Global Fixed Priority Scheduling with Deferred Pre-emption

R.I. Davis, A. Burns
University of York, UK.
{rob.davis, alan.burns}@york.ac.uk

J. Marinho, V. Nelis, S.M. Petters
CISTER/INESC-TEC, ISEP, Porto, Portugal.
{jmsm, nelis, smp}@isep.ipp.pt

M. Bertogna
University of Modena, Italy.
marko.bertogna@unimore.it

Abstract— This paper introduces schedulability analysis for global fixed priority scheduling with deferred pre-emption (gFPDS) for homogeneous multiprocessor systems. gFPDS is a superset of global fixed priority pre-emptive scheduling (gFPPS) and global fixed priority non-pre-emptive scheduling (gFPNS). We show how schedulability can be improved via appropriate choice of priority assignment and final non-pre-emptive region lengths, and we provide algorithms which optimize schedulability in this way. An experimental evaluation shows that gFPDS significantly outperforms both gFPPS and gFPNS.

I. INTRODUCTION

A common misconception with regard to fixed priority scheduling of sporadic tasks is that fully pre-emptive scheduling is more effective in terms of schedulability than non-pre-emptive scheduling. The two are however incomparable; there are tasksets that are schedulable under fixed priority non-pre-emptive scheduling that are not schedulable under fixed priority pre-emptive scheduling and vice-versa. This is the case for uniprocessor scheduling [27] and also the case for global multiprocessor scheduling [30], which is the focus of this paper.

While the blocking effect, due to long non-pre-emptive regions of low priority tasks degrades schedulability for single processor systems that have a wide range of task execution times and periods (as illustrated by Figure 7 in [27]), Guan et al. [30] showed that the same is not necessarily true for multiprocessor systems. With m processors rather than one, long non-pre-emptive regions can be accommodated without necessarily compromising the schedulability of higher priority tasks. However, this advantage only extends so far; with m processors then m long non-pre-emptive regions are enough to significantly compromise schedulability. In this context, limited non-pre-emptive execution has the advantage of reducing the number of pre-emptions, and potentially improving the worst-case response time of tasks, while also keeping blocking effects on higher priority tasks within tolerable limits.

In the literature, the term *fixed priority scheduling with deferred pre-emption* has been used to refer to a variety of different techniques by which pre-emptions may be deferred for some interval of time after a higher priority task becomes ready. These are described in a survey by Buttazzo et al. [21] and briefly discussed in Section II. In this paper, we assume a simple form of fixed priority scheduling with deferred pre-emption where each task has a single non-pre-emptive region at the end of its execution. If this region is of the minimum possible length for all tasks, then we have fully pre-emptive scheduling, whereas if it constitutes all of the task's execution time then we have non-pre-emptive scheduling.

In this paper, we introduce sufficient schedulability tests for global fixed priority scheduling with deferred pre-emption (gFPDS). gFPDS can be viewed as a superset of both global fixed priority pre-emptive scheduling (gFPPS) and global fixed priority non-pre-emptive scheduling (gFPNS) and strictly dominates both. With gFPDS, there are two key parameters that affect schedulability: the priority assigned to each task, and the length of each task's final non-pre-emptive region (FNR). The FNR length affects both the schedulability of the task itself, and the schedulability of tasks with higher priorities. This is a trade-off as increasing the FNR length can improve schedulability for the task itself by reducing the number of times it can be pre-empted, but potentially increases the blocking effect on higher priority tasks which may reduce their schedulability.

In 2012, Davis and Bertogna [27] introduced an optimal algorithm for fixed priority scheduling with deferred pre-emption on a single processor. This algorithm finds a schedulable priority assignment and set of FNR lengths whenever such a schedulable combination exists. In this paper we also build upon this work, extending it to the multiprocessor case. For a given priority ordering, we show how to find an assignment of FNR lengths that result in a system that is deemed schedulable under gFPDS according to our sufficient schedulability tests, whenever such an assignment exists of FNR lengths exists. We also show that the *Final Non-pre-emptive Region and Priority Assignment* (*FNR-PA*) algorithm from [27] is *not optimal* in the multiprocessor case, but nevertheless can be used as a heuristic for determining both priority ordering and final non-pre-emptive region lengths.

II. BACKGROUND RESEARCH

A. Deferred pre-emption

Two different models of fixed priority scheduling with deferred pre-emption have been developed in the literature.

In the *fixed* model, introduced by Burns in 1994 [19], the location of each non-pre-emptive region is statically determined prior to execution. Pre-emption is only permitted at pre-defined locations in the code of each task, referred to as *pre-emption points*. This method is also referred to as *co-operative scheduling*, as tasks co-operate, providing re-scheduling / pre-emption points to improve schedulability.

In the *floating* model [7], [32], an upper bound is given on the length of the longest non-pre-emptive region of each task. However, the location of each non-pre-emptive region is not known a priori and may vary at run-time, for example under the control of the operating system.

For uniprocessor systems: Exact schedulability analysis for the fixed model was derived by Bril et al. in 2009 [18]. Subsequently, Bertogna et al. integrated pre-emption costs and cache related pre-emption delays (CRPD) into analysis of the fixed model, considering both fixed [14] and variable

[15] pre-emption costs. In 2011, Bertogna et al. [16] derived a method for computing the optimal FNR length of each task in order to maximize schedulability assuming a given priority assignment. In 2012, Davis and Bertogna [27] introduced an optimal algorithm that is able to find a schedulable combination of priority assignment and FNR lengths whenever such a schedulable combination exists.

B. Global fixed priority scheduling

In 2003, Baker [5] developed a strategy that underpins an extensive thread of subsequent research into schedulability tests for gFPPS [9], [11], [12], [13], [29], [31], and gFPNS [30]. (For a comprehensive survey of multiprocessor real-time scheduling, the reader is referred to [26]). Baker's work was subsequently built upon by Bertogna et al. [11] [13]. They developed sufficient schedulability tests for gFPPS 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. [31] extended this approach using ideas from [8] to limit the amount of carry-in interference.

In 2009 and 2010, Davis and Burns [22], [23] showed that priority assignment is fundamental to the effectiveness of gFPPS. They proved that Audsley's optimal priority assignment algorithm [3], [4] is applicable to some of the sufficient tests developed for gFPPS, including the deadline-based test of Bertogna et al. [13], but not to others such as the later response time tests [12], [31].

In 2011, Guan et al. [30] provided schedulability analysis for gFPNS based on the approach of Baker [5], and the techniques introduced by Bertogna et al. in [11].

gFPDS is broadly similar to the dynamic algorithm FPZL [24], [25]. FPZL resembles gFPPS until a job reaches a state of zero laxity i.e. when its remaining execution time is equal to the elapsed time to its deadline. FPZL gives such a job the highest priority, and hence makes it non-pre-emptable. The length of time each job spends executing in this zero-laxity state is determined dynamically by FPZL. With FPZL, RTOS support for this dynamic behaviour is required, whereas with gFPDS the transition to non-pre-emptive execution may be controlled either by the RTOS, or via API calls suitably located within the code of each task.

