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

In this paper, 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.

1. Introduction

Today 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.

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. In 1972, Serlin [32] and Liu and Layland [30] 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). In 1982, Leung and Whitehead [31] showed that Deadline Monotonic priority ordering (DMPO) is optimal for independent synchronous tasks with *constrained deadlines* (less than or equal to their periods). In 1991, Audsley [6], [7] 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". In 1978, Dhall and Liu [25] 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. In 2000, Andersson and Jonsson [2] 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.

In 2001, Anderson et al. [3] gave a utilisation bound for global FP scheduling of periodic tasksets with implicit deadlines using the RM-US$\{\varsigma\}$ priority assignment policy. RM-US$\{\varsigma\}$ gives the highest priority to tasks with

1052-8725/09 $26.00 © 2009 IEEE
DOI 10.1109/RTSS.2009.31

utilisation greater than a threshold ς. In 2003, Andersson and Jonsson [4] 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. In 2005, Bertogna [15] extended the work of Andersson et al. [3] to sporadic tasksets with constrained deadlines forming the DM-DS$\{\varsigma\}$ priority assignment policy. DM-DS$\{\varsigma\}$ gives the highest priority to at most $m-1$ tasks with densities greater than the threshold ς, and otherwise uses DMPO. Bertogna [15] provided a density-based schedulability test for DM-DS$\{\varsigma\}$. In 2008, Andersson [5] proposed a form of Slack Monotonic priority assignment called SM-US$\{\varsigma\}$ Using a threshold of $2/(3+\sqrt{5})$, SM-US$\{\varsigma\}$ 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. In 2000, Andersson and Jonsson [1] gave a simple response time upper bound applicable to tasksets with constrained-deadlines. In 2001, Baker [8] 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 [11], [17], [14], [13], and global FP scheduling [12], [18], [9], [27].

Baker's work was subsequently built upon by Bertogna et al. [15] in 2005, and Bertogna and Cirinei [18] in 2009. They developed sufficient schedulability tests for: (i) any work conserving algorithm, (ii) global EDF, and (iii) global FP, 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. In 2007, Bertogna and Cirinei [16] 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.

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 [29]. This restriction manifests itself as the critical instant effect [28], where simultaneous release of 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 [20], [21].

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 [17] showed that for sporadic tasksets with constrained deadlines, the response time test [16] for global FP scheduling – using DMPO, outperformed all other tests known at the time, including those for global FP, global EDF, and EDZL [10]; 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 [31], this optimality does not extend to multiprocessors.

In this paper, we prove that Audsley's Optimal Priority Assignment (OPA) algorithm [6], [7], 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 simple 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, OPA-incompatible schedulability test with DMPO. Further, we build on the work of Andersson and Jonsson [2], 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 recapitulates existing 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 with a summary and suggestions for future research.

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 ($\tau_1 ... \tau_n$), where each task τ_i is assigned a unique priority i, from 1 to n (where n is the lowest priority).

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 the task arriving to it completing execution.

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

Intra-task parallelism is not permitted; hence, at any given time, each job may execute on at most one processor. As a result of pre-emption and subsequent resumption, a job may migrate from one processor to another. The cost of pre-emption, migration, and the run-time operation of the scheduler is assumed to be either negligible, or subsumed into the worst-case execution time of each task.

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 tasksets / priority ordering combinations that it deems schedulable are in fact schedulable. Similarly, test S is said to be *necessary* if all of the tasksets / priority ordering combinations 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 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 the following holds: P is optimal 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 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 [20], [21] 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 [22]. 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 test S we cannot classify all global FP feasible tasksets, due to the sufficiency of the test. 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 the sufficiency of the test, Y is a strict subset of the set G containing all global FP feasible tasksets ($G \supset Y \supseteq Z$).

3. Recapitulation of schedulability tests

In this section, we outline two sufficient schedulability tests for global fixed priority scheduling of sporadic tasksets developed by Bertogna et al [18], and Bertogna and Cirinei [16]. Both of these tests are based on the fundamental strategy derived by Baker [8], 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.
2. Establish a condition *necessary* for the job to miss its deadline, for example, all m processors executing

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

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.

