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

In this report, 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 is highly effective with performance that is close to optimal.

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 the 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 demands for increasing 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.

Global scheduling has the advantage that there are typically fewer context switches as the scheduler will only pre-empt a task when there are no processors idle. There is also no need to run task allocation algorithms when the set of tasks changes.

Multiprocessor 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 report, 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 [2] and Liu and Layland [3] showed that Rate Monotonic priority ordering (RMPO) is optimal for independent *synchronous* tasks (that share a common release time), that have *implicit deadlines* (equal to their periods).

In 1982, Leung and Whitehead [4] 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 [1] devised an optimal priority assignment (OPA) algorithm that solved the problem of priority assignment for asynchronous tasksets, and 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” [19].

In 1978, Dhall and Liu [19] showed that under global FP scheduling with RMPO, a set of tasks with implicit deadlines, and total utilisation just greater than 1 can be unschedulable on m processors. For this problem to occur requires at least one task with a high utilisation.

In 2000, Andersson and Jonsson [18], 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 implicit deadline tasksets.

In 2001, Anderson et al. [24] gave a utilisation bound for global FP scheduling using the RM-US$\{\varsigma\}$ priority assignment policy. RM-US$\{\varsigma\}$ gives the highest priority to tasks with utilisation greater than a threshold ς. Subsequently, Andersson and Jonsson [25] showed that the maximum utilisation bound for global FP scheduling, where priorities are defined as a scale invariant function of task worst-case execution times and periods, is: $(\sqrt{2}-1)m \approx 0.41m$.

In 2005, Bertogna [11] extended the work of Andersson et al. [24] to constrained deadline tasksets, 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 [11] provided a density-based schedulability test for DM-DS$\{\varsigma\}$.

In 2008, Andersson [26] proposed a form of Slack Monotonic priority assignment called SM-US$\{\varsigma\}$, which using a threshold of $2/(3+\sqrt{5})$, has a utilisation bound of $2/(3+\sqrt{5})m \approx 0.382m$.

More sophisticated schedulability tests for global FP scheduling have been developed using analysis of response times and processor load.

In 1998, Lundberg [5] gave a simple response time upper bound applicable to sporadic tasksets with constrained-deadline.

In 2001, Baker [6] 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 [7], [8], [9], [10], and global FP scheduling [12], [13], [21], [22], [23].

Baker’s work was subsequently built upon by Bertogna et al [11] in 2005, who developed sufficient schedulability tests for: any work conserving algorithm, global EDF, and global FP, based on bounding the maximum workload in a given interval. This 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 [13]. Bertogna and Cirinei [14] 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 [31]. This restriction manifests itself as the critical instant effect [27], where simultaneous release of tasks does not lead to worst-case response times. As a result, no exact tests are known for global FP scheduling of sporadic tasksets. Exact tests are only known for the strictly periodic case [28], [29], [30].

1.2. Intuition and motivation

The research described in this report 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 [8] showed that for sporadic tasksets with constrained deadlines, the response time test [14] for global FP scheduling – using DMPO, outperforms all other known tests for: global FP, global EDF and also similar tests for EDZL [15] (EDF until zero laxity), which is 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 [4], this optimality does not extend to multiprocessors.

In this report, we prove that Audsley’s Optimal Priority Assignment (OPA) algorithm [1], originally devised for uniprocessor scheduling, is applicable to the multiprocessor case provided that the schedulability test used meets four 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 [18], 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 report 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 a 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 report, we are interested in global FP scheduling of an application on a homogeneous multiprocessor system comprising *m* identical processors. The application (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).

Application tasks may arrive either *periodically* at fixed intervals of time, or *sporadically* after some minimum inter-arrival time has elapsed. 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 that all tasks have constrained deadlines, less than or equal to their periods ($D_i \le T_i$). 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. It is assumed that once a task starts to execute it will not suspend itself.

Each task gives rise to a potentially infinite sequence of invocations (or jobs). The arrival times of the jobs of different tasks are assumed to be independent. 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. Schedulability and optimality

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*: For a given system model, a priority assignment policy *P* is referred to as *optimal* with respect to a schedulability test *S*, if there are no tasksets, compliant with the system 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*.

3. Recapitulation of schedulability tests

In this section, we outline two sufficient schedulability tests for global fixed priority scheduling developed by Bertogna et al [13], and Bertogna and Cirinei [14]. Both of these tests are based on the fundamental strategy derived by Baker [6], the outline of which is as follows:

- 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.
- 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.
- Derive an upper bound I^{UB} on the maximum interference in the interval due to other tasks.
- 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 [13], 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 invocation 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 invocations executing as early as possible – see Figure 1.