III. System model, Terminology and Notation

In this paper, we are interested in global fixed priority scheduling of an application on a homogeneous multiprocessor system with m identical processors. The application or taskset is assumed to consist of a static set of n tasks ($\tau_1 ... \tau_n$), with each task τ_i assigned a unique priority i, from 1 to n (where n is the lowest priority). We assume a discrete time model, where all task parameters are positive integers (e.g. processor clock cycles). We use the notation $hp(i)$ (and $lp(i)$) to mean the set of tasks with priorities higher than (lower than) i.

Tasks are assumed to comply with the *sporadic* task model. In this model, each task gives rise to a potentially unbounded sequence of jobs. Each job 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 ($C_i \le D_i$), and minimum inter-arrival time or *period* T_i. It is assumed that all tasks have constrained deadlines ($D_i \le T_i$). The *utilisation* U_i of each task is given by C_i / T_i. Under gFPDS, each task is assumed to have a final non-pre-emptive region of length F_i in the range $[1, C_i]$ (Here, the minimum value is 1 rather than 0 as a task can only be pre-empted at discrete times corresponding to processor clock cycles). Finding an appropriate FNR length for each task is assumed to be part of the scheduling problem.

The *worst-case response time* R_i of a task is the longest possible time from the release of the task until it completes execution. Thus task τ_i is schedulable if and only if $R_i \le D_i$ and a taskset is schedulable if and only if $\forall i \quad R_i \le D_i$. We use R_i^{UB} to indicate an upper bound on the worst-case response time of task τ_i.

Under gFPDS, at any given time, the m ready tasks with the highest priorities are selected for execution. Final non-pre-emptive regions are assumed to be implemented by manipulating task priorities, thus a task executing its FNR has the highest priority and will not be pre-empted.

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 job starts to execute it will not voluntarily suspend itself.

Job 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 are assumed to be either negligible, or subsumed within the worst-case execution time of each task. (Pre-emption costs are an issue we aim to address in future work).

A taskset is said to be *schedulable* with respect to some scheduling algorithm, if all valid sequences of jobs that may be generated by the taskset can be scheduled by the algorithm without any missed deadlines.

A priority assignment policy P is said to be *optimal* with respect to a schedulability test for some type of fixed priority scheduling algorithm (e.g. gFPPS, gFPNS, or gFPDS) if there are no tasksets that are deemed schedulable, according to the test, under the scheduling algorithm using any other priority ordering policy, that are not also deemed schedulable with the priority assignment determined by policy P.

IV. Schedulability Analysis for gFPDS

In this section, we introduce sufficient schedulability tests for global fixed priority scheduling with deferred pre-emption (gFPDS).

On a uniprocessor, under fixed priority scheduling with deferred pre-emption, a higher priority task can only be blocked by a single job of a lower priority task that starts

executing non-pre-emptively prior to the release of the higher priority task. The multiprocessor case is however significantly different. This is illustrated by Figure 1 below, for the case of 4 processors. Here, the task of interest τ_k (priority 2) is released at time t=1, along with a job of the higher priority task τ_1. τ_k is unable to execute initially due to blocking from three jobs of lower priority tasks (τ_3, τ_4, and τ_5) that have entered their FNRs (shown in dark grey in Figure 1). At time t=4, τ_k begins executing. At t = 7, three further jobs of lower priority tasks (τ_6, τ_7, and τ_5 again) enter their FNRs. At t=8, task τ_k is pre-empted by a second job of τ_1 and misses its deadline at t=12.

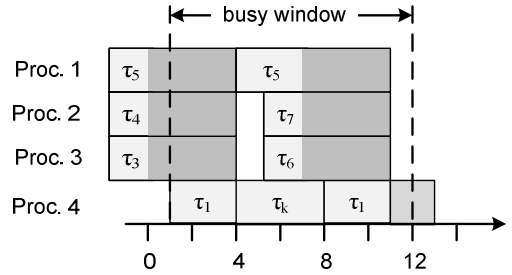

Figure 1: Blocking effect due to FNRs of lower priority jobs.

This example serves to illustrate the following:

- Multiple lower priority tasks may contribute interference in the busy window of the task of interest. Further, the number of lower priority tasks that may contribute is not limited to m as it is in the non-pre-emptive case [30].
- Multiple jobs of the same lower priority task may contribute interference, due to the fact that the task of interest does not occupy all of the processors when it executes; unlike in the uniprocessor case.
- If there were multiple non-pre-emptive regions within each lower priority task, then each of these regions could potentially contribute interference. (This is easy to see by assuming that all of the execution of task τ_5 on processor 1 belongs to one job rather than two).

While no worst-case scenario is currently known, we can obtain an upper bound on the interference from the non-pre-emptive execution of lower priority tasks, by modelling this non-pre-emptive execution as a set of virtual tasks executing at the highest priority. Thus for each lower priority task $\tau_i \in lp(k)$, we assume a virtual task τ_{iv} with the following parameters: $C_{iv} = F_i - 1$, $T_{iv} = T_i$, $D_{iv} = D_i$, $R_{iv}^{UB} = R_i^{UB}$ and the highest priority. (We note that $C_{iv} = F_i - 1$ as the task must have actually entered its FNR in order to be non-pre-emptable).

We note the following points about gFPDS:

1. Once a task τ_k enters its FNR it will execute to completion. Hence with gFPDS if we can show that the task is guaranteed to execute for $C_k^* = C_k - (F_k - 1)$ within an effective deadline of $D_k^* = D_k - (F_k - 1)$, then it is guaranteed to execute for C_k by its deadline D_k.
2. In the worst-case scenario, at most $m - 1$ higher priority tasks can have carry-in jobs (Theorem 1 of [23]).
3. Virtual tasks representing the FNRs of lower priority tasks can effectively be released at any point during the interval in which the corresponding lower priority task may execute, hence the argument of Theorem 1 in [23] relating to a maximum of $m - 1$ carry-in jobs does not apply to virtual tasks.

A. *Deadline Analysis for gFPDS*

We now extend and adapt the deadline-based, schedulability test of Bertogna et al. (Theorem 8 in [13]) to gFPDS. Under gFPPS, if task τ_k is schedulable in an interval of length L, with an execution time of C, then an upper bound on the interference over the interval due to a higher priority task τ_i with a *carry-in job*[1] is given by the following equation [13].

