On Optimal Priority Assignment for Response Time Analysis of Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Hard 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 investigates the problem of optimal priority assignment in multiprocessor real-time systems using global fixed task-priority pre-emptive scheduling.

Previous work in this area showed that arguably the most effective pseudo-polynomial schedulability tests for global fixed priority pre-emptive scheduling, based on response time analysis, are not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm.

In this paper, we derive upper and lower bounds on these response time tests that are compatible with the OPA algorithm. We show how these bounds can be used to limit the number of priority ordering combinations that need to be examined, and thus derive an optimal priority assignment algorithm with backtracking that is compatible with response time analysis. We show that response time analysis combined with the OPA-backtracking algorithm dominates previous approaches using OPA-compatible polynomial-time schedulability tests.

1. Introduction

Approaches to multiprocessor real-time scheduling, can be categorised into two broad classes: *partitioned* and *global*. Partitioned approaches allocate each task to a single processor, dividing the multiprocessor scheduling problem into one of task allocation followed by uniprocessor scheduling. In contrast, global approaches allow tasks to migrate from one processor to another at run-time. 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 [25] and Liu and Layland [23] 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 [24] 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 [5], [6] 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). Subsequently, George et al. [21] showed that Audsley's OPA algorithm also solves the problem of determining the optimal priority assignment for system using fixed priority non-preemptive scheduling.

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 [18] 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 2001, Andersson et al. [2] 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 utilisation greater than a threshold ς. In 2003, Andersson and Jonsson [3] 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 [11] extended the work of Andersson et al. [2] to sporadic tasksets with constrained deadlines forming the DM-DS$\{\varsigma\}$ priority assignment policy. In 2008, Andersson [4] 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 [7] 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 [10], [11], [13], [14], [8], [19], [22].

Baker's work was subsequently built upon by Bertogna

et al. [11] in 2005, (see also Bertogna and Cirinei [14]). They developed sufficient schedulability tests for global FP scheduling based on bounding the maximum workload in a given interval. In 2007, Bertogna and Cirinei [12] adapted this approach to iteratively compute an upper bound on the response time of each task, using the upper bound response times of other tasks to limit the amount of interference considered. In 2009, Guan et al. [22] extended the response time analysis of Bertogna and Cirinei [12], limiting the amount of carry-in interference due to jobs released prior to the start of the interval, using ideas from [9]. Guan et al. also extended the approach to tasksets with arbitrary deadlines.

In 2009, Davis and Burns [16] showed that priority assignment is fundamental to the effectiveness of global FP scheduling. They proved that Audsley's Optimal Priority Assignment (OPA) algorithm [5], [6], originally developed for uniprocessor FP scheduling, is applicable to some of the sufficient tests for global FP scheduling. These tests are referred to as *OPA-compatible* [16]. The response time test of Andersson and Jonsson [1] and the deadline-based test of Bertogna et al. [14] are OPA-compatible [16], while the response time analysis of Bertogna and Cirinei [12], and Guan et al. [22] are *OPA-incompatible*.

Davis and Burns [17] also studied the effectiveness of various heuristic priority assignment policies, including DkC [16], RM-US$\{\varsigma\}$ [2], DM-DS$\{\varsigma\}$ [11], and SM-US$\{\varsigma\}$ [4], while all of these policies typically outperformed DMPO in terms of the number of tasksets found to be schedulable by the schedulability tests given in [14] and [12]. Their performance was shown to fall significantly below that of optimal priority assignment.

1.2. Intuition and motivation

Dynamic priority scheduling has the potential to schedule many more tasksets than fixed task or fixed job priority scheduling algorithms. However, this theoretical advantage must be balanced against the increased overheads inherent in dynamic changes in priority. For example, algorithms such as LLREF [15] and LRE-TL [20] which are optimal for implicit-deadline periodic and sporadic tasksets respectively, can in the worst-case result in n-1 pre-emptions per job release, where n is the number of tasks. In systems with a large number of tasks, this level of pre-emptions leads to prohibitively high overheads. By contrast, global FP scheduling results in at most one pre-emption per job release.

In this paper, we are interested in priority assignment policies that enable the maximum possible guaranteed performance to be obtained from the simplest possible global scheduling algorithm; global FP scheduling.

The motivation for our research comes from the fact that arguably the most effective schedulability test for global FP scheduling, the response time test of Bertogna and Cirinei [12] as improved by Guan et al. [22], is not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm. Hence the current state-of-the-art [16] involves either combining optimal priority assignment with polynomial-time schedulability tests [14], which are dominated by response time analysis, or combining heuristic priority assignment policies such as DkC [16], which are dominated by OPA, with response time analysis. Clearly, improved performance can be obtained if a method of optimal priority assignment can be found that is compatible with response time analysis.

In this paper, we derive upper and lower bounds on response time analysis that are OPA-compatible. We show how these bounds can then be used to reduce the number of priority ordering combinations that need to be examined, and thus derive an optimal priority assignment algorithm with backtracking that is compatible with response time analysis. While this algorithm can still require that $n!/m!$ priority orderings are considered in the worst case (where n is the number of tasks, and m is the number of processors), in practice the upper and lower bounds can significantly reduce the number of priority ordering combinations that need to be examined. We show that response time analysis combined with the OPA-backtracking algorithm dominates previous approaches using OPA-compatible polynomial-time schedulability tests.

1.3. Organisation

The remainder of this paper is organised as follows: Section 2 describes the terminology, notation and system model used. Section 3 describes response time analysis for global FP scheduling, while Section 4 discusses optimal priority assignment. Section 5 derives upper and lower bounds on response time analysis and shows that they are OPA-compatible. Section 6 introduces an optimal priority assignment algorithm with backtracking that is compatible with response time analysis. Section 7 presents the results of an empirical investigation into the effectiveness of the OPA-Backtracking algorithm combined with response time analysis. Finally, Section 8 concludes with a summary and an outline of future work.

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).

Tasks are assumed to comply with the *sporadic* task model. In this model, tasks give rise to a potentially infinite sequence of jobs. Each job of a task may arrive at any time once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task.

Each task 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 costs of pre-emption, migration, and the run-time operation of the scheduler are 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.

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. Using an optimal priority assignment policy for a sufficient test 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.