In [18], Bertogna et al. derived a sufficient schedulability 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 1.

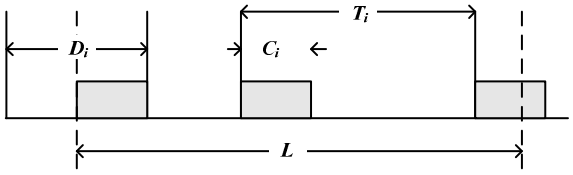

Figure 1

Bertogna et al. [18] showed that $W_i^D(L)$ is an upper bound on the workload of task τ_i in an interval of length L:

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

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

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

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:

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

Note, the '+1' term in Equation (3) is a result of the approach to time representation[2] used in [18].

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

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

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 in [18] for ease of comparison with the schedulability test given in [16].

Bertogna and Cirinei [16] 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 [18], 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 [16] 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[3].

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:

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

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 [16]).

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

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.

For convenience, in the rest of this paper, we refer to the sufficient schedulability test based on deadline analysis, given by Equation (4), as the "DA test", and that based on response time analysis, given by Equation (8), as the "RTA test".

4. Priority assignment

In 2000, Andersson and Jonsson [1] 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 concluded that even if an exact schedulability test were known[4], then it would not be possible to use Audsley's OPA algorithm [6], [7] to determine the optimal priority ordering. While this is undoubtedly true, we believe that it has also lead to a

[2] 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.

[3] Bertogna and Cirinei [16] also investigated global EDF scheduling and the slack terms are necessary in that case.

[4] Note, such a test is now known, see [20], [21].

common misconception that the OPA algorithm cannot be applied to schedulability tests for global FP scheduling.

In this section, we explore the problem of optimal priority assignment for global FP scheduling. First we provide an overview of Audsley's OPA algorithm, [6], [7] derived for uniprocessor systems.

4.1. Optimal priority assignment

The pseudo code for the OPA algorithm, using some 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
        {
            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 orderings. Note that the OPA algorithm does not specify the order in which tasks should be tried at each priority level.

Let *S* be some schedulability test for global FP scheduling which complies with the following conditions:

Condition 1: The schedulability of a task τ_k may, according to test *S*, be dependent on the set of higher priority tasks, but not on the relative priority ordering of those tasks.

Condition 2: The schedulability of a task τ_k may, according to test *S*, be dependent on the set of lower priority tasks, but not on the relative priority ordering of those tasks.

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 1: 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 assume for contradiction that there exists a taskset *X* that is schedulable according to test *S* with priority ordering Q, and further that the OPA algorithm is unable to generate a schedulable priority ordering for taskset *X*.

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 find a priority ordering *P* for taskset *X* that is schedulable according to test *S*. This contradicts the assumption and hence proves the theorem.

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 $Q_{n-1} \ldots Q_0$, where Q_0 is equivalent to *P*, the priority ordering generated by the OPA algorithm. The transformation will be such that after each iteration *k*, (from *n* 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} and *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, and also the index for the transformed priority ordering.

Proof 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* ($lp(k,Q_k) = lp(k,P)$), 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 2.

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 affected 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 □

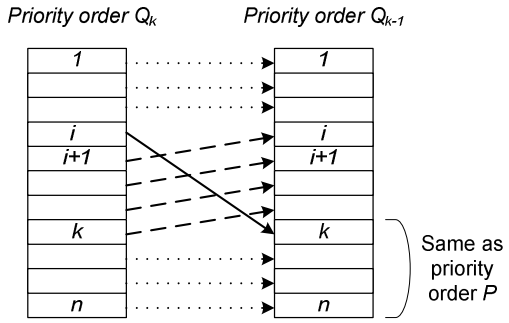

Figure 2

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

Theorem 2: Conditions 1-3 are all necessary conditions for the OPA algorithm to correctly identify a priority ordering that is deemed schedulable by schedulability test S if 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, the 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 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.