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

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

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

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

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

Making use of D_k^* and C_k^* to account for the fact that task τ_k is schedulable under gFPDS if it is able to start its FNR by D_k^* results in the following schedulability test:

Deadline Analysis (DA) test for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, inequality (4) holds:*

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

where $lpv(k)$ is the set of virtual tasks used to model the non-pre-emptive execution of tasks in $lp(k)$. (Note the floor function comes from the use of integer values for all task parameters).

We now improve the DA test using the approach of Guan et al. [31]. They showed that for gFPPS, an upper bound on the interference over an interval L due to a higher priority task τ_i *without* a carry in job is given by:

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

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 interference terms (1) and (5) is:

$$I_i^{DIFF-D}(L,C) = I_i^D(L,C) - I_i^{NC}(L,C) \quad (8)$$

Davis and Burns [23] showed that the worst-case scenario for gFPPS occurs when there are at most m-1 carry-in jobs. Thus an improved test for gFPDS is as follows:

[1] Here, a carry-in job is defined as a job that is released strictly prior to the start of the interval, and causes interference within that interval.

Deadline Analysis – Limited Carry-in (DA-LC test) for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, inequality (9) holds:*

$$D_k^* \geq C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(D_k^*, C_k^*) + \sum_{i \in MD(k,m-1)} I_i^{DIFF-D}(D_k^*, C_k^*) + \sum_{\forall j \in lpv(k)} I_j^{D}(D_k^*, C_k^*) \right) \right\rfloor \quad (9)$$

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

B. Response Time Analysis for gFPDS

We now extend and adapt the response time test of Bertogna and Cirinei [12] to gFPDS. They showed that under gFPPS, if task τ_k is schedulable in an interval of length L, completing an execution time C, then an upper bound on the interference in that interval due to a higher priority task τ_i with a carry-in job is given by:

$$I_i^R(L, C) = \min(W_i^R(L), L - C + 1) \quad (10)$$

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

$$W_i^R(L) = N_i^R(L) C_i + \min(C_i, L + R_i^{UB} - C_i - N_i^R(L) T_i) \quad (11)$$

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

$$N_i^R(L) = \left\lfloor \frac{L + R_i^{UB} - C_i}{T_i} \right\rfloor \quad (12)$$

Making use of D_k^* and C_k^* to account for the fact that task τ_k is schedulable under gFPDS if it is able to start its FNR by D_k^* results in the following schedulability test. (Note, we return later to the order in which upper bound response times are computed, which is resolved by Algorithm 1).

Response Time Analysis (RTA) test for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, the upper bound response time R_k^S for the start (first unit of execution) of the task's FNR, computed via the fixed point iteration given by (13) within Algorithm 1, is less than or equal to the task's effective deadline D_k^*:*

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^R(R_k^S, C_k^*) + \sum_{\forall i \in lpv(k)} I_i^R(R_k^S, C_k^*) \right) \right\rfloor \quad (13)$$

In (13), the second summation term models the blocking effect from lower priority tasks via the set of virtual tasks. If task τ_k is schedulable, then $R_k^{UB} = R_k^S + (F_k - 1)$.

We now improve the RTA test using the approach of Guan et al. [31]. They showed that under gFPPS, if a higher priority task τ_i does not have a carry-in job, then the interference term is given by (5) rather than (10). The difference between the two interference terms is:

$$I_i^{DIFF-R}(L, C) = I_i^R(L, C) - I_i^{NC}(L, C) \quad (14)$$

Further, at most m-1 higher priority tasks with carry-in jobs may contribute interference in the worst-case. Thus an improved test for gFPDS is as follows:

Response Time Analysis – Limited Carry-in (RTA-LC) test for gFPDS: *A sporadic taskset is schedulable, if for every task τ_k, the upper bound response time R_k^S for the start (first unit of execution) of the task's FNR, computed via the fixed point iteration given by (15) within Algorithm 1, is less than or equal to the task's effective deadline D_k^*:*

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(R_k^S, C_k^*) + \sum_{i \in MR(k,m-1)} I_i^{DIFF-R}(R_k^S, C_k^*) + \sum_{\forall j \in lpv(k)} I_j^{R}(R_k^S, C_k^*) \right) \right\rfloor \quad (15)$$

where $MR(k, m\text{-}1)$ is the subset of at most m-1 tasks with the largest values of $I_i^{DIFF-R}(R_k^{UB}, C_k)$, given by (14), from the set of tasks $hp(k)$. If task τ_k is schedulable, then $R_k^{UB} = R_k^S + (F_k - 1)$.

```
1   Initialize all R_i^UB = C_i
2   repeat = true
3   while (repeat) {
4       repeat = false
5       for (each priority level k, highest first) {
6           Calc. R_k^UB via RTA or RTA-LC test for gFPDS
7           if ( R_k^UB > D_k ) {
8               Return unschedulable
9           }
10          if ( R_k^UB differs from its previous value) {
11              repeat = true
12          }
13      }
14  }
15  return schedulable
```

Algorithm 1: Response time iteration

We note that in adapting the methods of Bertogna and Cirinei [12] and Guan et al. [31] to gFPDS there is a difficulty in accounting for the interference from virtual tasks. When computing the upper bound response time for task τ_k the upper bound response times of each higher priority task are required. This can easily be achieved for the set of tasks $hp(k)$ simply by computing response times in order, highest priority first, which is all that is needed for gFPPS. However, when considering gFPDS we also include interference from virtual tasks corresponding to tasks in $lp(k)$. Here, the upper bound response time R_{iv}^{UB} for each virtual task equates to that of its corresponding (lower priority) task $R_{iv}^{UB} = R_i^{UB}$, which itself depends on the upper bound response time of task τ_k, leading to an apparent circularity. This issue can be solved by noting that the upper bound response time R_{iv}^{UB} of each virtual task is monotonically non-decreasing with respect to increases in the upper bound response times of all tasks in $hp(i)$, and the upper bound response time R_k^{UB} of each task τ_k is monotonically non-decreasing with respect to increases in the upper bound response times of all virtual tasks

associated with tasks in $lp(k)$. Thus we can employ a fixed point iteration to solve for all upper bound response times starting with values that are guaranteed to be no larger than any possible solution, for example $R_i^{UB} = C_i$. The pseudo code in Algorithm 1 implements this approach.

C. Complexity and comparability

The DA and DA-LC tests for gFPDS are polynomial in complexity: $O(n^2)$ for a taskset of cardinality n. (Note, the (m-1) largest I_i^{DIFF} terms may be obtained by *linear-time selection* [17]). The RTA and RTA-LC tests are pseudo-polynomial in complexity, $O(n^2 D_{\max} D_{sum})$ where $D_{\max}$ is the longest task deadline, and D_{sum} is the sum of task deadlines. This derives from the fact that on each iteration of (13) or (15) the response time must increase by at least one for iteration to continue and after $D_{\max}$ such iterations the task would be deemed unschedulable. Further, the number of while loop iterations in Algorithm 1 is limited to D_{sum}, since on each iteration some response time must increase by at least one for the loop to continue iterating.