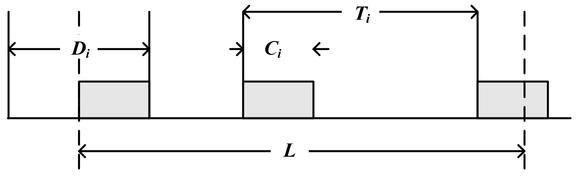

Figure 1

Bertogna et al. [13] 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[1] used in [13].

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 [13] for ease of comparison with the schedulability test given in [14].

Bertogna and Cirinei [14] 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 essentially the same logic as [13], 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 [14] 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[2].

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:

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

Where $N_i(L)$ is again given by Equation (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 R_k^{UB} is given by:

$$I_i(R_k^{UB}) = \min(W_i^R(R_k^{UB}), R_k^{UB} - C_k + 1) \quad (6)$$

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 [14]).

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{\forall i \in hp(k)} I_i(R_k^{UB}) \right\rfloor \quad (7)$$

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 report, we will 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 (7), as the "RTA test".

4. Priority assignment

In 2000, Andersson and Jonsson [17] made the following observation: *"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 the those tasks".*

Andersson and Jonsson concluded that even if an exact schedulability test were known, then it would not be possible to use Audsley's OPA algorithm [1] 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.

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, [1]

[1] 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.

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

derived for uniprocessor systems.

4.1. Optimal priority assignment

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

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 compared to 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.

```
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
```

Let *S* be some schedulability test for global FP scheduling which complies with the following three 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 $Q_{n-1} .. Q_0$, 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 *k* in priority ordering Q_k.
- $hp(i, Q_k)$ is the set of tasks with priority strictly higher than *k* in priority ordering Q_k.
- $lep(i, Q_k)$ is the set of tasks with priority lower than or equal to *k* in priority ordering Q_k.
- $lp(i, Q_k)$ is the set of tasks with priority strictly lower than *k* in priority ordering Q_k.

In the proof which follows, we use *k* to represent both the iteration of the OPA algorithm (the priority level examined) and 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 $hep(k, Q_k)$ = $hep(k, P)$, and so, given Condition 1, on iteration *k*, the OPA algorithm is guaranteed to find a task in the set of unassigned tasks (i.e. $hep(k, P)$) that is schedulable at priority *k* according to test *S*. This task is designated $P(k)$.

There are two cases to consider:

1. $P(k)$ is the task at priority *k* in Q_k (i.e. $Q_k(k)$), 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. $P(k)$ 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 (see Figure 2 below).

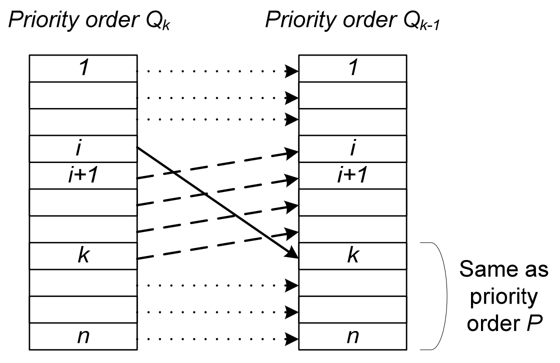

Figure 2

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 $Q_{k-1}(k) = P(k)$: As the tasks of lower priority than *k* are the same in both Q_k and *P*, the OPA algorithm selected task $P(k)$ from the set of tasks $hep(k,Q_k)$ on the basis of its schedulability at priority *k* with the set of tasks $hep(k,Q_k) - k$ = $hp(k,Q_{k-1})$ 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 $hp(k,Q_{k-1})$.
4. $lp(k,Q_{k-1})$: These tasks are assigned the same priorities in both Q_k and Q_{k-1}. As $hep(k,Q_{k-1}) = hep(k,Q_k)$, and given Condition 1, they all 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 values of *k* from *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* □

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

As shown by Andersson and Jonsson [17], there exist tasksets for which the exact response time and hence schedulability of a task depends not only on the set of higher-priority tasks, but also on the relative priority ordering of those tasks. This non-compliance with Condition 1 means that any exact test is OPA-incompatible (as noted in [17]).

The dependency of schedulability on the relative priority ordering of higher priority tasks also applies to the RTA test [14] given by Equation (7). Here, the workload $W_i^R(L)$ (Equation (5)) depends on the response times of higher priority tasks, so this test is non-compliant with Condition 1 and therefore OPA-incompatible. By contrast, both the simple response time test of Lundberg [5], and the DA test [13] given by Equation (4), comply with Conditions 1-3 and are therefore OPA-compatible.