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 1 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 3: Any exact schedulability test for periodic tasksets 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 tasksets / priority ordering combinations as schedulable or unschedulable it follows that all exact schedulability tests for periodic tasksets are OPA-incompatible □

Theorem 4: The RTA test [16] for global FP scheduling of sporadic tasksets (Equation (8)) 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 the RTA test 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 RTA 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 5: The response time test of Andersson and Jonsson [1] (Equation (9) below) for global FP scheduling of sporadic tasksets is OPA-compatible:

$$R_k^{ub} \leftarrow C_k + \frac{1}{m} \sum_{\forall i \in hp(k)} \left(\left\lceil \frac{R_k^{ub}}{T_i} \right\rceil C_i + C_i \right) \tag{9}$$

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

Inspection of Equation (9) 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 (9)) 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 (9)) is the removal of task A from the set of tasks that have higher priority than task B, hence Condition 3 holds □

Theorem 6: The DA test [18] (Equation (4)) for global FP scheduling of sporadic tasksets is OPA-compatible:

Proof: Follows the same logic as the proof of Theorem 5, with the upper bound response time given by the right hand side of Equation (4) rather than Equation (9) □

4.2. Heuristic priority assignment

In this section, we investigate heuristic priority assignment policies.

In his thesis [17], Bertogna evaluates the effectiveness of a number of different schedulability tests. Bertogna's experiments show that using DMPO the RTA test 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):

$$D_k - C_k \geq \left\lfloor \frac{1}{m} \sum_{\forall i \in hp(k)} I_i^D(D_k) \right\rfloor \tag{10}$$

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 [2] 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:

$$k = \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{2m} \tag{11}$$

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 (11).

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{ς} [15] and SM-US{ς} [5] priority assignment policies. These algorithms, although more complex, were found to be no more effective than the DkC policy. Details of the algorithms and their performance can be found in [24].

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. [26] in 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 [19], 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 [19] (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-1));
     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 [17], [18], [16], [10]. 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 tasksets.

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 7: The tasksets produced by UUnifast-Discard are unbiased, i.e. uniformly distributed [19], 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 [19] 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 tasksets 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 3 shows the maximum taskset utilisation that UUnifast-Discard is able to generate, using a discard limit of 1000, plotted against taskset cardinality. For example, UUnifast-Discard 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).

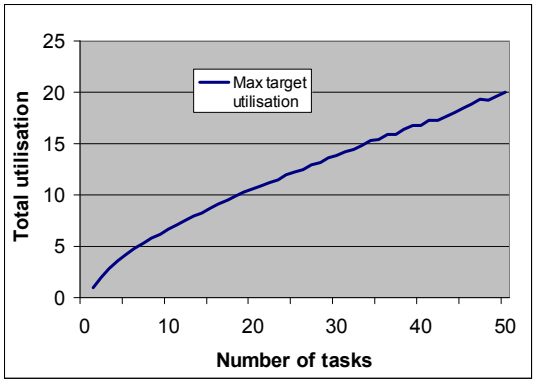

Figure 3

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 test" (Equation (4)), which is OPA-compatible, and the "RTA test" (Equation (8)), which is OPA-incompatible. The priority assignment policies studied are DMPO, D-CMPO, DkC and the OPA algorithm (DA test only).

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[5] with a factor of 1000 difference between the minimum and maximum possible task period. This represents a spread of task periods from 1ms to 1000ms, 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 two schedulability tests. Figures 4 to 7 show this data for 2, 4, 8, and 16 processors respectively.

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 4 solid lines on each graph depicting the performance of the DA test for DMPO (lowest performance), D-CMPO, DkC, and OPA (highest performance / optimal with respect to this schedulability test). Note the lines on the graphs appear in the order given in the legend.