The following comparability relationships hold between the various schedulability tests. The RTA-LC test dominates the RTA test and the DA-LC test, both of which dominate the DA test. The DA-LC and RTA tests are incomparable.

D. Optimal priority assignment

In [22] and [23], Davis and Burns showed that Audsley's OPA algorithm [3], [4] can be used to obtain an optimal priority assignment with respect to any schedulability test that fulfils the following three conditions:

Condition 1: The schedulability of a task τ_k may, according to test *S*, depend on the set of tasks with priorities higher than *k*, but not on their relative priority ordering.

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

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

Inspection of the DA and DA-LC tests for gFPDS shows that these conditions hold (assuming fixed values of F_i) and so these tests are OPA-compatible. Whereas the dependency on the upper bound response time R_i^{UB} of higher priority tasks in (11) means that the RTA and RTA-LC tests are not OPA-compatible.

E. Example of gFPDS

We now provide an example comparing gFPDS with gFPPS and gFPNS. The example is based on the taskset in TABLE I. This taskset is trivially unschedulable on two processors with any form of fixed priority scheduling unless task τ_C has the lowest priority. Since task τ_A and task τ_B are equivalent, placing either of them at the lowest priority would make that task have a response time of 6 and so be unschedulable. Thus, there is only one viable priority ordering: τ_A, τ_B, τ_C.

TABLE I: TASK PARAMETERS

Task	Execution time	Period	Deadline
τ_A	3	10	5
τ_B	3	10	5
τ_C	8	25	12

With pre-emptive scheduling (gFPPS), if tasks τ_A and τ_B are released simultaneously, then task τ_C misses its deadline, as shown in Figure 2.

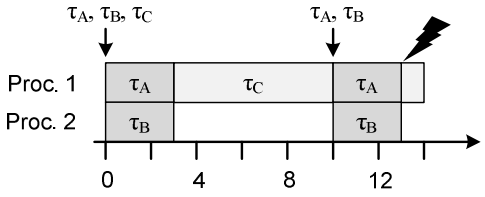

Figure 2: Schedule with gFPPS.

Similarly, with non-pre-emptive scheduling (gFPNS), if task τ_C is released just before tasks τ_A and τ_B, as shown in Figure 3, then task τ_B misses its deadline.

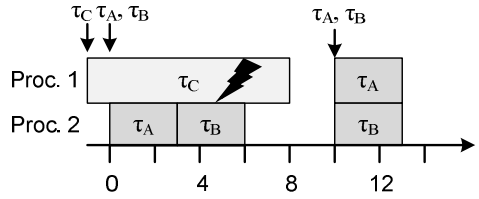

Figure 3: Schedule with gFPNS.

However, if we use deferred pre-emption and let $F_A = 1$, $F_B = 1$, and $F_C = 3$, then using the RTA test, we obtain $R_1^{UB} = 3$, $R_2^{UB} = 5$, and $R_3^{UB} = 11$; proving that the taskset is schedulable. Here, the FNR of task τ_C is enough to ensure that there can be no second pre-emption by task τ_A, yet task τ_C only blocks tasks τ_A and τ_B for a maximum of 2 time units enabling their deadlines to be met. This example illustrates the strict dominance, rather than equivalence, of gFPDS over gFPPS and gFPNS.

Note, this example has been deliberately constructed with Deadline Monotonic Priority Ordering (DMPO) as the only feasible priority ordering; however, it is well known that DMPO is not optimal for global fixed priority scheduling, and is not even a good heuristic [22], [23].

V. OPTIMAL gFPDS

In this section, we build upon the ideas and techniques developed in [27] which provide optimal algorithms for fixed priority scheduling with deferred pre-emption for uniprocessor systems. We pose the same two problems relating to the assignment of FNR lengths and priorities for the multiprocessor case, i.e. under gFPDS. We show that the first of these problems can be solved in a similar way to the uniprocessor case, and via a counterexample, that the second problem cannot.

Problem 1: *Final Non-pre-emptive Region length Problem* (*FNR Problem*). For a given taskset complying with the task

model described in Section III, and a given priority ordering *X*, find a length for the FNR of each task such that the taskset is deemed schedulable under gFPDS by schedulability test *S*.

Definition 1: An algorithm *A* is said to be *optimal* for the *FNR Problem* with respect to a schedulability test *S*, if there are no taskset / priority assignment combinations that are deemed schedulable under gFPDS by test *S* with some set of FNR lengths, that are not also deemed schedulable by the test using the set of FNR lengths determined by algorithm *A*.

Problem 2: *Final Non-pre-emptive Region Length and Priority Assignment Problem* (*FNR-PA Problem*). For a given taskset complying with the task model described in Section III, find both (i) a priority assignment, and (ii) a set of FNR lengths that makes the taskset schedulable under gFPDS according to schedulability test *S*.

Definition 2: An algorithm *B* is said to be *optimal* for the *FNR-PA Problem* with respect to a schedulability test *S*, if there are no tasksets compliant with the task model that are deemed schedulable under gFPDS by test *S* with some priority assignment *X* and some set of FNR lengths, that are not also deemed schedulable using the priority assignment and set of FNR lengths determined by algorithm *B*.

A. Sustainability with respect to FNR lengths

In order to be able to solve Problems 1 and 2 efficiently, we need to use schedulability tests that are sustainable [6], [20] with respect to changes in the length of a task's FNR. With a sustainable test, we can use binary search to help solve the problems. In contrast with an unsustainable test, we would potentially need to check every possible value for the FNR length of each task which is not practical.

Theorem 1: *The DA and DA-LC schedulability tests for task* τ_k *under gFPDS are sustainable with respect to increases in the length* F_k *of the task's FNR.*

Proof: To prove the theorem, it suffices to show that if (4) or (9) hold for some pair of values (C_k^* , D_k^*), then they continue to hold for the pair of values ($C_k^* - z$, $D_k^* - z$) where z is a positive integer ($z \le C_k^*$). Substituting $C_k^* - z$ for C_k^* and $D_k^* - z$ for D_k^* in (4) and (9), we need to show that the summation terms do not increase. By inspecting the component equations (1) – (3), and (5) – (8), we observe that the interference within a window of length L is monotonically non-decreasing with respect to the length of the window (i.e. it is no larger for an interval of length $D_k^* - z$ than it is for an interval of length D_k^*). Further, we must also consider the dependence of component equations (1) and (5) on C . C appears in the expression $L - C + 1$. which is unchanged by subtracting z from both L and C. The summation terms in (4) and (9) are therefore monotonically non-increasing with respect to increasing values of z □

Corollary 1: The schedulability of a task is, according to the DA and DA-LC tests, a monotonically non-decreasing function of the length of its FNR.

