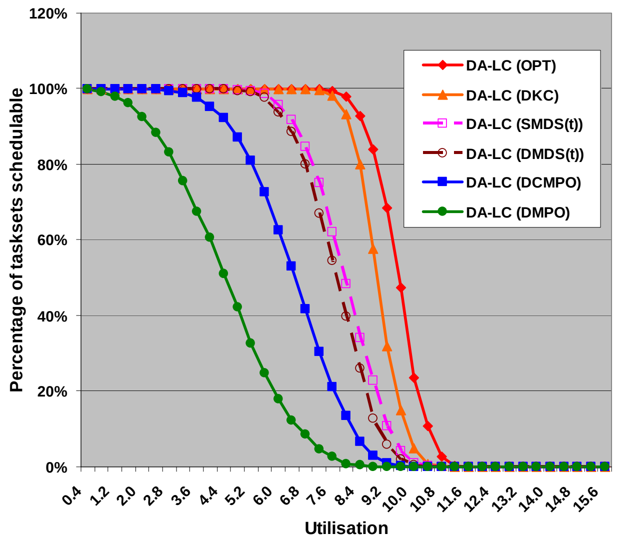

Figure 19: (16 processors, 80 tasks)

A possible explanation for the variable and relatively poor performance of DM-DS$\{1/3\}$ and SM-DS$\{2/(3+\sqrt{5})\}$ is the choice of threshold. While Bertogna et al. (2005) and Andersson (2008) were able to derive appropriate thresholds for DM-DS$\{\varsigma\}$ and SM-US$\{\varsigma\}$ in order to derive maximum density or utilisation bounds, it is not obvious what the thresholds should be for constrained-deadline tasksets which exceed these bounds.

We now describe variants of the DM-DS$\{\varsigma\}$ and SM-US$\{\varsigma\}$ priority assignment policies, which address the problem of selecting an appropriate threshold. Here, we employ an idea used by Goosens et al. (2003) and Baker (2005) in global EDF scheduling. We refer to these algorithms as DM-DS(h) and SM-DS(h).

The DM-DS(h) and SM-DS(h) priority assignment algorithms both assign the highest h priorities based on task density, highest density first. The remaining tasks are then assigned priorities in either Deadline Monotonic (DM-DS(h)) or Slack Monotonic (SM-DS(h)) priority order. Instead of using a threshold, the DM-DS(h) and SM-DS(h) algorithms, simply try all values of h, from zero, (which is equivalent to DMPO or Slack Monotonic priority order), to n-1, (which is equivalent to ordering all of the tasks based on decreasing density). Thus for a taskset of cardinality n, applying either the DM-DS(h) or the SM-DS(h) priority assignment algorithm implies checking taskset schedulability for n different priority orderings, (corresponding to h = 0 to n-1), stopping only when a schedulable priority ordering is found, or when all n priority orderings are found to be unschedulable.

The DM-DS(h) and SM-DS(h) priority assignment algorithms circumvent the problem of finding an appropriate threshold, by effectively examining all of the priority orderings that could possibly be generated by any arbitrary threshold value.

Figure 20 illustrates the impact of the DM-DS(h) and SM-DS(h) priority assignment algorithms on the percentage of tasksets deemed schedulable by the DA-LC schedulability test. This data is again for constrained-deadline tasksets and so is directly comparable with that presented in Figure 19.