In the 16 processor case (Figure 7), using DMPO, approx. 50% of the tasksets are unschedulable according to the DA test at a utilisation level of 4.4 (= 0.28m); however, using the OPA algorithm, approx. 50% of the tasksets are schedulable according to the same test at a utilisation level of 9.4 (= 0.59m). Hence, in this case, optimal priority assignment effectively enables 114% better utilisation of the processing resource than DMPO. D-CMPO is more effective than DMPO, and the DkC priority assignment policy is notably almost as effective as optimal priority assignment. Note, the performance of DkC and D-CMPO are identical in the 2 processor case (Figure 4) as k = 1 in Equation (11). Comparison between the four figures shows that the difference between OPA and DMPO becomes larger as the number of processors increases.

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

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

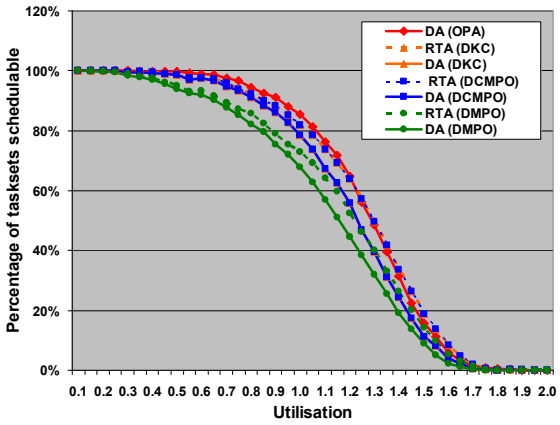

Figure 4: (2 processors, 10 tasks)

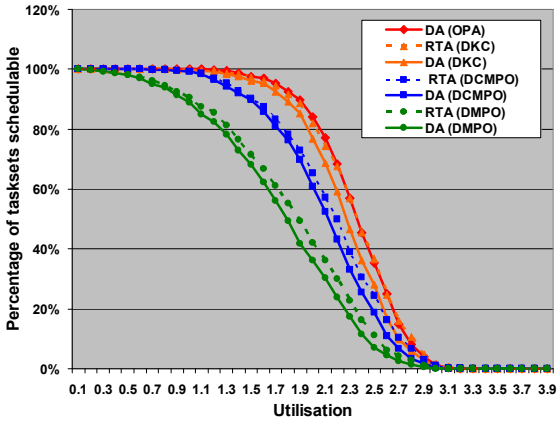

Figure 5: (4 processors, 20 tasks)

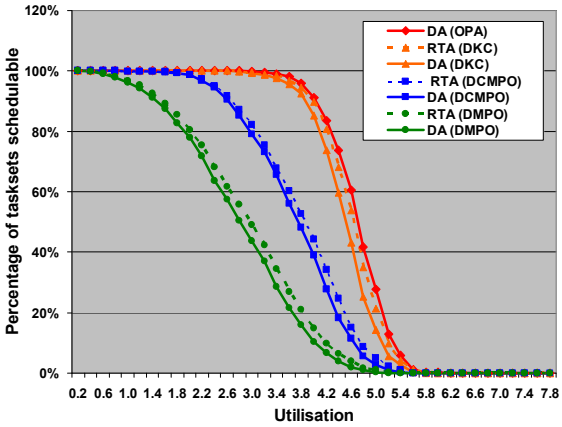

Figure 6: (8 processors, 40 tasks)

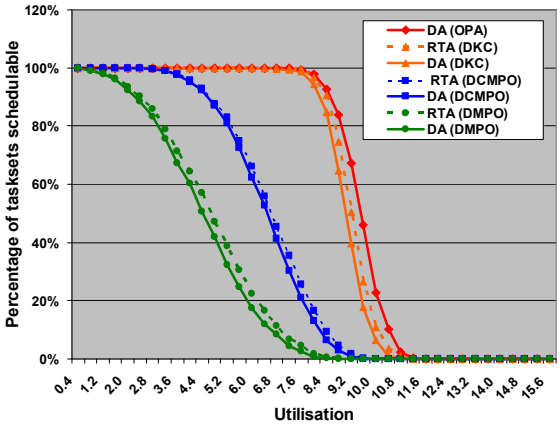

Figure 7: (16 processors, 80 tasks)

The data shown in Figures 4 to 7 is for systems with 5 times as many tasks as processors. We repeated these 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.