Theorem 2: (Negative result) *The RTA and RTA-LC schedulability tests for task* τ_k *under gFPDS are* ***not sustainable*** [6], [20] *with respect to increases in the length* F_k *of the task's FNR.*

Proof: Increasing the FNR length F_k of task τ_k increases the execution time of its associated virtual task τ_{kv} (as $C_{kv} = F_k - 1$). With the RTA and RTA-LC tests this can result in a large increase in the upper bound response time R_i^{UB} of some higher priority task τ_i due to the inclusion of interference from an extra job of a yet higher priority task, as well as the extra interference from τ_{kv} (i.e. blocking). The increase in R_i^{UB} can cause an extra job of task τ_i to interfere in the busy window of task τ_k making it unschedulable.

This scenario occurs with the taskset described in TABLE II below, assuming two processors. In this case, if task τ_D is fully pre-emptive, then the computed upper bound response times are 10, 5, 10 and 23 for tasks τ_A , τ_B , τ_C , and τ_D respectively; however, increasing the FNR length of task τ_D , so that $F_D = 2$, results in upper bound response times of 10, 6, 15, and 27, which would make task τ_D unschedulable if it had a deadline of 25. This increase in the upper bound response time of task τ_D is due to the large increase in the upper bound response time of task τ_C from 10 to 15, and the subsequent inclusion of an extra job of task τ_C in the busy window of task τ_D . It is easy to construct examples where decreasing the FNR length of a task τ_k can result in the task becoming unschedulable due to additional pre-emptions from higher priority tasks □

TABLE II: EXAMPLE TASK PARAMETERS

Task	Execution time	Period	Deadline
τ_A	10	100	10
τ_B	5	10	10
τ_C	5	15	15
τ_D	7	100	100

B. Solving the FNR and FNR-PA Problems

Due to the fact that the RTA and RTA-LC tests are unsustainable with respect to changes in FNR lengths, we now focus solely on the DA and DA-LC tests.

To aid in solving the FNR and FNR-PA problems, we introduce the concept of a *blocking vector*. For a given taskset and priority ordering *X*, we use $B(k)$ to represent the blocking vector at priority *k*, where the blocking vector relates to the set of FNR lengths of the ordered set of lower priority tasks $lp(k)$. Hence:

$$B(k) = ((F_n - 1),(F_{n-1} - 1)...(F_{k+1} - 1)) \quad (16)$$

We define a 'greater than or equal to' ($\ge$) and similarly a 'less than or equal to' ($\le$) relationship between blocking vectors with the meaning $B^1 \ge B^2$ if every element in B^2 is no larger than the corresponding element in B^1 .

We now state two corollaries about the DA and DA-LC tests for gFPDS.

Corollary 2: *Task schedulability under gFPDS according to the DA and DA-LC tests is sustainable with respect to decreases in the blocking vector*. Stated otherwise, according to the DA and DA-LC tests, a task that is schedulable at priority *k* with a blocking vector $B(k)$ remains schedulable when the blocking vector is reduced (e.g. by reducing the FNR length of one or more lower priority tasks) and the sets $lp(k)$ and $hp(k)$ of lower and higher priority tasks remain unchanged.

Corollary 3: *Using the DA and DA-LC schedulability tests for gFPDS, the minimum schedulable FNR length F_k for a task τ_k is monotonically non-increasing with respect to decreases in the blocking vector.* Stated otherwise, a smaller blocking vector at priority k cannot result in a larger minimum length for the FNR of the task at that priority level.

We now investigate using the FNR and FNR-PA algorithms presented in [27] to solve Problems 1 and 2 for multiprocessor systems. The two algorithms are the same as those used in the uniprocessor case with the exception that the schedulability tests used are the DA or DA-LC tests for gFPDS and due to Theorem 1, a binary search may be used to determine the smallest FNR length commensurate with task schedulability.

The proof of Theorem 3 uses the techniques from the uniprocessor case with minor adjustments for the way in which lower priority tasks now impinge on the schedulability of higher priority tasks.

```
for each priority level k, lowest first {
    determine the smallest value for the final
    non-pre-emptive region length F(k) such that
    the task at priority k is schedulable
    according to test S.
    Set the length of the final non-pre-emptive
    region of the task to this value.
}
```

Algorithm 2: FNR Algorithm

Theorem 3: The FNR algorithm (Algorithm 2) is *optimal* for the FNR problem (see Problem 1 and Definition 1).

Proof: We assume (for contradiction) that there exists a taskset τ and priority ordering X that is schedulable according to schedulability test S, with some set of FNR lengths F'_k for $k = 1$ to n, and that the FNR algorithm fails to determine a set of FNR lengths F_k for $k = 1$ to n, that results in the taskset being schedulable according to the test.

Let $B'(k)$ be the blocking vector at priority k with the schedulable set of FNR lengths, and $B(k)$ be the blocking vector at priority k with the set of FNR lengths computed by the FNR Algorithm. At each priority level, we will show that $F_k \le F'_k$ and hence that $B(k) \le B'(k)$ thus proving via Corollary 2 *sustainability of task schedulability with respect to blocking vectors* that the taskset is schedulable according to test S, with priority ordering X and the FNR lengths determined by the FNR Algorithm, thus contradicting the original assumption. The proof is by induction over each priority level k from n to 1.

Initial step: At the lowest priority level n, trivially we have $B(n) = B'(n) = \phi$. At priority n, the FNR Algorithm (Algorithm 2) computes, according to test S, the minimum schedulable FNR length F_n for task τ_n hence $F_n \le F'_n$.

Inductive step: We assume that at priority k, $B(k) \le B'(k)$ and $F_k \le F'_k$, hence $B(k-1) \le B'(k-1)$ and thus via Corollary 3, $F_{k-1} \le F'_{k-1}$

Iterating over all of the priority levels shows that for all k from n to 1, $B(k) \le B'(k)$ and so by Corollary 2, the taskset is schedulable, according to test S, with the set of FNR lengths F_k obtained by Algorithm 2 □

Corollary 4: (Follows from the proof of Theorem 3). For a given taskset and fixed priority ordering X, that is schedulable according to the DA or DA-LC schedulability test under gFPDS with some set of FNR lengths, Algorithm 2 minimises the FNR length of every task, and hence minimises the blocking vector at every priority level.

```
for each priority level k, lowest first {
   for each unassigned task τ {
        determine the smallest value for the
        final non-pre-emptive region length F(k)
        such that task τ is schedulable at
        priority k, according to test S assuming
        all other unassigned tasks have higher
        priorities.
        Record as task Z the unassigned task
        with the minimum value for the length of
        its final non-pre-emptive region F(k).
   }
   if no tasks are schedulable at priority k {
        return unschedulable
   }
   else {
        assign priority k to task Z and use the
        value of F(k) as the length of its final
        non-pre-emptive region.
   }
}
return schedulable
```

Algorithm 3: FNR-PA Algorithm

In contrast to the FNR problem, the FNR-PA problem requires a schedulable priority ordering to be established as part of the solution to the problem. Algorithm 3 which provides a solution to the FNR-PA problem in the uniprocessor case is based on Audsley's Optimal Priority Assignment (OPA) algorithm and uses a greedy bottom up approach. Hence it is in any case only compatible with the DA and DA-LC tests, and not the RTA and RTA-LC tests.