3. Response Time Analysis

In this section, we outline the pseudo-polynomial time sufficient test for global fixed priority scheduling of sporadic tasksets introduced by Bertogna and Cirinei [12]. This test was subsequently improved by Guan et al. [22], using ideas from [9] to limit the amount of carry-in interference.

In [12], Bertogna and Cirinei showed that if task τ_k is schedulable in an interval of length *L*, then an upper bound on the interference in that interval due to a higher priority task τ_i with a carry-in job, released prior to the start of the interval, is given by[1]:

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

where, $W_i^R(L)$ is an upper bound on the workload of task τ_i in an interval of length *L*:

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

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

$$N_i^R(L) = \left\lfloor \frac{L + X_i - C_i}{T_i} \right\rfloor \quad (3)$$

where X_i is the upper bound response time R_i^{UB} of higher priority task τ_i. The response time test of Bertogna and Cirinei [12] may be expressed as follows:

RTA test for global FP scheduling (Theorem 7 in [12]): A sporadic taskset is schedulable, if for every task τ_k in the taskset, the upper bound response time R_k^{UB} computed via the fixed point iteration given in Equation (4) is less than or equal to the task's deadline:

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{\forall i \in hp(k)} I_i^R(R_k^{UB}) \right\rfloor \quad (4)$$

where *hp*(*k*) is the set of tasks with priorities higher than k. Iteration starts with $R_k^{UB} = C_k$, and continues until the value of R_k^{UB} converges or until $R_k^{UB} > D_k$, in which case task τ_k is unschedulable. We note that using Equation (4), task schedulability needs to be determined in priority order, highest priority first, as upper bounds on the response times of higher priority tasks are required for computation of the interference term $I_i^R(R_k^{UB})$.

In [22], Guan et al. showed that at most *m*-1 higher priority tasks with carry-in jobs may contribute interference in the worst-case, and used this result to improve the above test as follows:

Guan et al. [22] showed that if task τ_i does not have a carry-in job, then the interference term is given by:

$$I_i^{NC}(L) = \min(W_i^{NC}(L), L - C_k + 1) \quad (5)$$

where:

$$W_i^{NC}(L) = N_i^{NC}(L)C_i + \min(C_i, L - N_i^{NC}(L)T_i) \quad (6)$$

and

$$N_i^{NC}(L) = \lfloor L / T_i \rfloor \quad (7)$$

The difference between the two interference terms (Equation (1) and Equation (5)) is then given by:

$$I_i^{DIFF-R}(L) = I_i^R(L) - I_i^{NC}(L) \quad (8)$$

*RTA-LC test for global FP scheduling (*Guan et al. [22]*): A sporadic taskset is schedulable, if for every task τ_k in the taskset, the upper bound response time R_k^{UB} computed via*

[1] Note we adopt the approach to time representation used in [14]. 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.

the fixed point iteration given in Equation (9) is less than or equal to the task's deadline:

$$R_k^{UB} \leftarrow C_k + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(R_k^{UB}) + \sum_{i \in MR(k,m-1)} I_i^{DIFF-R}(R_k^{UB}) \right) \right\rfloor \quad (9)$$

where $MR(k, m\text{-}1)$ is the subset of the $min(k, m\text{-}1)$ tasks with the largest values of $I_i^{DIFF-R}(R_k^{UB})$, given by Equation (8), from the set of tasks $hp(k)$. Iteration again starts with $R_k^{UB} = C_k$, and continues until the value of R_k^{UB} converges or until $R_k^{UB} > D_k$, in which case task τ_k is unschedulable.

We note that the RTA-LC test reduces to the RTA test if the $I_i^{DIFF-R}(R_k^{UB})$ term is included for all of the higher priority tasks, rather than just those with the m-1 largest values. Hence the RTA-LC test dominates the earlier RTA test. Both RTA and RTA-LC tests dominate the polynomial-time deadline-based analysis of Bertogna and Cirinei [14].

4. Optimal Priority Assignment

In this section, we outline prior results on optimal priority assignment for global FP scheduling.

In 2009, Davis and Burns [16] proved that Audsley's Optimal Priority Assignment (OPA) algorithm [5], [6] is applicable to any schedulability test S for global FP scheduling that 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).