The OPA algorithm solves the problem of priority assignment for all OPA-compatible global FP schedulability tests. For OPA-incompatible schedulability tests, optimal priority assignment remains an open problem, as checking all *n*! priority orderings is infeasible for tasksets with non-trivial cardinality.

4.2. Heuristic priority assignment

In this section, we investigate heuristic priority assignment policies for OPA-incompatible schedulability tests.

In his thesis [8], Bertogna evaluates the effectiveness of a number of different schedulability tests. Bertogna's experiments show that using DMPO the RTA test outperforms all other 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 heuristic priority assignment policy can be obtained by re-arranging Equation (4):

$$D_k - C_k \geq \left\lfloor \frac{1}{m} \sum_{\forall i \in hp(k)} I_i^D(D_k) \right\rfloor \tag{8}$$

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 [18] 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:

$$k = \frac{m - 1 + \sqrt{5m^2 - 6m + 1}}{2m} \tag{9}$$

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 (9).

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 RM-US$\{\varsigma\}$ [34] and SM-US$\{\varsigma\}$ [26] priority assignment policies. These algorithms although more complex, were found to be no more effective than the DkC policy. (Appendix B gives details of these policies and their performance).

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, 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 the target utilisation (Ut) required, and generate a new taskset by returning to step 1.

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

The naive approach is not viable in practice; however, the UUnifast algorithm of Bini and Buttazzo [20] (pseudo code given above), replicates the same unbiased distribution of task utilisations. 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.

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 [8], [13], [14], [15]. 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:

1. Run UUnifast with the required target utilisation Ut (which may be > 1). UUnifast will generate an unbiased distribution of n task utilisation values in the range [0, Ut] which sum to Ut.
2. If the taskset generated has a task with $U_i > 1$, then discard the taskset and repeat step 1.

The valid tasksets that remain have an unbiased distribution of task utilisations in the range [0, min(Ut,1)] which sum to Ut. This is exactly what is required to study the effectiveness of multiprocessor schedulability tests. Unfortunately, there is a drawback to this 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 the vast majority of commercial real-time systems using multiprocessors will have significantly more tasks than processors. In any case, we can simply set a pragmatic limit on the proportion of tasksets we are prepared to discard (say 1,000 times the number of useful tasksets generated) and investigate as much of the problem space as possible within this limit.

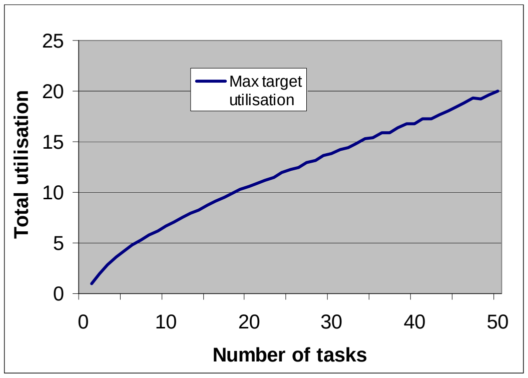

Figure 3

Figure 3 shows the maximum taskset utilisation that UUnifast is able to generate plotted against taskset cardinality. For example, UUnifast 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 algorithm is independent of the number of processors).

As we will see in the next section, the scope of this taskset generation method is sufficient to examine the effectiveness of schedulability tests for 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 (7)), which is OPA-incompatible. The priority assignment policies studied are DMPO, D-CMPO, DkC and Audsley's optimal priority assignment (OPA) algorithm (DA test only).

The results presented in this section are for constrained-deadline tasksets, Appendix A presents the results of essentially the same experiments applied to implicit-deadline tasksets.

6.1. Parameter generation

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

- Task utilisations were generated using the UUnifast algorithm, discarding any tasksets with a task with $U_i > 1$. (The maximum number of taskset generation attempts was 1000 times the number of tasksets required).
- Task periods were generated according to a log-uniform distribution[3] with a factor of 10^3 difference between the minimum and maximum possible task period. This represents a spread of task periods of 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, 100-1000ms), 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.

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

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. (Note that we varied the number of tasks across these experiments to keep the number of tasks per processor constant at 5).