Theorem 4: (Negative result) The Final Non-pre-emptive Region Priority Assignment (FNR-PA) algorithm (Algorithm 3) is ***not optimal*** for the FNR-PA problem (see Problem 2 and Definition 2) in the multiprocessor case i.e. gFPDS using the DA or DA-LC schedulability tests.

Proof: Proof is via a counterexample where the FNR-PA algorithm fails to find a schedulable combination of priority assignment and FNR lengths, when such a combination exists. The example is for the DA test, similar tasksets can be constructed for the DA-LC test. We assume a system with two processors and the taskset given in TABLE III. With four tasks, there are 24 distinct priority orderings ($n!$ = 24); however, in this case only two are schedulable given appropriate choices of FNR lengths. Attempting to build a schedulable priority ordering from the lowest priority upwards, we find that neither task τ_A nor task τ_B is schedulable at the lowest priority (priority 4) even if they are made completely non-pre-emptable.

Case 1: If we assign task τ_D priority 4, then it requires a minimum FNR length of $F_D = 42$ to be schedulable. Then at priority level 3, we find that tasks τ_A and τ_B are again not schedulable, but task τ_C is schedulable with a minimum FNR length of $F_C = 38$. However, now due to the large combined blocking effect modelled as the virtual tasks τ_{Dv}

and τ_{Cv} (i.e. 41 + 37 = 78) neither task τ_A nor τ_B is schedulable at priority 2 and hence there is no schedulable priority assignment with task τ_D at the lowest priority.

Case 2: If we assign task τ_C the lowest priority, then it requires a minimum FNR length of $F_C = 58$ to be schedulable. Again we find that tasks τ_A and τ_B are not schedulable at priority 3. Now assigning task τ_D to priority 3, we find that it is schedulable with $F_D = 1$ (i.e. fully pre-emptive). Now, the blocking effect on whichever task, τ_A or τ_B, we choose for priority 2 is only 57, and hence either task is schedulable at that priority with the other at priority 1. In both cases we have $F_A = 1$ and $F_B = 1$.

The behaviour of the FNR-PA algorithm corresponds to Case 1 and so using the DA test, it would fail to find a schedulable combination of priority ordering and FNR lengths for this taskset; however, such a schedulable combination exists as shown in Case 2 □

TABLE III: COUNTEREXAMPLE TASK PARAMETERS

Task	Execution time	Period	Deadline
τ_A	36	207	110
τ_B	86	178	141
τ_C	93	525	195
τ_D	62	767	195

We note that the optimality of the FNR-PA algorithm breaks down in the multiprocessor case, because the blocking effect depends on a summation over the FNR lengths of lower priority tasks rather than a maximum, as in the single processor case. Minimising the FNR length at a given priority level does not necessarily minimise this summation, as shown in the above counterexample.

VI. EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of gFPDS with respect to gFPPS and gFPNS. We compared the scheduling algorithms under the following priority assignment policies: (i) Deadline Monotonic (DMPO), (ii) DkC [22], [23], and (iii) Audsley's Optimal Priority Assignment (OPA) algorithm for gFPPS and gFPNS. In the case of gFPDS, we used the FNR algorithm to obtain optimum final non-pre-emptive region lengths in conjunction with the heuristic priority assignment policies, and the FNR-PA Algorithm to provide both priority and FNR length assignment. We also made comparisons with the dynamic scheduling algorithm FPZL [24], [25] which has some similarities in its behaviour to gFPDS. The lines on the graphs are labelled according to the scheduling algorithm and priority assignment policy used, e.g. gFPDS (DkC). In all cases, we used the appropriate DA-LC test.

A. Parameter generation

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

- First, an unbiased set of n utilisation values $U_i \leq 1$, were generated with a total utilisation of U, (see [28] and [23] for how to generate an unbiased set of such values).
- Task periods were generated according to a log-uniform distribution (i.e. such that $\ln(T)$ has a uniform distribution). Here the ratio between the maximum and the minimum permissible task period was given by 10^r. By default, this range was 100, i.e. $r = 2$.
- Task execution times were set based on the task utilisation and period selected: $C_i = U_i T_i$.
- Task deadlines were *implicit*: $D_i = T_i$
- Taskset cardinality was z times the number of processors. By default, $z = 5$.

We examined systems with m = 2, 4, and 8 processors. In each experiment, the taskset utilisation was varied from $0.025m$ to $0.975m$ in steps of $0.025m$. For each utilisation value, 1000 tasksets were generated and their schedulability determined according to the various scheduling algorithms. Note due to the large number of lines on the graphs, the figures are best viewed online in colour.

B. Success ratio

In our first set of experiments, we compared the performance of the scheduling algorithms via the *success ratio*; the proportion of randomly generated tasksets that are deemed schedulable in each case.

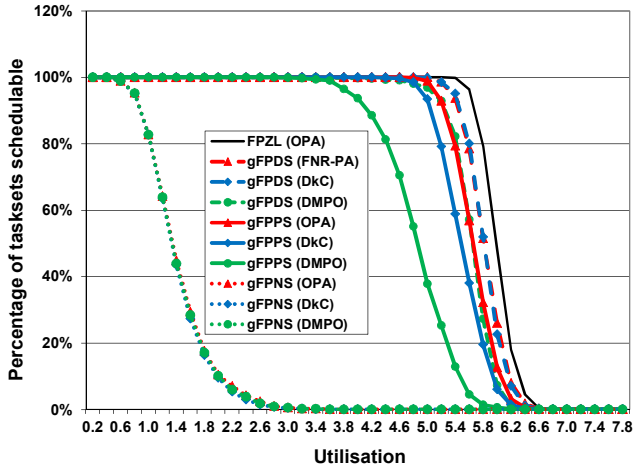

Figure 4: Success ratio for m = 8, n = 40, implicit deadlines

Figure 4 shows the results of this experiment for an 8 processor system with an implicit deadline taskset of cardinality 40, and a range of task periods of 100. We observe that the performance of gFPNS (dotted lines) was relatively poor for all priority assignment policies, due to the difficulty in accommodating tasks with long execution times. As expected, the results for gFPPS (solid lines with markers), show that optimal priority assignment outperformed the various heuristic priority assignment policies. Using gFPDS substantially better results were obtained for the various heuristic priority assignment policies as compared to gFPPS, with the best performance obtained using the FNR-PA algorithm. In all cases, gFPDS significantly outperformed gFPPS and gFPNS assuming a like-for-like priority assignment policy. gFPDS using the FNR-PA algorithm resulted in performance roughly half-way between that of gFPPS and the dynamic FPZL algorithm (solid line, no markers) assuming optimal priority assignment.