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 test S
            with all unassigned tasks assumed to have higher
            priorities){
            assign τ to priority k
            break (continue outer loop)
        }
    }
    return unschedulable
}
return schedulable
```

Figure 1: OPA algorithm

The pseudo code for Audsley's OPA algorithm, using some schedulability test S, is given in Figure 1 above.

For n tasks, the OPA algorithm performs at most $n(n+1)/2$ schedulability tests and is guaranteed to find a priority assignment that is schedulable according to schedulability test S if one exists. This is a significant improvement over inspecting all $n!$ possible priority orderings. (Such an exhaustive search becomes intractable even for modest values of n. For example, $12! \approx 2^{32}$). Note that the OPA algorithm does not specify the order in which tasks should be tried at each priority level.

As the RTA and RTA-LC tests are dependent on the upper bound response times of higher priority tasks, which are themselves dependent on the relative priority ordering of those tasks, these tests do not comply with Condition 1, and so are not compatible with the OPA algorithm (For a proof see Theorem 4 of [16]).

5. Upper and Lower Bounds

In this section, we derive a schedulability test (the D-RTA-LC test) that lower bounds the RTA-LC test given in Equation (9), and a pseudo-schedulability condition (the C-RTA condition) that upper bounds the RTA-LC test. We then prove some theorems about these conditions and the RTA-LC test. These Theorems are subsequently used in the construction of the OPA-backtracking algorithm described in Section 6.

The D-RTA-LC test is formed from Equation (9) by using the largest possible schedulable value that the response time upper bound of each higher priority task can take (i.e., by setting $X_i = D_i$ instead of R_i^{UB} in Equations (2) and (3)).

We observe that as Equation (2) is monotonically non-decreasing in X_i, then the RTA-LC test dominates the D-RTA-LC test. By this we mean that any taskset / priority ordering combination deemed schedulable by the D-RTA-LC test will also be deemed schedulable by the RTA-LC test. Thus the D-RTA-LC test is a lower bound on task schedulability under the RTA-LC test.

The C-RTA[2] condition is formed from Equation (9) by using the smallest possible value that the response time upper bound of each higher priority task could take (i.e. by setting $X_i = C_i$ instead of R_i^{UB} in Equations (2) and (3)). It is important to note that the C-RTA condition is *not* a schedulability test; it may deem tasksets / priority ordering combinations schedulable that are unschedulable. It may also deem tasksets / priority ordering combinations unschedulable, when in fact they are schedulable.

We observe that as Equation (2) is monotonically non-decreasing in X_i and the minimum possible value for R_i^{UB} is C_i, then the C-RTA condition dominates the RTA-LC test. By this, we mean that any taskset / priority ordering combination deemed unschedulable by the C-RTA condition will *necessarily* also be deemed unschedulable by the RTA-LC test. Thus the C-RTA condition forms an upper bound on task schedulability under the RTA-LC test.

Theorem 1: The D-RTA-LC schedulability test is OPA-compatible.

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

Inspection of Equation (9) and its component equations

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

shows that the upper bound response time R_k^{UB} computed for task τ_k depends on the set of higher priority tasks, and their parameters (C_i, D_i, T_i) but not on their upper bound response times (as $X_i = D_i$) or their relative priority ordering, hence Condition 1 holds.

Equation (9) has no dependency on the set of tasks with priorities lower than *k*, hence Condition 2 holds.

Consider two tasks *A* and *B* initially at priorities *k* and *k*+1 respectively. The upper bound response time of task *B* cannot increase when it is shifted up one priority level to priority *k*, as the only change in the response time computation (Equation (9)) is the removal of task *A* from the set of tasks that have higher priority than task *B,* hence Condition 3 holds □

Theorem 2: The C-RTA condition is OPA-compatible.

Proof: Follows exactly the same logic as the proof of Theorem 1, noting that $X_i = C_i$ in this case □

Theorem 3: The RTA-LC test, although OPA-incompatible due to non-compliance with Condition 1 [16], is compliant with Condition 3.

Proof: Follows exactly the same logic as the proof of compliance with Condition 3, given in Theorem 1 above □

In attempting to find a priority ordering that is schedulable according to the RTA-LC test, we would like to use the greedy assignment method of the OPA algorithm for as many of the lower priority levels as possible, without needing to backtrack and revise these priority assignments. The following theorem proves that we can do this, so long as the D-RTA-LC test continues to find a schedulable task at each priority level examined.

Theorem 4: For any sporadic taskset, where there exists a priority ordering *Q* that is schedulable according to the RTA-LC test, then assuming that the D-RTA-LC test is used in conjunction with the OPA algorithm to successfully assign schedulable tasks to priority levels from *n* to *k* generating a partial priority ordering *P*, then there also exists a complete priority ordering that is schedulable according to the RTA-LC test that also has the tasks at priority levels *n* to *k* in partial priority order *P*.

Proof: Let $Q(j)$ identify the task at priority *j* in priority order *Q*. Similarly, let $P(i)$ identify the task at priority *i* in priority order *P*.

We prove the theorem by iterating over increasing priority levels (values of *i*) from *n* to *k*. On each iteration, we transform priority ordering $Q = Q_n$ into $Q_{n-1} \ldots Q_{n-k}$. After each iteration, the new priority ordering remains schedulable according to the RTA-LC test, and once all of the iterations have been completed, then the tasks assigned to priority levels *n* to *k* are the same as those in partial priority order *P*.

Each iteration / transformation of the priority order Q_i, over values of *i* from *n* to *k*, works as follows: As the RTA-LC test is compliant with Condition 3 (Theorem 2), we repeatedly swap the priority of task $P(i)$ in priority order Q_i with the next lowest priority task until it reaches priority *i* (see Figure 2). This cannot cause any of the tasks of priority higher than *i* to become unschedulable according to the RTA-LC test (Theorem 3). Further, as task $P(i)$ is deemed schedulable at priority *i* by the D-RTA-LC test, then it follows that it is also guaranteed to be schedulable at that priority according to the RTA-LC test, independent of the priority ordering of those tasks with higher priorities, assuming only that they remain schedulable according to the RTA-LC test, which they do. Further, as the tasks of lower priority than *i* are schedulable according to the D-RTA-LC test, they are guaranteed to remain schedulable according to the RTA-LC test independent of any changes in the relative priority ordering of higher priority tasks.

After *n*-*k*+1 iterations, we have a priority ordering Q_{n-k} that is schedulable according to the RTA-LC test and has the tasks at priority levels *n* to *k* assigned according to the partial priority order *P* □

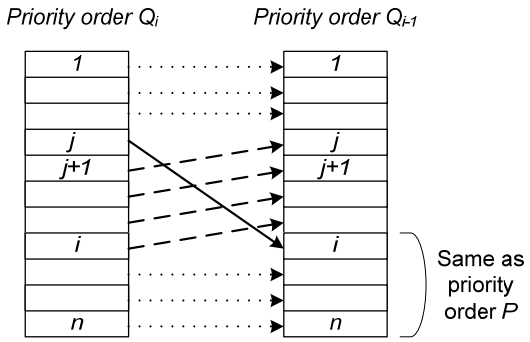

Figure 2: Priority re-ordering

Theorem 5: The *m* highest priority tasks are schedulable according to the RTA-LC test, with upper bound response times equal to their execution times, i.e. $R_k^{UB} = C_k$.

Proof: With the RTA-LC test, Equation (9) is used to compute R_k^{UB} for each task τ_k in priority order, highest priority first. The proof is by induction:

Consider the highest priority task *k*=1, it is subject to no interference and hence Equation (9) trivially converges on the initial value C_k.

Inductive step: Now consider each task in priority order from *k* = 2 to *m*. Every task τ_i with a higher priority than *k*, has $R_i^{UB} = C_i$, hence Equations (1) and (5) are equivalent, and so the I_i^{DIFF-R} term defined by Equation (8) is zero. Using Equation (9) to calculate R_k^{UB}, starting with C_k as an initial value therefore results in a value of *k*-1 for the first summation term and zero for the second. As *m* > *k*-1, the floor function evaluates to zero, and Equation (9) immediately converges on the initial value of C_k □

Theorem 6: The schedulability of any task τ_k ($k \geq m$) is, according to the RTA-LC test, independent of the relative priority ordering of the *m* highest priority tasks.

Proof: Follows from the fact that the *m* highest priority tasks have $R_i^{UB} = C_i$ (Theorem 5), independent of their relative priority order. Hence their interference on any lower priority task τ_k, given by Equations (8), (5) and (1), is independent of their relative priority order. (In fact Equations (1) and (5) are equivalent in this case, and so the $I_i^{DIFF-R}(R_k^{UB})$ term is zero for these tasks) □

Theorem 7: The D-RTA-LC test dominates the polynomial-

time schedulability test of Bertogna and Cirinei [14], referred to in [16] as the DA test.

Proof: We observe that the D-RTA-LC test effectively reduces to the DA test if we include all of the $I_i^{DIFF-R}(R_k^{UB})$ terms, and use an initial value of D_k. Hence, if task τ_k is schedulable according to the DA test, then the D-RTA-LC test is guaranteed to converge to a value of $R_k^{UB} \leq D_k$ □

Theorem 8: The RTA-LC test and the C-RTA condition are equivalent for the m+1 highest priority tasks.

Proof: Using the RTA-LC test, the m highest priority tasks have $R_i^{UB} = C_i$ (Theorem 5), hence for any task τ_k among the m+1 highest priority tasks, the interference term (floor function in Equation (9)) due to higher priority tasks is the same for both the RTA-LC test and the C-RTA condition. Thus the computed upper bound response time for task τ_k is the same in each case □

6. OPA-Backtracking Algorithm

In this section, we derive an optimal priority assignment algorithm that is compatible with the RTA-LC test. We refer to this algorithm as the OPA-Backtracking algorithm.

The basic intuition behind the OPA-Backtracking algorithm is to start at the lowest priority level, and use the D-RTA-LC test to assign tasks to each priority level, until a priority level k is reached where no tasks are schedulable according to that test. Assignment of tasks to priority levels n to k+1 is effectively permanent, since no backtracking is needed at these priority levels (Theorem 4). At all subsequent priority levels (k to 1), the algorithm uses the C-RTA condition to assign a potentially schedulable task to each priority level. The complete priority ordering is then checked using the RTA-LC test. If it is not schedulable, then the algorithm backtracks. It revisits the priority levels where the C-RTA condition was used, and assigns a different task that is also potentially schedulable at that priority level according to the C-RTA condition. Note that backtracking skips over priority levels 1 to m as these priorities are equivalent in terms of schedulability.

The detailed operation of the OPA-Backtracking algorithm is as follows, the line numbers refer to the pseudo-code[3] shown in Figure 3:

(1) It is assumed that the tasks are initially ordered according to some heuristic, for example DkC [16]. This determines the order in which unassigned tasks will be examined at each priority level. Each task is identified by an index value from 1 to n, corresponding to the initial heuristic priority ordering.

(2) The outer 'while' loop (lines 8-79) examines different priority orderings. Once a priority ordering is found that is schedulable according to the RTA-LC test (line 63), then the algorithm exits declaring the taskset schedulable, otherwise it continues to examine alternative priority orderings that could potentially be schedulable, only exiting and declaring the taskset unschedulable when all viable options have been exhausted, or some pragmatic limit on the maximum number of iterations has been reached (line 8).

(3) Within the outer 'while' loop, is a 'for' loop which attempts to construct a new priority ordering (lines 10-38). This loop begins by mimicking the operation of the OPA algorithm combined with the D-RTA-LC test (lines 12-21). Thus, starting at the lowest priority level, it assigns tasks that are schedulable according to the D-RTA-LC test, until a priority level is reached at which no unassigned tasks are found that are schedulable according to that test. Due to Theorem 4, the algorithm never needs to backtrack on these priority assignments. Alternatively, the D-RTA-LC condition may be successful in assigning a task to every priority level, in which case a schedulable priority ordering has been found and the algorithm exits (lines 60-61).

OPA-Backtracking Algorithm