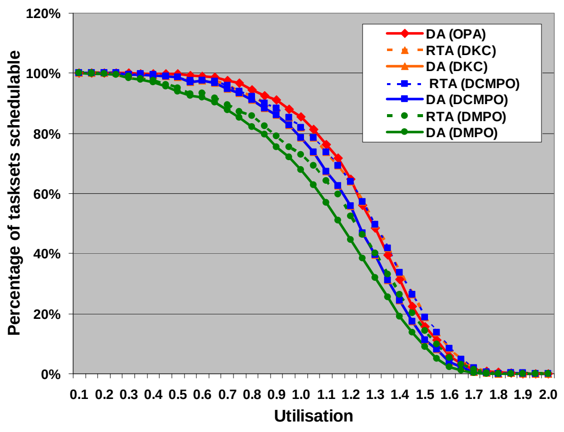

Figure 4: (2 processors, 10 tasks)

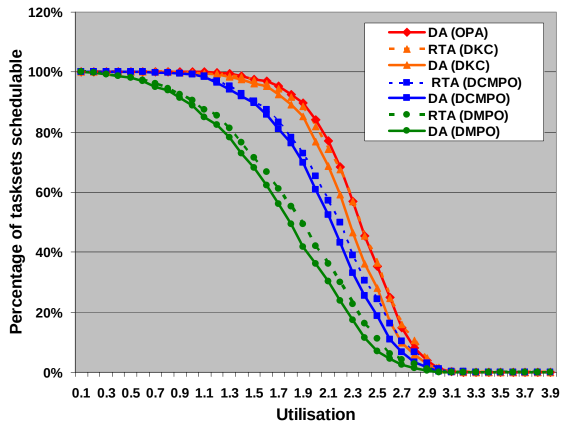

Figure 5: (4 processors, 20 tasks)

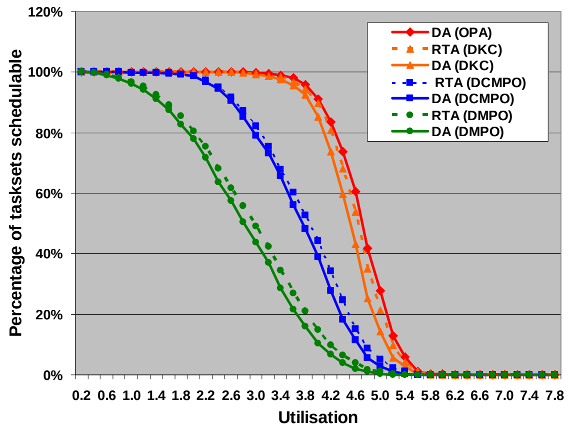

Figure 6: (8 processors, 40 tasks)

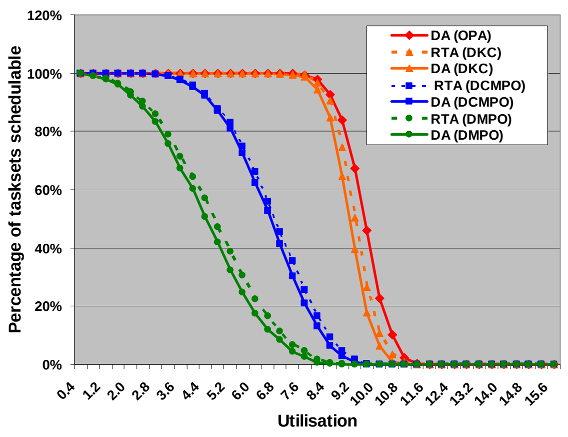

Figure 7: (16 processors, 80 tasks)

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).

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 (= 0.28*m*); however, using the OPA algorithm, approx. 50% of the tasksets are schedulable according to the same test at a utilisation level of 9.4 (= 0.59*m*) 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 (Equation (9)). Comparison between the four figures shows that the difference between OPA and DMPO becomes more significant 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.

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. Experiment 2 below examines the dependency on the number of tasks.

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.

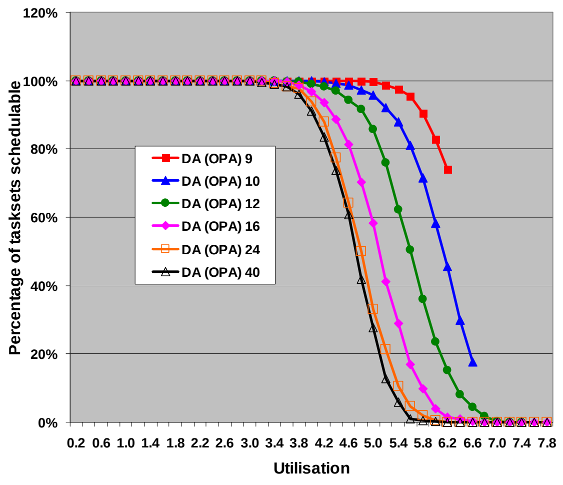

Figure 8: (taskset cardinality from 9 to 40)