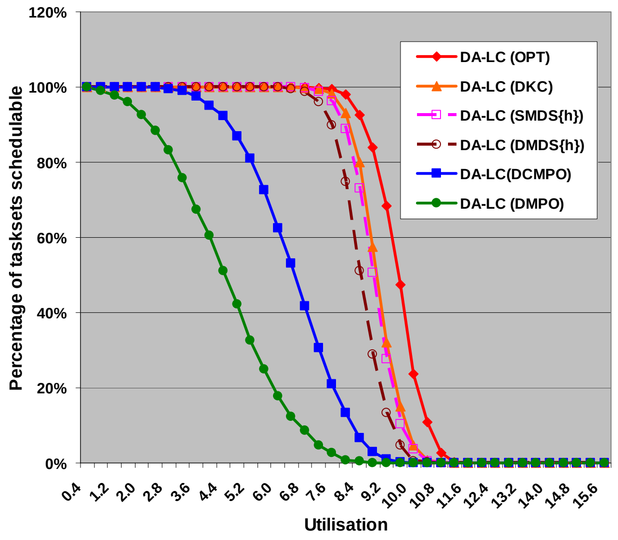

Figure 20: (16 processors, 80 tasks)

From Figure 20, it is clear that the performance of the SM-DS(h) priority assignment algorithm is similar to that of DkC, with DM-DS(h) providing somewhat inferior performance. The results shown in Figure 20 are for a 16 processor system, with 80 tasks. We also repeated this experiment for smaller (2) and larger (20) numbers of tasks per processor, and 2, 4, and 8 processors. In each case, although the data points changed, the relationships between the different priority assignment methods remained essentially the same.

While the SM-DS(h) priority assignment algorithm gives very similar performance to that of DkC, SM-DS(h) is more complex, requiring a schedulability test to be performed for n different priority orderings rather than just one. For this reason we recommend using DkC priority assignment in conjunction with OPA-incompatible schedulability tests for global FP scheduling. For OPA-compatible schedulability tests, then Audsley's optimal priority assignment algorithm should be used.

References

Andersson B, Jonsson J (2000a) Some insights on fixed-priority pre-emptive non-partitioned multiprocessor scheduling. In Proc. RTSS – Work-in-Progress Session.

Andersson B, Jonsson J (2000b) Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition, In Proc. RTCSA.

Andersson B, Baruah SK, Jonsson J. (2001) Static-priority scheduling on multiprocessors. In Proc. RTSS, pp. 193–202.

Andersson B, Jonsson J (2003) The Utilization Bounds of Partitioned and Pfair Static-Priority Scheduling on Multiprocessors are 50%, In Proc. ECRTS, pp. 33-40.

Andersson B (2008) Global static-priority preemptive multiprocessor scheduling with utilization bound 38%. In Proc. International Conference on Principles of Distributed Systems, pp. 73-88

Audsley NC (1991) Optimal priority assignment and feasibility of static priority tasks with arbitrary start times, Technical Report YCS 164, Dept. Computer Science, University of York, UK.

Audsley NC (2001) On priority assignment in fixed priority scheduling, Information Processing Letters, 79(1): 39-44.

Baker TP (2003) Multiprocessor EDF and deadline monotonic schedulability analysis. In Proc. RTSS, pp. 120–129.

Baker TP (2005) An analysis of EDF scheduling on a multiprocessor. IEEE Trans. on Parallel and Distributed Systems, 15(8):760–768.

Baker TP (2006) An analysis of fixed-priority scheduling on a multiprocessor. Real Time Systems, 32(1-2), 49-71.

Baker TP, Cirinei M, Bertogna M (2008) EDZL scheduling analysis. Real-Time Systems. 40(3) : 264-289.

Baker TP, Baruah SK. (2009) Sustainable multiprocessor scheduling of sporadic task systems. In Proc. ECRTS, pp. 141-150.

Baruah SK (2007) Techniques for Multiprocessor Global Schedulability Analysis. In proc. RTSS, pp. 119-128.

Baruah SK, Fisher N (2008) Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems. In Proc. of the 9th Int'l Conference on Distributed Computing and Networking, pp. 215-226.

S Baruah SK, Bonifaci V, Marchetti-Spaccamela A, Stiller S (2009) Implementation of a speedup-optimal global EDF schedulability test, In Proc. ECRTS, pp. 259-268.

Baruah SK, Baker TP (2009) An analysis of global EDF schedulability for arbitrary sporadic task systems. Real-Time Systems ECRTS special issue, 43(1): 3-24.