```
0     numTries = 0;
1 Pass2: // label
2     startPri = n;
3     bCRTAUsed = false;
4     for(each priority i) {
5           CRTAIndx[i] = n;
6           CRTALevel[i] = false;
7     }
8     while(numTries < MAX_ITERATIONS) {
9           numTries++;
10          for(each priority i from startPri to 1) {
11                bSchedulable = false
12                if(!bCRTAUsed       // Use D-RTA
                        || (bHeuristic && !CRTALevel[i])) {
13                      for(each task t) {
14                            if(assigned t) continue
15                            bSchedulable = DRTATest(t, i);
16                            if(bSchedulable) {
17                                  AssignTask(t,i)
18                                  break
19                            }
20                      }
21                }
22                if(!bSchedulable) {   // Use C-RTA
23                      bCRTAUsed = true
24                      CRTALevel[i] = true
25                      for(each task t from CRTAIndx[i] to 1) {
26                            if(assigned t) continue
27                            bSchedulable = CRTATest(t,i)
28                            if(bSchedulable) {
29                                  AssignTask(t,i)
30                                  CRTAIndx[i] = t-1
31                                  break;
32                            }
33                      }
34                }
35                if(!bSchedulable) {
36                      break
37                }
38          }
39          if(!bSchedulable) {   //Incomplete priority assignment
40                if(CRTAIndx[i] == n) {
41                      return unschedulable
42                }
43                else {      // Backtrack if possible
44                      CRTAIndx[i] = n
                        CRTALevel[i] = false
45                      i++
46                      while((i <= n) && !CRTALevel[i]) {
```

[3] Detailed pseudo-code is provided to ensure that the algorithm and experiments can be replicated by other researchers.