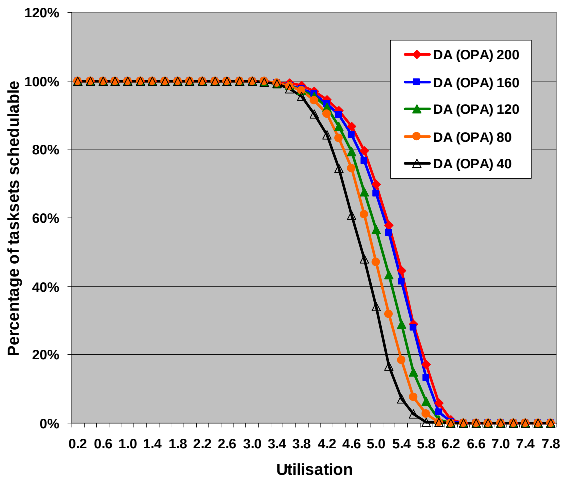

Figure 9: (taskset cardinality from 40 to 200)

There are some data points missing from the right hand side of the Figure 8. This is because the UUnifast algorithm was unable to generate tasksets with cardinality 9 and utilisation greater than 6.6 (or cardinality 10 and utilisation greater than 6.8); however, despite this the trend is 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 that tasksets with a larger number of tasks are more difficult to schedule; however, 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 (5)). 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 2*m* 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 10*m*, 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 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).

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 report 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. [13] which showed that the best schedulability tests available for global FP scheduling using Deadline Monotonic Priority Ordering (DMPO) outperform the best tests 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, we found that DMPO cannot even be considered a good heuristic for multiprocessors.

The key contributions of this report are as follows:

- The observation that although Audsley's Optimal Priority Assignment algorithm [1] cannot be applied to an exact schedulability test for global fixed priority scheduling (should such a test be derived), this does *not* preclude its use in conjunction with sufficient schedulability tests.
- Proof that Audsley's OPA algorithm is the optimal priority assignment policy with respect to any global FP schedulability test 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. [13], and the response time test of Lundberg [5] are OPA-compatible, while any exact test (that might be derived), and the response time test ("RTA test") of Bertogna and Cirinei [14] are OPA-incompatible.
- Extension of the TkC [18] priority assignment policy to constrained deadline tasksets forming a DkC priority assignment policy. This heuristic policy can be used in conjunction with any schedulability test.
- Adaptation of the UUnifast task parameter generation algorithm to the multiprocessor case. This enables the generation of tasksets with specific parameter settings, facilitating an empirical study of the dependency of schedulability test effectiveness without the problem of confounding variables.
- An empirical study showing that using Audsley's 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 in conjunction 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 0.28*m* or 0.29*m* (for the DA test or RTA test with DMPO) to 0.58*m* or 0.59*m* (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.

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.

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.

Appendix A: Implicit-deadline tasksets

In this appendix, we present the results of Experiments 1 and 2 (Sections 6.2 and 6.3), repeated for implicit-deadline tasksets.

A.1 Experiment 1 (Priority assignment)

In this experiment, we examined 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 (7)), which is OPA-incompatible. The priority assignment policies studied were:

- DMPO (which reduces to Rate Monotonic priority ordering [2], [3] in the case of implicit-deadline tasksets),
- D-CMPO,
- DkC (which reduces to TkC [17] in the case of implicit-deadline tasksets), and
- Audsley's optimal priority assignment (OPA) algorithm [1] (DA test only).

We used exactly the same parameter settings described in Section 6.1, save for task deadlines which were set equal to their periods. (By comparison, when studying constrained-deadline tasksets, task deadlines were chosen according to a uniform random distribution, in the range $[C_i, T_i]$).

Figures 10 to 13 illustrate the impact of each of the priority assignment policies on the percentage of tasksets deemed schedulable by the two schedulability tests for 2, 4, 8, and 16 processors respectively.

In the two processors case (Figure 10) all of the priority assignment policies have similar performance[4], and it is the effectiveness of the schedulability tests which dominates the results obtained. As the number of processors is increased, from 2 up to 16 (in Figure 13) the difference in performance between the two schedulability tests diminishes and 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 test using DMPO (approx. 9.0 = 0.56*m*), versus using the optimal priority assignment algorithm (approx. 11.4 = 0.71*m*). This difference corresponds to an effective increase in usable processing capacity of around 27%.

It is interesting to compare the graphs for implicit-deadline tasksets (Figures 10 to 13) with their counterparts for constrained-deadline tasksets (Figures 4 to 7). We conclude from this comparison, that the selection of an appropriate priority ordering is more significant in the general case where task deadlines are permitted to be less than or equal to their periods.