6.3. Experiment 2 (Number of tasks)

In this experiment we investigated the effect of varying the number of tasks. Figure 8 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 test with optimal priority assignment (solid lines). Data for the RTA test with DkC priority assignment was almost identical (not shown on the graph). Figure 9 shows similar data for tasksets of cardinality 40, 80, 120, 160, and 200.

There are some data points missing from the right hand side of Figure 8. 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 8, 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 9 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.

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 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.

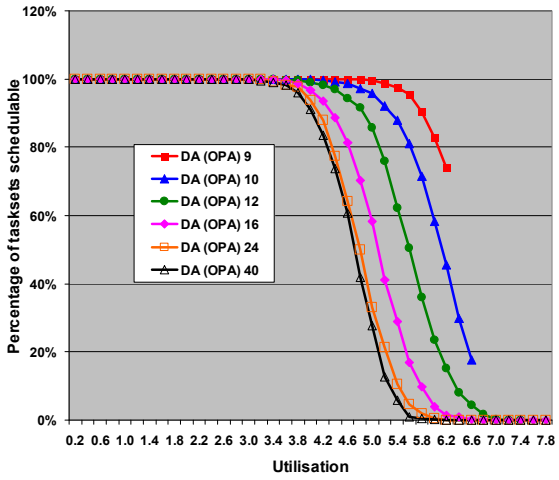

Figure 8: (taskset cardinality from 9 to 40)

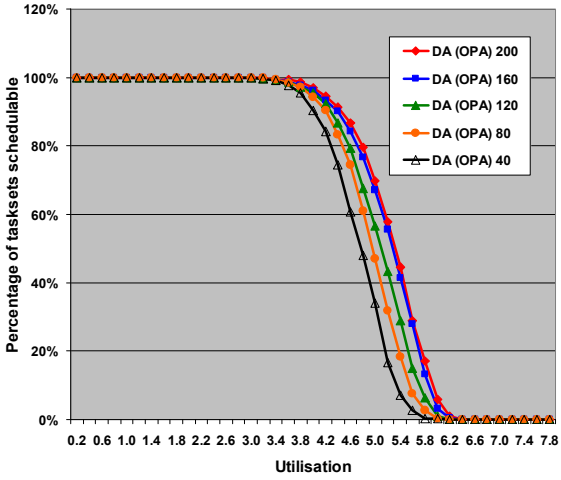

Figure 9: (taskset cardinality from 40 to 200)

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 recent work of Bertogna et al. [18] which showed that the best schedulability tests available for global FP scheduling using Deadline Monotonic Priority Ordering (DMPO) outperform 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:

- The observation that although Audsley's Optimal Priority Assignment algorithm [6], [7] 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 ("DA test") of Bertogna et al. [18], and the response time test of Andersson and Jonsson [1] for sporadic tasksets are OPA-compatible, while any exact test for periodic tasksets, and the response time test ("RTA test") of Bertogna and Cirinei [16] for sporadic tasksets are OPA-incompatible.
- Extension of the TkC [2] 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 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 test can schedule significantly more tasksets. Our study also showed that the DkC priority assignment policy is almost as effective as optimal priority assignment when applied in conjunction with the DA test, and similarly highly effective when applied with the RTA test.

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, the utilisation level at which 50% of the tasksets were schedulable increased from 0.28m or 0.29m (for the DA test or RTA test with DMPO) to 0.58m or 0.59m (for the RTA test with DkC priority assignment, or the DA test with optimal priority assignment). This represents an effective increase in the usable processing resource of 100% or more. 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.

In future, we intend to explore the use of the optimal priority assignment algorithm, and heuristic priority assignment policies, such as DkC, in conjunction with other schedulability tests for global FP scheduling. Other interesting areas of possible future work include the extension of optimal priority assignment techniques to uniform processors.

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 EU Jeopard and EU eMuCo projects.