```
47                          UnassignTask at priority i
48                          i++
49                    }
50                    if((i <= n) && CRTALevel[i]) {
51                          UnassignTask at priority i
52                          startPri = i
53                    }
54                    else {
55                          return unschedulable
56                    }
57              }
58        }
59        else {      //Complete priority assignment
60              if(!bCRTAUsed) {
61                    return schedulable
62              }
63              else if(RTA-LCTest()) {
64                    return schedulable
65              }
66              else {      //Backtrack to priority m+1
67                    for(priority i = 1 to m) {
68                          UnassignTask at priority i
69                          CRTAIndices[i] = n
                            CRTALevel[i] = false
70                    }
71                    while((i<=n) && !CRTALevel[i]) {
72                          UnassignTask at priority i
73                          i++
74                    }
75                    UnassignTask at priority i
76                    startPri = i;
77              }
78        }
79  }
80  return unschedulable
```

Figure 3: OPA-Backtracking algorithm

(4) Assuming that a priority level is reached where no unassigned tasks are schedulable according to the D-RTA-LC test, then the 'for' loop switches to using the C-RTA condition and again examines the suitability of unassigned tasks for assignment to the current priority level (lines 22-34).

(5) The C-RTA condition is used to identify the first unassigned task that is potentially schedulable at the current priority level. Note that checking begins at the highest task index value not yet tried at this priority level (line 25)). If a potentially schedulable task is found, then it is assigned to the current priority level, and the task's index – 1 recorded, to avoid repeating this priority ordering later (line 30). The algorithm then continues to the next higher priority level and so on, now using *only* the C-RTA condition (lines 23 and 12). Note for now we assume that the value of the Boolean variable 'bHeuristic' (line 12) is false.

(6) If a priority level is reached where the C-RTA condition is unable to identify a potentially schedulable unassigned task (that has not been examined before in combination with the current assignment of tasks to lower priority levels), then priority assignment cannot continue and so control breaks out of the 'for' loop (lines 35-37).

(7) On exit from the 'for' loop, there is either a complete and potentially schedulable priority ordering, or step (6) applied and the priority assignment is incomplete.

(8) If the priority assignment is incomplete (lines 39-58), then the algorithm first checks if there were any potentially schedulable tasks (at all) at the current priority level. If not, then no schedulable priority ordering exists according to the C-RTA condition and the algorithm exits declaring the taskset unschedulable (line 40). If there were some potentially schedulable tasks at the current priority level, then these have all been examined and so the algorithm attempts to backtrack (lines 43-57). The algorithm backtracks to the next lower priority level, if any, at which the C-RTA condition was used. All priority assignments down to and including that priority level are revoked, and the algorithm continues from there (lines 44-52, back to line 10), building up a new priority assignment. If no lower priority level is found at which the C-RTA condition was used, then no further backtracking is possible, as all viable alternatives have been tried, and the algorithm therefore exits, declaring the taskset unschedulable (line 55).

(9) If the priority assignment is complete (lines 59-78) then an immediate exit is possible if the D-RTA-LC test alone was sufficient to allocate all tasks to priority levels (line 61). Alternatively, the RTA-LC test is used to determine if the current priority ordering is in fact schedulable (line 63). If so, then the algorithm exits (line 64), if not then it backtracks (lines 66-77).

(10) Once a priority ordering has been examined using the RTA-LC test and found unschedulable, then Theorem 6 shows that all other priority orderings where only the relative priority ordering of the m highest priority tasks is changed are also unschedulable. The algorithm therefore skips all of these priority combinations by backtracking to priority level m+1 (lines 67-70). It then continues to backtrack to lower priority levels until a priority level is found where the C-RTA condition was used. Note that this must be the case, as the alternative is that the D-RTA-LC test succeeded in placing tasks at priorities n to m+1, in which case, the RTA-LC test is guaranteed to find the priority ordering schedulable. Note, the RTA-LC test is also guaranteed to find the taskset schedulable if there are m or fewer tasks.

6.1. Example of OPA-Backtracking operation

We now illustrate the operation of the OPA-Backtracking algorithm via a simple example. In the following discussion, 'step (x)' refers to steps (1) to (10) in the above description of the algorithm.

The example comprises a two processor system, with five tasks identified by indices 1-5, and five priority levels 1-5. Figure 4 (a) illustrates the first iteration of the OPA-backtracking algorithm. The cells highlighted in grey represent the first complete priority assignment examined. Note that '✓' indicates schedulability according to the D-RTA-LC test, while '**?**' and '**×**' indicate potentially schedulable and unschedulable respectively, according to the C-RTA condition.

The OPA-Backtracking algorithm operates as follows: First the D-RTA-LC test identifies that task 5 is schedulable