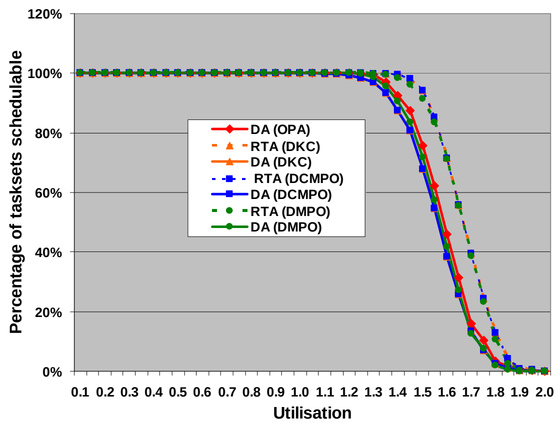

Figure 10: (2 processors, 10 tasks)

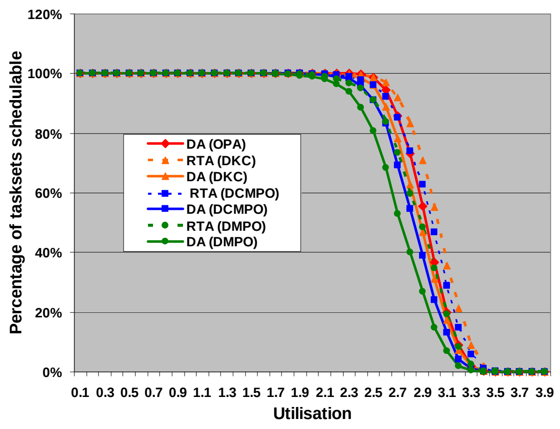

Figure 11: (4 processors, 20 tasks)

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

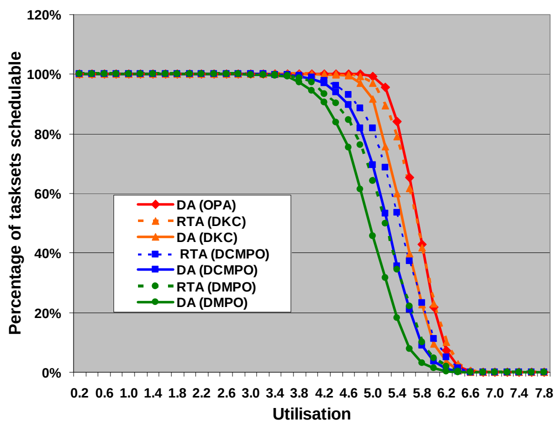

Figure 12: (8 processors, 40 tasks)

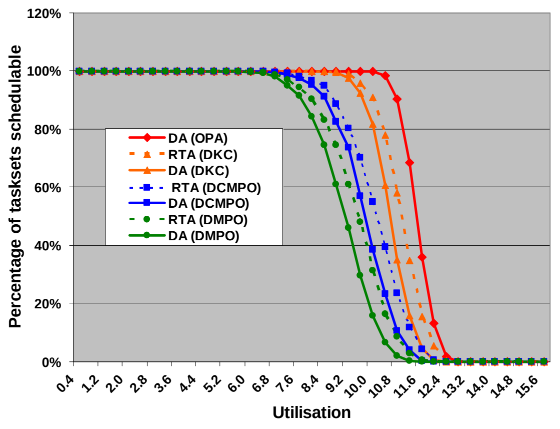

Figure 13: (16 processors, 80 tasks)

A.2 Experiment 2 (Number of tasks)

In this experiment we investigated the effect of varying the number of tasks. Figure 14 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). Figure 15 shows similar data for tasksets of cardinality 40, 80, 120, 160, and 200. This data is for implicit-deadline tasksets and is directly comparable with that shown in Figure 8 and Figure 9 for constrained-deadline tasksets.

There are some data points missing from the right hand side of the Figure 14. Again, this is because the UUnifast algorithm was unable to generate tasksets with cardinality 9 and utilisation greater than 6.6 (or cardinality 10 and utilisation greater than 6.8).

Figure 14 is similar to Figure 8 in that the percentage of schedulable tasksets decreases as the number of tasks is increased from 9 to 24. Further, Figure 15 is similar to Figure 9 in that the percentage of schedulable tasksets increases again as the number of tasks is increased from 40 to 200. The explanation for this behaviour is given in Section 6.3.