C. *Weighted schedulability*

In our second set of experiments we compared how the overall performance of each of the scheduling algorithms varies with respect to changes in a specific parameter via *weighted schedulability* [10].

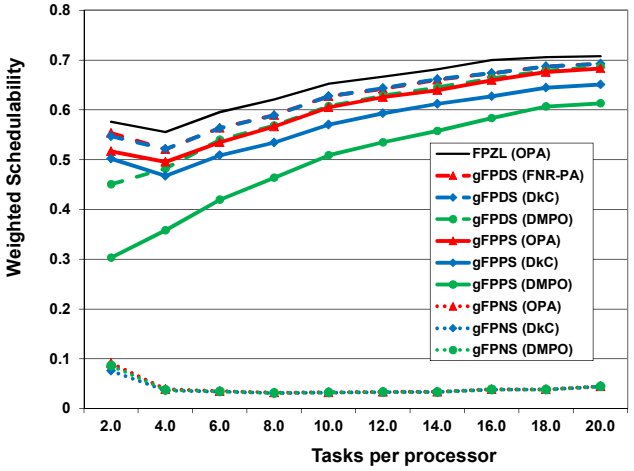

Figure 5: Weighted schedulability as a function of taskset size

The first parameter examined was taskset cardinality. Figure 5 shows how the weighted schedulability varies with increasing taskset size (from 2 to 20 tasks per processor, i.e. from 16 to 160 tasks on an 8 processor system) for each of the algorithms. We observe that increasing taskset cardinality results in tasks that have smaller utilisation on average and are therefore easier to schedule in the multiprocessor case, as noted in [22], [23]. As the ratio of tasks to processors increases, the advantage conferred by deferred pre-emption (and the dynamic FPZL algorithm) gradually decreases. This is because the individual utilisation of each task is becoming quite small reducing the benefits that can be obtained over fully pre-emptive scheduling.

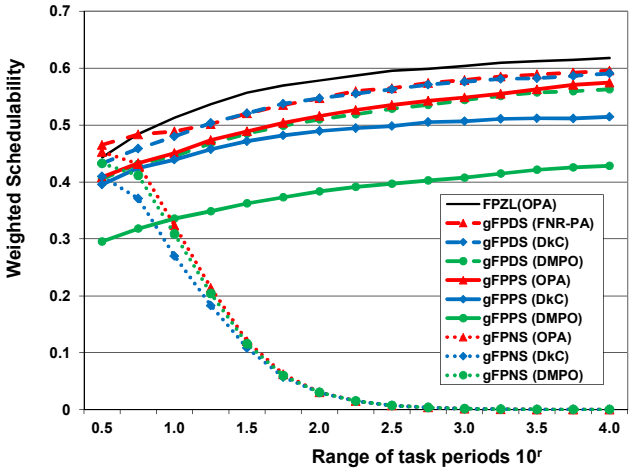

Figure 6: Weighted schedulability as a function of period range, D ≤ T

The second parameter we examined was the range of task periods. Figure 6 shows how the weighted schedulability varies with the log-range r of task periods given by the ratio 10^r between the maximum and the minimum permissible task period. Here, the value of r was varied from $r = 0.5$ ($10^{0.5} = 3.16$) to $r = 4$ ($10^4 = 10{,}000$). Figure 6 shows that gFPDS shows the largest improvement over gFPPS when the range of task periods is relatively small. This is because with all task periods and deadlines of a similar duration, all of the tasks can typically tolerate significant blocking and so there is scope to choose FNR lengths that improve schedulability.

As expected, both gFPPS and gFPDS show improved performance as the range of task periods increases, while gFPNS shows rapidly declining performance. This is because tasks with relatively long periods tend to have large execution times which may be longer than the deadlines of other tasks. Once there are more of these tasks than processors, non-pre-emptive scheduling becomes infeasible. (It is interesting to note that for very small ranges of task periods, gFPNS, and hence also gFPDS can be more effective than FPZL).

VII. Summary and Conclusions

Global fixed priority scheduling with deferred pre-emption (gFPDS), dominates both global fixed priority fully pre-emptive (gFPPS) and global fixed priority non-pre-emptive scheduling (gFPNS). In this paper we provided analysis for a simple model of gFPDS on homogeneous multiprocessors, where each task has a single non-pre-emptive region at the end of its execution. We showed that an appropriate choice of the length of this region can enhance schedulability.

The main contributions of this paper are as follows:

- Introduction of sufficient schedulability tests for gFPDS.
- Proof that the FNR algorithm [27] is compatible with the DA and DA-LC tests for gFPDS, and can be used to obtain the optimal final non-preemptive region lengths for a given priority ordering.
- Proof via a counterexample, that the joint problem of priority and FNR length assignment cannot be solved optimally via a greedy, bottom-up approach using the FNR-PA Algorithm from [27].
- An experimental evaluation of the performance benefits of gFPDS over gFPPS and gFPNS. We note that the additional comparisons with FPZL could be interpreted as suggesting that the dynamic algorithm FPZL is preferable; however, we have shown that much of the improvement FPZL obtains over gFPPS can be achieved by the simple adaptation of Final Non-pre-emptive regions (gFPDS). This approach fits better with the current fixed priority scheduling approaches used for example in the automotive electronics industry, and raises fewer issues for resource locking as under gFPDS, the priority of a task can only increase when it is actually running..

Building on this work, there are two key areas which we aim to explore. Firstly, in single processor systems, tasks often execute as a series of non-pre-emptive regions with pre-emption points between them [16]. However, with the normal fixed priority scheduling policy, such an arrangement is ineffective in the multiprocessor case. This is illustrated in

Figure 1, which shows that there is the potential for every non-pre-emptive region of every lower priority task to interfere with the execution of a higher priority task, making the approach unworkable. To address this problem, we intended to investigate simple modifications to the fixed priority scheduling policy that reduce such blocking effects.

Secondly, our simple model assumes that task execution times are independent of pre-emption and pre-emption and migration costs are negligible; however, in many real-time systems each pre-emption and migration incurs a significant cost, particularly in systems using cache. For large tasksets, allowing arbitrary pre-emption can result in lower priority tasks being pre-empted a large number of times, significantly increasing cache-related pre-emption delays (CRPD) to the detriment of schedulability [1], [2]. The integration of CRPD and schedulability analysis is a key area which we intend to explore further.