at priority level 5. This assignment is permanent (see step (3)). However, at priority level 4, the D-RTA-LC test finds that no unassigned tasks are schedulable. Hence the algorithm switches to using the C-RTA condition (step (4)). The C-RTA condition finds that task 4 is potentially schedulable at priority level 4 and assigns it, setting the index for priority level 4 to 3, so that this assignment will not be revisited (step (5)). Priority level 3 is then examined, task 3 assigned priority 3, and the index for priority level 3 set to 2, and so on for priority levels 2 and 1. At this point, a complete priority assignment has been made (Figure 4 (a)), and the RTA-LC test is therefore used to determine if the assignment is schedulable (step (9)). We will assume that it is not.

The algorithm then backtracks to priority level m+1 = 3 (step (10)) and continues its next iteration from that priority level. At priority level 3, there are no potentially schedulable tasks identified by the C-RTA condition that have not already been tried (only tasks 2 and 1 are checked as the index for priority level 3 is 2). The algorithm therefore backtracks further to priority level 4 (via steps (6) and (8)), in the process resetting the index for priority level 3 to the value 5. At priority level 4, task 3 is now examined and found to be potentially schedulable according to the C-RTA condition. Task 3 is therefore assigned priority 4 and the index for priority level 4 is set to 2, so that this assignment will not be revisited (step (5)). The algorithm then proceeds by assigning tasks 4, 2, and 1 to priority levels 3, 2, and 1 respectively (See Figure 4 (b)). At this point, a complete priority assignment has again been made, so the RTA-LC test is used to determine if it is schedulable (step (6)). Again we will assume that it is not.

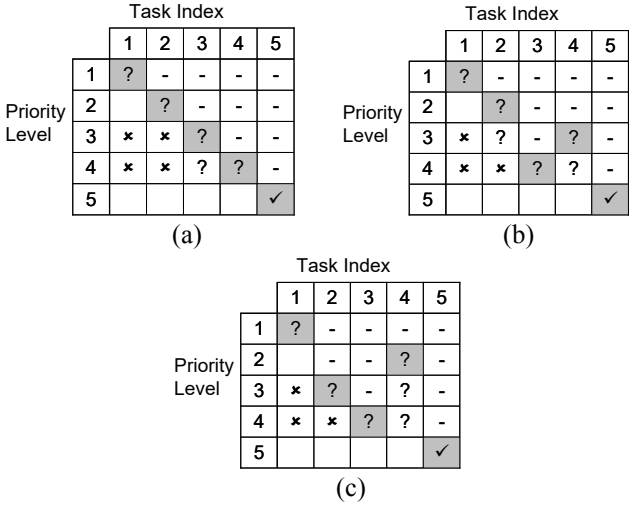

Figure 4: Priority assignments

The algorithm again backtracks to priority level 3 (step (10)); however, this time task 2 is potentially schedulable according to the C-RTA condition, so it is assigned priority 3, with tasks 4, and 1 subsequently assigned priorities 2, and 1 respectively (See Figure 4 (c)). This is the final complete priority assignment that the algorithm will examine. We will assume that this priority ordering is schedulable according to the RTA-LC test and so the algorithm is successful.

If the priority ordering shown in Figure 4 (c) was not schedulable, then the algorithm would backtrack again looking for a priority level at which there were remaining tasks, with indices less than or equal to those currently set for the priority level, that are schedulable according to the C-RTA condition. As there are now none at priority level 3, and none at priority level 4, the algorithm would exit declaring the taskset unschedulable (step (8)).

Note in this example, there were $n!$ = 120 possible priority orderings; however, the OPA-Backtracking algorithm only had to examine three complete priority orderings to find a schedulable ordering. The algorithm used three techniques to prune away unschedulable priority orderings without checking them:

1. Theorem 6 shows that the relative priority ordering of the highest m priority tasks is unimportant This rule reduces the number of distinct priority orderings that need to be examined to $n!/m!$ = 60, eliminating 60 alternative priority orderings.
2. The D-RTA-LC test enabled task 5 to be assigned priority 5. Theorem 4 shows that if any schedulable ordering exists according to the RTA-LC test, then a schedulable ordering will exist with this priority assignment. Discounting other tasks from consideration at priority level 5 eliminated a further 48 alternatives.
3. Finally, the C-RTA condition removed two possibilities (tasks 1 and 2) from consideration at priority level 4, eliminating 6 alternative priority orderings. Then, with task 4 at priority 4, it also removed tasks 1 and 2 from consideration at priority level 3, whereas, with task 3 at priority 4, it removed task 1 from consideration at priority level 3. In total, the C-RTA condition removed a further 9 alternative priority orderings, leaving just 3 which were examined using the RTA-LC test.

6.2. Heuristic OPA-Backtracking

The D-RTA-LC test and the C-RTA condition typically result in significant pruning of the number of priority ordering combinations examined by the OPA-Backtracking algorithm. Nevertheless, the number of priority orderings that remain to be explored can be excessive. The main reason for this is that once a priority level k is reached at which there is no schedulable task according to the D-RTA-LC test, then in order to ensure optimality, all combinations of assignments of the remaining tasks to priority levels k to 1, that are potentially schedulable according to the C-RTA condition, need to be explored. (With the exception of those varying only in the relative priority ordering of the highest m priority tasks). The reason that all these possibilities need to be explored to ensure optimality is that once a task is placed according to the C-RTA condition, its schedulability according to the RTA-LC test is dependent on the relative priority ordering of higher priority tasks, and so these relative priority orderings must be explored.

In pathological cases, the D-RTA-LC test may fail to find a schedulable task at priority n, and yet all of the tasks may be potentially schedulable at all priority levels according to the C-RTA condition. In this case, the OPA-Backtracking algorithm can attempt to explore $n!/m!$ priority

orderings. With 8 processors and 40 tasks, this equates to $>10^{43}$ priority ordering combinations. It is therefore essential to set a pragmatic limit on the number of iterations of the algorithm. Given a finite iteration limit ($<< n!/m!$), the OPA-Backtracking algorithm is no longer guaranteed to find a schedulable priority ordering if one exists according to the RTA-LC test. In this case, the order in which candidate priority orderings are examined has an influence on the overall effectiveness of the algorithm.