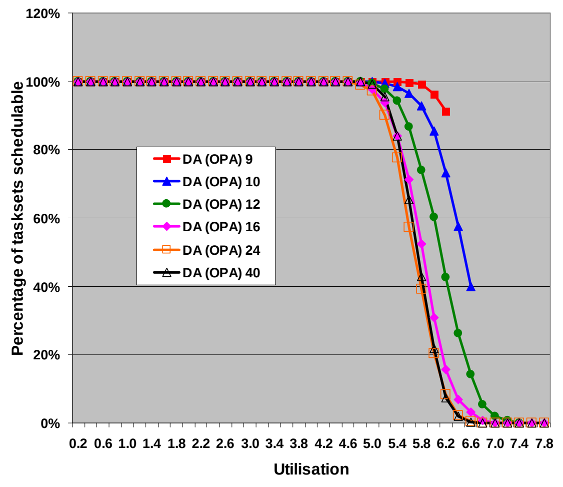

Figure 14: (taskset cardinality from 9 to 40)

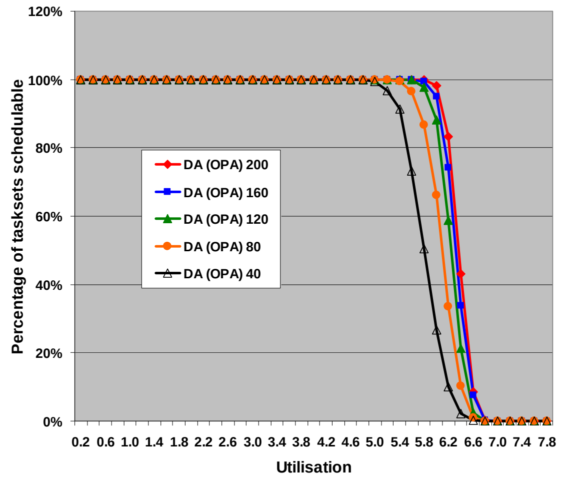

Figure 15: (taskset cardinality from 40 to 200)

Appendix B: Other heuristic priority assignment policies

In this section, we examine the performance of two heuristic priority assignment algorithms, derived from RM-US{ς} [34] and SM-US{ς} [26].

B.1 Implicit-deadline tasksets

The RM-US{ς} priority assignment policy, was derived by Andersson et al. [34] with the aim of addressing the "Dhall effect" [19] for implicit-deadline tasksets using global FP scheduling. RM-US{ς} assigns the highest priority[5] to tasks with utilisation greater than some threshold ς. The remaining tasks are then assigned priorities in Rate Monotonic priority order.

[5] 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.

In 2002, Lundberg [35] showed that 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:

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

The SM-US{ς} priority assignment policy was derived by Andersson and Jonsson [26] 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 and Jonsson [26] showed that using a threshold of $2/(3+\sqrt{5})$ results in the following utilisation bound for SM-US{ς}:

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

Figure 16 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 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}.

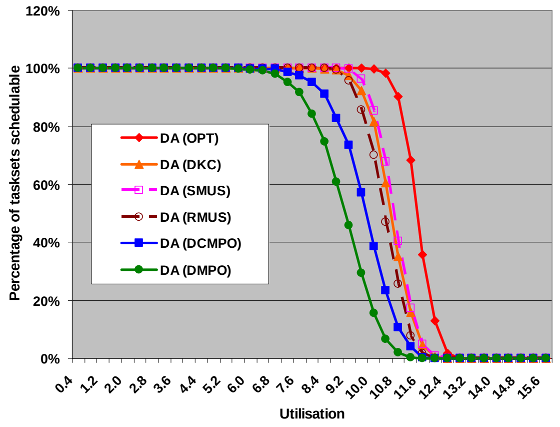

Figure 16: (16 processors, 80 tasks)

From Figure 16, 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.

B.2 Constrained-deadline tasksets

Bertogna et al. [11] 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:

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

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 17 below shows the results of essentially the same experiment as Figure 16; 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 worse than that of DMPO.

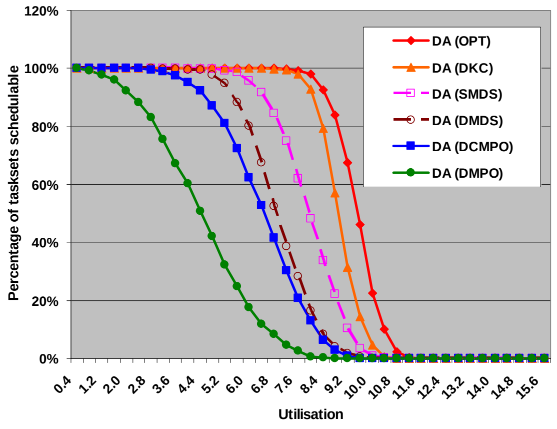

Figure 17: (16 processors, 80 tasks)

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. [11] and Andersson and Jonsson [26] were able to derive appropriate thresholds for DM-DS{ς} and SM-US{ς} 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{ς} and SM-US{ς} priority assignment policies, which address the problem of selecting an appropriate threshold. Here, we employ an idea used by Goosens et al. [33] and Baker [32] 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 18 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 schedulability test. This data is again for constrained-deadline tasksets and so is directly comparable with that presented in Figure 17.