Baruah SK (2009) Schedulability analysis of global deadline monotonic scheduling. Technical report available from http://www.cs.unc.edu/~baruah/Pubs.shtml.

Bertogna M, Cirinei M, Lipari G (2005) 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.

Bertogna M, Cirinei M (2007) Response Time Analysis for global scheduled symmetric multiprocessor platforms. In Proc. RTSS, pp. 149-158.

Bertogna M (2007) Real-Time Scheduling for Multiprocessor Platforms. PhD Thesis, Scuola Superiore Sant'Anna, Pisa.

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

Bertogna M (2009) Evaluation of existing schedulability tests for global EDF, In proceedings of the First International Workshop on Real-time Systems on Multicore Platforms: Theory and Practice.

Bini E, Buttazzo GC (2005) Measuring the Performance of Schedulability tests. Real-Time Systems, 30(1–2):129–154.

Cirinei M, Baker TP (2007) "EDZL scheduling analysis". In Proc. ECRTS, pp. 9–18.

Cucu L, Goossens J (2006) Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors, In Proc. 11th IEEE International Conference on Emerging Technologies and Factory Automation.

Cucu L, Goossens J (2007) Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary Deadline Periodic Systems , In Proc. DATE, pp. 1635-1640.

Cucu L (2008) Optimal priority assignment for periodic tasks on unrelated processors. In Proc. ECRTS WiP session.

Davis RI, Burns A (2007) Robust Priority Assignment for Fixed Priority Real-Time Systems. In Proc. RTSS, pp. 3-14.

Davis RI, Burns A (2009a) Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems. In Proc. RTSS, pp. 398-409.

Davis RI, Burns A (2009b) A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems, Technical Report YCS-2009-443, Dept. of Computer Science, University of York (*to appear in ACM Computing Surveys*).

Davis RI, Burns A (2010) On Optimal Priority Assignment for Response Time Analysis of Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Hard Real-Time Systems. Department of Computer Science, University of York, Technical Report YCS-2010-451.

Dhall SK, Liu CL (1978) On a Real-Time Scheduling Problem, Operations Research, vol. 26, No. 1, pp. 127-140.

Easwaran A, Shin I, Lee I (2008) Toward Optimal Mutiprocessor Scheduling for Arbitrary Deadline Tasks. In Proc. RTSS WiP pp. 1-4.

Fisher N, Baruah SK (2006) Global Static-Priority Scheduling of Sporadic Task Systems on Multiprocessor Platforms. In Proc. IASTED International Conference on Parallel and Distributed Computing and Systems.

Goossens J, Funk S, Baruah SK, (2003) Priority-driven scheduling of periodic task systems on multiprocessors. Real Time Systems, 25(2–3):187–205.

Guan N, Stigge M, Yi W, Yu G (2009) New Response Time Bounds for Fixed Priority Multiprocessor Scheduling. In proceedings of the Real-Time Systems Symposium, pp. 388-397.

Lauzac S, Melhem R, Mosse D (1998) Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In Proc. of the EuroMicro Workshop on Real-Time Systems, pp. 188–195.

Liu CL (1969) Scheduling algorithms for multiprocessors in a hard real-time environment. JPL Space Programs Summary, vol. 37-60, pp. 28-31.

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM, 20(1): 46-61.

Leteinturier P (2007) Multi-Core Processors: Driving the Evolution of Automotive Electronics Architectures. In embedded.com 16/09/2007.

Leung JY-T, Whitehead J (1982) On the complexity of fixed-priority scheduling of periodic real-time tasks. Performance Evaluation, 2(4): 237-250.

Lundberg L (2002) Analyzing Fixed-Priority Global Multiprocessor Scheduling. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium.

Rosenburg B (2009) Product Focus: Software. In Avionics Magazine, 1st Oct. 2009.

Serlin O (1972) Scheduling of time critical processes. In proceedings AFIPS Spring Computing Conference, pp 925-932.