References

[1] B. Andersson, J. Jonsson, “Some insights on fixed-priority pre-emptive non-partitioned multiprocessor scheduling”. In Proc. RTSS – Work-in-Progress Session, Nov. 2000.

[2] B. Andersson, J. Jonsson, “Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition”, In Proc. RTCSA, Dec. 2000.

[3] B. Andersson, S. Baruah, J. Jonsson, “Static-priority scheduling on multiprocessors”. In Proc. RTSS, pp. 193–202, 2001.

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

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

[6] N.C. Audsley, "Optimal priority assignment and feasibility of static priority tasks with arbitrary start times", Technical Report YCS 164, Dept. Computer Science, University of York, UK, Dec. 1991.

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

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

[9] T.P. Baker. “An analysis of fixed-priority scheduling on a multiprocessor”. Real Time Systems, 32(1-2), 49-71, 2006.

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

[11] T.P. Baker, S.K. Baruah. “Sustainable multiprocessor scheduling of sporadic task systems”. In Proc. ECRTS, pp. 141-150, 2009.

[12] S.K. Baruah, N. Fisher. “Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic ...” In Proc. of the 9th Int’l Conference on Distributed Computing and Networking, pp. 215-226, Jan 2008.

[13] S.K. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, “Implementation of a speedup-optimal global EDF schedulability test”, In Proc. ECRTS, pp. 259-268, 2009.

[14] S.K. Baruah, T.P. Baker, “An analysis of global EDF schedulability for arbitrary sporadic task systems. Real-Time Systems ECRTS special issue, 43(1): 3-24, Sept. 2009.

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

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

[17] M. Bertogna, “Real-Time Scheduling for Multiprocessor Platforms”. PhD Thesis, Scuola Superiore Sant’Anna, Pisa, 2007.

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

[19] E. Bini and G.C. Buttazzo. “Measuring the Performance of Schedulability tests”. Real-Time Systems, 30(1–2):129–154, May 2005.

[20] L. Cucu, J. Goossens, "Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors", In Proc. 11th IEEE International Conference on Emerging Technologies and Factory Automation, (ETFA'06), Sept. 2006.

[21] L. Cucu, J. Goossens, "Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary Deadline Periodic Systems ", In Proc. DATE, pp. 1635-1640, April 2007.

[22] L. Cucu, "Optimal priority assignment for periodic tasks on unrelated processors", In Proc. ECRTS WiP session. June 2008.

[23] R.I. Davis, A. Burns. “Robust Priority Assignment for Fixed Priority Real-Time Systems”. In Proc. RTSS, pp. 3-14, Dec. 2007.

[24] R.I. Davis, A. Burns, “Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems”. University of York, Dept. of Computer Science Technical Report YCS-2009-440, May 2009.

[25] S.K. Dhall, C.L. Liu, “On a Real-Time Scheduling Problem”, Operations Research, vol. 26, No. 1, pp. 127-140, 1978.

[26] A. Easwaran, I. Shin, I. Lee, “Toward Optimal Mutiprocessor Scheduling for Arbitrary Deadline …”. In Proc. RTSS WiP pp. 1-4, 2008.

[27] N. Fisher, S.K. Baruah. “Global Static-Priority Scheduling of Sporadic Task Systems on Multiprocessor Platforms.” In Proc. IASTED International Conference on Parallel and Distributed Computing and Systems. Nov. 2006.

[28] S. Lauzac, R. Melhem, and D. Mosse. “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, June 1998.

[29] C.L. Liu, “Scheduling algorithms for multiprocessors in a hard real-time …”. JPL Space Programs Summary, vol. 37-60, pp. 28-31, 1969.

[30] C.L. Liu, J.W. Layland, "Scheduling algorithms …", Journal of the ACM, 20(1): 46-61, Jan. 1973.

[31] J. Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," Performance Evaluation, 2(4): 237-250, Dec. 1982.

[32] O. Serlin, “Scheduling of time critical processes”. In proceedings AFIPS Spring Computing Conference, pp 925-932, 1972.