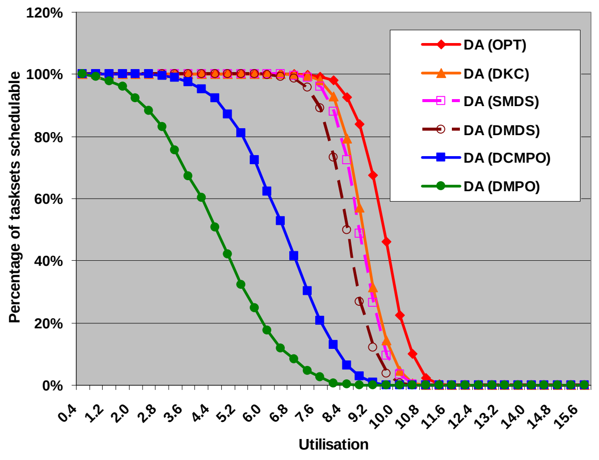

Figure 18: (16 processors, 80 tasks)

From Figure 18, it is clear that the performance of the SM-DS(*h*) priority assignment algorithm is very similar to that of DkC, with DM-DS(*h*) providing somewhat inferior performance. The results shown in Figure 18 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

[1] 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, December 1991.

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

[3] C. L. Liu and J. W. Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment", Journal of the ACM, 20(1): 46-61, January 1973.

[4] 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, December 1982.

[5] L. Lundberg, “Multiprocessor scheduling of age constraint processes”. Proceedings of the International Conference on Real-Time Computing Systems and Applications (RTCSA), 1998.

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

[7] S.K. Baruah, “Sustainable schedulability analysis of global EDF. In Proc. RTSS 2008.

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

[9] Baruah, S.K., Baker, T.P.: “An analysis of global EDF schedulability for arbitrary sporadic task systems. Real-Time Systems ECRTS special issue, to appear 2009.

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

[11] M. Bertogna, M. Cirinei, and 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, Dec. 2005.

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

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

[14] M. Bertogna, M. Cirinei, “Response Time Analysis for global scheduled symmetric multiprocessor platforms”. In proceedings 28th IEEE Real-Time Systems Symposium, pp. 149-158, 2007.

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

[16] R.I. Davis, A. Burns. “Robust Priority Assignment for Fixed Priority Real-Time Systems”. In proceedings IEEE Real-Time Systems Symposium pp. 3-14. Tucson, Arizona, USA. December 2007

[17] B. Andersson, J. Jonsson, “Some insights on fixed-priority pre-emptive non-partitioned multiprocessor scheduling”. In Proceedings of the Real-Time Systems Symposium – Work-in-Progress Session, November 2000.

[18] B. Andersson, J. Jonsson, “Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition”, Proceedings of the International Conference on Real-Time Computing Systems and Applications, Cheju Island, Korea (December 2000).

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

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

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

[22] N. Fisher, S.K. Baruah. “Global Static-Priority Scheduling of Sporadic Task Systems on Multiprocessor Platforms.” Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, Dallas, TX. November 2006.

[23] S.K. Baruah and N. Fisher. “Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems.” Proceedings of the 9th International Conference on Distributed Computing and Networking, Kolkata, India. January 2008.

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

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

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

[27] 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, pages 188–195, June 17–19, 1998.

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

[29] Cucu L. and Goossens, J. "Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary Deadline Periodic Systems ", 10th Design, Automation and Test in Europe (DATE'07), ACM Press, Nice, April 2007

[30] Cucu L., "Optimal priority assignment for periodic tasks on unrelated processors", Euromicro Conference on Real-Time Systems (ECRTS'08), WIP session, Prague, June 2008.

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

[32] Baker, T. P. “An analysis of EDF scheduling on a multiprocessor”. IEEE Trans. on Parallel and Distributed Systems, 15(8):760–768, Aug. 2005.

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

[34] B. Andersson, S. Baruah, and J. Jonsson. Static-priority scheduling on multiprocessors. In Proc. 22nd IEEE Real-Time Systems Symposium, pages 193–202, London, UK, Dec. 2001.

[35] Lundberg, L.: Analyzing Fixed-Priority Global Multiprocessor Scheduling. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’02), 2002.