With the aim of finding schedulable priority orderings quickly, we can form a heuristic version of the OPA-Backtracking algorithm by setting the value of the Boolean variable 'bHeuristic', tested on line 12 of Figure 3, to true. The effect of this is to first employ the D-RTA-LC test at each priority level, even if the C-RTA condition has been used to assign a potentially schedulable task to a lower priority level. Further, if a task is found to be schedulable according to the D-RTA-LC test, then it is the only task tried at that priority level in conjunction with the current assignment of tasks to lower priority levels. Hence backtracking effectively only takes place over priority levels where the C-RTA condition has been employed.

This approach is not optimal as it does not necessarily consider all viable priority orderings above the priority level k at which the C-RTA condition is first employed. Thus the heuristic algorithm may fail to examine the particular relative priority ordering of tasks at priorities k+1 to 1 needed to make the task assigned to priority k schedulable.

The heuristic version of the OPA-Backtracking algorithm uses the D-RTA-LC test to greedily assign tasks that are themselves guaranteed to be schedulable at a given priority level. Thus, it tends to build up priority orderings where most of the tasks are known to be schedulable, assigned by the D-RTA-LC test, and only a few are potentially schedulable, assigned by the C-RTA condition. Further, once a particular priority ordering is found to be unschedulable, the heuristic algorithm varies its choice of potentially schedulable tasks, rather than varying the priority order of tasks at higher priorities to try and make a potentially schedulable task schedulable. Intuitively, this would seem to be a more effective way of finding schedulable priority orderings quickly.

The heuristic OPA-Backtracking algorithm is similar in its approach to the priority ordering algorithm for real-time wormhole communication given by Zheng and Burns in [26]. We note that for the same basic reasons, the algorithm given in [26] is also heuristic rather than optimal.

7. Empirical investigation

In this section, we present the results of an empirical investigation, examining the effectiveness of the RTA-LC test for global FP scheduling when combined with the OPA-Backtracking algorithm. We examined the performance of three variants of the algorithm, (i) the standard OPA-Backtracking approach, (labelled OPA-Bk), (ii) the heuristic approach (labelled OPA-heuristic), and (iii) a two pass approach (labelled OPA-2Pass), which first uses the heuristic algorithm (bHeuristic = true), and then if that fails to find a schedulable priority ordering, makes a second pass using the standard OPA-Backtracking approach (bHeuristic = false). In each case, the total number of iterations per taskset was limited to 1000. This relatively low limit was used as our experiments needed to explore 1000's of tasksets. In examining the schedulability of a single taskset, a much higher limit could be used. The order of task indices was set according to the DkC heuristic [16].

For comparison purposes, we also provide results for: the DA-LC and D-RTA-LC tests with optimal priority assignment, labelled DA-LC(OPA) and D-RTA-LC(OPA) respectively; and the C-RTA condition with optimal priority assignment, labelled C-RTA(OPA). The results for the DA-LC and D-RTA-LC tests were almost identical, with the D-RTA-LC test able to schedule only a few additional tasksets at each utilisation level.

We note that D-RTA-LC(OPA) lower bounds the performance of RTA-LC (OPA-Bk), while C-RTA(OPA) upper bounds it. Recall that the C-RTA condition is *not* a schedulability test. Instead, it is a necessary condition for task schedulability under the RTA-LC test. Hence in the graphs below, the line for the C-RTA condition indicates only potentially schedulable tasksets.

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

- Task utilisations were generated using the UUnifast-Discard algorithm [16], using a discard limit of 1000.
- Task periods were generated according to a log-uniform distribution with a factor of 1000 difference between the minimum and maximum possible task period. This represents a spread of task periods from 1ms to 1 second, as found in most hard real-time applications.
- Task execution times were set based on the utilisation and period selected: $C_i = U_i T_i$.
- Task deadlines were assigned according to a uniform random distribution, in the range $[C_i, T_i]$.

7.1. Experimental results

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 each combination of priority assignment policy and schedulability test / condition.

The graphs plot the percentage of tasksets generated that were deemed schedulable in each case. Figures 5 to 8 show this data for 2, 4, 8, and 16 processors. In each case, the number of tasks was set to 5 times the number of processors. Note the differing x-axis scale on the graphs.

From the graphs, we can see that the D-RTA-LC (OPA) lower bound and the C-RTA (OPA) upper bound tightly envelop the performance of RTA-LC test with optimal priority assignment. We observe that the results for the three variants of the backtracking algorithm were very similar for small numbers of processors / tasks; with differences becoming apparent as the number of processors / tasks increased. As the number of processors / tasks was increased, the OPA-heuristic and OPA-2Pass approaches become more effective than the OPA-Bk approach at

identifying schedulable priority orderings.

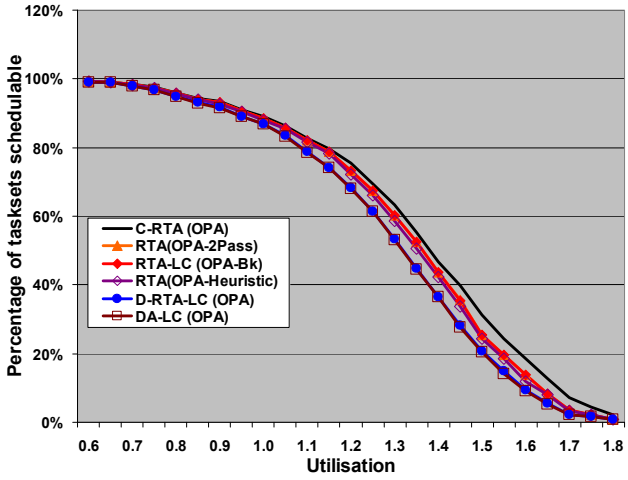

Figure 5: (2 processors, 10 tasks, D≤T)

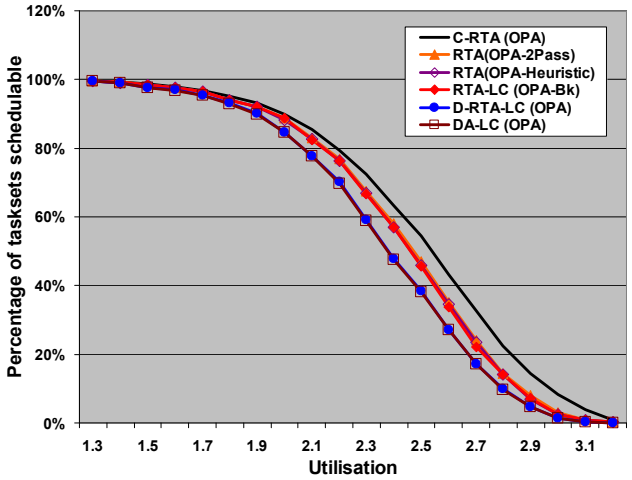

Figure 6: (4 processors, 20 tasks, D≤T)

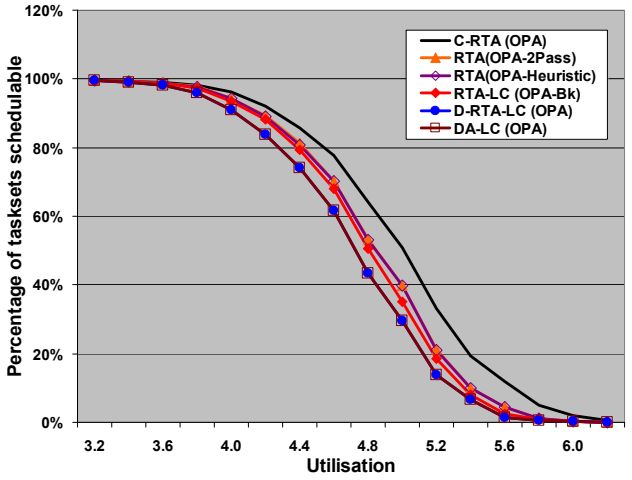

Figure 7: (8 processors, 40 tasks, D≤T)

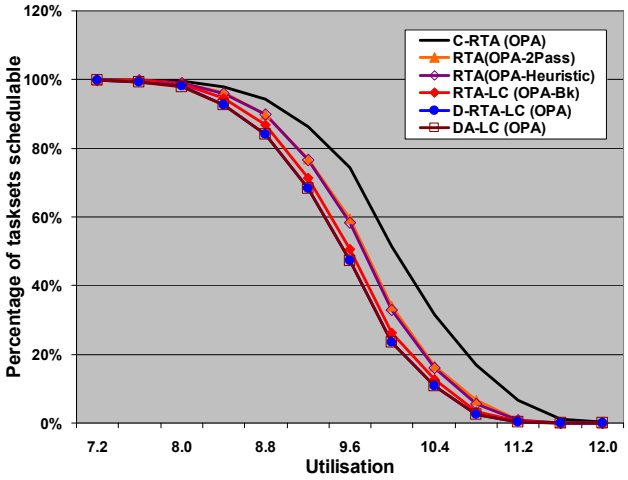

Figure 8: (16 processors, 80 tasks, D≤T)

Table 1 below shows the total number of tasksets found to be schedulable by each of the algorithms studied for 2, 4, 8, and 16 processors. The first row of data, marked OPA (DA-LC), summarises the results for the polynomial-time schedulability test, DA-LC, combined with Audsley's OPA algorithm. The second, third, and fourth rows of data give the results for the three variants of the OPA-Backtracking approach.

Table 1: Number of tasksets found to be schedulable

	#Processors			
Algorithm	**2**	**4**	**8**	**16**
DA-LC (OPA)	24,278	23,085	22,989	23,270
RTA-LC(OPA-Bk)	25,096	23,748	23,393	23,444
RTA-LC(OPA-heuristic)	24,925	23,768	23,593	23,747
RTA-LC(OPA-2Pass)	25,099	23,846	23,615	23,795

We conclude that for larger numbers of processors / tasks, the OPA-2Pass approach (using the heuristic algorithm first) is more effective than using either the standard backtracking algorithm or the heuristic algorithm alone. In each case, the OPA-2Pass approach found approximately 3% more schedulable tasksets than the polynomial-time DA-LC test combined with Optimal Priority Assignment.

8. Summary and conclusions

The motivation for this research comes from the fact that priority assignment is of fundamental importance to the effectiveness of global FP scheduling [16], and yet prior to this work, arguably the most effective schedulability test for global FP scheduling, the response time analysis of Bertogna and Cirinei [12], improved by Guan et al. [22] (RTA-LC test), could only be used with heuristic priority assignment policies. This was due to its incompatibility with Audsley's Optimal Priority Assignment algorithm [16].

The key contribution of this paper is in providing an optimal priority assignment algorithm (OPA-Backtracking) that is compatible with the RTA-LC test. This algorithm eliminates large numbers of priority ordering combinations

from consideration via utilising: (i) The D-RTA-LC test which lower bounds the RTA-LC test; (ii) the C-RTA condition which upper bounds the RTA-LC test; (iii) the fact that the relative priority ordering of the *m* highest priority tasks is unimportant.

The RTA-LC test combined with OPA-Backtracking dominates the D-RTA-LC test combined with OPA. It also dominates the DA test (Deadline Analysis test of Bertogna and Cirinei [14]) combined with OPA, which was previously shown to be the best performing approach in [16]. Further, these dominance results hold even if the OPA-Backtracking algorithm is only permitted a single iteration (i.e. no backtracking). This is because if a taskset is schedulable according to the D-RTA-LC test combined with OPA, then it will be found schedulable on the first iteration of the OPA-Backtracking algorithm. As the D-RTA-LC test dominates the DA test, then the same applies to any taskset that is schedulable according to the DA test with OPA.

Given an unlimited number of iterations, the OPA-Backtracking algorithm is guaranteed to find a priority ordering that is schedulable according to the RTA-LC test if one exists. However, in pathological cases it may need to explore up to $n!/m!$ priority orderings, hence in practice a limit needs to be set, either in terms of the number of iterations, or the execution time of the algorithm, once this limit is reached, then the taskset is declared unschedulable.

Given that a pragmatic limit is placed on the number of iterations, then the order in which priority orderings are examined has an influence on the effectiveness of the algorithm. In this respect, we found that first using a heuristic version of the backtracking algorithm that greedily employs the D-RTA-LC test, and then switching to the standard approach, improved performance particularly in cases with a larger number of processors and tasks.

We note that the backtracking algorithms described in this paper typically require a large number of iterations of the RTA-LC schedulability test, which itself is a pseudo-polynomial time algorithm. Hence the backtracking approach to priority assignment is only appropriate for use in an off-line, as opposed to an on-line, context.

The main result of this paper is to provide an indication of the maximum possible performance that can be obtained from the state-of-the-art RTA-LC schedulability test via appropriate priority assignment. The paper achieves this in two ways. Firstly, it derives the OPA-Backtracking algorithm that can be used to search for a schedulable priority assignment in a way that is more effective than exhaustive search. Secondly it provides an upper bound on the maximum possible performance that could ever be obtained from the RTA-LC test. That bound is given by combining the C-RTA condition with Audsley's OPA algorithm.

8.1. Acknowledgements

This work was funded in part by the EPSRC Tempo project (EP/G055548/1) and the Artist Design Network of Excellence.

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, S. Baruah, J. Jonsson, "Static-priority scheduling on multiprocessors". In Proc. RTSS, pp. 193–202, 2001.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] S.K. Baruah, "Techniques for Multiprocessor Global Schedulability Analysis". In proc. RTSS, pp. 119-128, 2007.

[10] S.K. Baruah, N. Fisher. "Global Fixed-Priority Scheduling of Arbitrary-Deadline Sporadic Task Systems" In Proc. International Conference on Distributed Computing and Networking, pp. 215-226, Jan 2008.

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

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

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

[14] 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.

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

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

[17] 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-440-2009, May 2009.

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

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

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

[21] George, L., Rivierre, N., Spuri, M., "Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling", INRIA Research Report, No. 2966, September 1996.

[22] N. Guan, M. Stigge, W.Yi, G. Yu, "New Response Time Bounds for Fixed Priority Multiprocessor Scheduling". In Proc. RTSS, pp. 388-397 2009.

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

[24] 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.

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

[26] S. Zheng, A. Burns, “Priority Assignment for Real-Time Wormhole Communication in On-Chip Networks”. In Proc. RTSS, pp. 421-430, 2008.