ACKNOWLEDGEMENTS

This work was partially funded by the UK EPSRC Tempo project (EP/G055548/1), the UK EPSRC MCC project (EP/K011626/1), and by Portuguese National Funds through FCT (Portuguese Foundation for Science and Technology), and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within the RePoMuC project, (FCOMP-01-0124-FEDER-015050).

REFERENCES

[1] S. Altmeyer, R.I. Davis, C. Maiza "Cache related Pre-emption Delay aware response time analysis for fixed priority pre-emptive systems". In proceedings Real-Time Systems Symposium, pp. 261-271, 2011.

[2] S. Altmeyer, R.I. Davis, C. Maiza "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems" . Real-Time Systems, 48 (5), pp. 499-526, 2012

[3] 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, 1991.

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

[5] T.P. Baker. "Multiprocessor EDF and deadline monotonic schedulability analysis". In proceedings. Real-Time Systems Symposium (RTSS), pp. 120–129, 2003.

[6] S.K. Baruah, A. Burns, "Sustainable Scheduling Analysis". In proceedings Real-Time Systems Symposium, pp. 159-168, 2006.

[7] S.K. Baruah. "The limited-preemption uniprocessor scheduling of sporadic task systems". In Proceedings Euromicro Conference on Real-Time Systems, pp. 137–144, 2005.

[8] S.K. Baruah, "Techniques for Multiprocessor Global Schedulability Analysis". In proceedings Real-Time Systems Symposium, pp. 119-128, 2007.

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

[10] A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability" In Proceedings of OSPERT, , pp. 33-44, Brussels, Belgum, 2010.

[11] M. Bertogna, M. Cirinei, G. Lipari, "New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors". In proceedings International Conf. 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 proceedings Real-Time Systems Symposium, pp. 149-158, 2007.

[13] 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.

[14] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, Francesco Esposito, Marco Caccamo. "Preemption points placement for sporadic task sets", In Proceedings Euromicro Conference on Real-Time Systems, Bruxelles, Belgium, June 2010.

[15] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, G. Buttazzo. "Optimal Selection of Preemption Points to Minimize Preemption Overhead", In Proceedings Euromicro Conference on Real-Time Systems, Porto, Portugal, July 2011.

[16] M. Bertogna, G. Buttazzo, G. Yao. "Improving Feasibility of Fixed Priority Tasks using Non-Preemptive Regions", In proceedings Real-Time Systems Symposium, 2011.

[17] M. Blum, R..W. Floyd, V. Pratt, R.L. Rivest, R.E. Tarjan, "Time bounds for selection". *Journal of Computer and System Sciences 7*, 4 (Aug. 1973), 448–461.

[18] R. Bril, J. Lukkien, and W. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. Real-Time Systems, 42(1-3):63–119, 2009.

[19] A. Burns. "Preemptive priority based scheduling: An appropriate engineering approach". S. Son, editor, Advances in Real-Time Systems, pp. 225–248, 1994.

[20] A. Burns, S.K. Baruah "Sustainability in real-time scheduling". Journal of Computing Science and Engineering 2 (1), pp 74-97. 2008.

[21] G.C. Buttazzo, M. Bertogna, G. Yao. "Limited Preemptive Scheduling for Real-Time Systems: A Survey". IEEE Transactions on Industrial Informatics, 9(1) pp. 3-15 Feb 2013.

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

[23] R.I. Davis, A. Burns, "Improved Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems". Real-Time Systems 47 (1) pp1-40, 2011.

[24] R.I. Davis, A. Burns, "FPZL Schedulability Analysis", In proceedings Real-Time Applications and embedded Technology Symposium (RTAS), pp. 245-256, 2011.

[25] R.I. Davis and S. Kato "FPSL, FPCL and FPZL schedulability analysis." Real-Time Systems, 48 (12), pp 750-788, 2012.

[26] R.I. Davis, A. Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems", ACM Computing Surveys, 43, 4, Article 35 44 pages, October 2011,

[27] R.I. Davis, M. Bertogna "Optimal Fixed Priority Scheduling with Deferred Pre-emption". In proceedings Real-Time Systems Symposium, 2012.

[28] P. Emberson, R. Stafford, R.I. Davis "Techniques For The Synthesis Of Multiprocessor Tasksets". In proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010) , pp. 6-11, July 6th, 2010.

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

[30] N. Guan, W. Yi, Q. Deng, Z. Gu, G. Yu, "Schedulability analysis for non-preemptive fixed-priority multiprocessor scheduling". Journal of Systems Architecture - Embedded Systems Design 57(5), pp. 536-546, 2011.

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

[32] G. Yao, G. Buttazzo, M. Bertogna. "Bounding the Maximum Length of Non-Preemptive Regions Under Fixed Priority Scheduling", In proceedings RTCSA 2009, Beijing, China, August 2009.

APPENDIX

We note that while our schedulability tests for gFPDS dominate the equivalent tests for gFPPS, they do not dominate the equivalent tests for gFPNS which include blocking from a most m lower priority tasks [30]. We can however apply specific schedulability tests for gFPDS for the special case of a task τ_k which is fully non-pre-emptive, as set out below.

A. Special case of a fully non-pre-emptive task in a gFPDS system

In the case of gFPDS scheduling where a task τ_k is fully non-pre-emptive, i.e. when $F_k = C_k$, $D_k^* = D_k - (C_k - 1)$, and $C_k^* = 1$, then more precise analysis is possible. This analysis is based on the approach of Guan et al. [30] for gFPNS (where *all* tasks are fully non-pre-emptive).

For a fully non-pre-emptive task τ_k we note that:

(i) A lower priority task τ_j can only delay the execution of task τ_k if it begins to execute its final non-pre-emptive region prior to the release of task τ_k. Hence the maximum interference from a lower priority task τ_j is zero if it does not have a carry-in job[2] and $F_j - 1$ if it has a carry-in job. So, for the virtual task τ_{jv} representing τ_j, we have $I_{jv}^D(L,C) = I_{jv}^R(L,C) = F_j - 1$, and $I_{jv}^{NC}(L,C) = 0$.

(ii) With m processors, at most m lower priority tasks can be in non-pre-emptive regions when task τ_k is released (see Lemma 5.2 of [30]). So at most m virtual tasks can have carry-in jobs.

(iii) In the worst-case scenario, at most m –1 higher priority tasks can have carry-in jobs (Theorem 1 of [23]).

DA test: *for a fully non-pre-emptive task τ_k under gFPDS:*

$$D_k^* \geq C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^D(D_k^*, C_k^*) + \sum_{\forall j \in MB(k)} (F_j - 1) \right) \right\rfloor \quad \text{(A.1)}$$

where $MB(k)$ is the subset of at most m tasks with the largest values of $F_j - 1$, from the set of tasks $lp(k)$.

The DA test for gFPDS takes account of points (i), and (ii) above, but not point (iii).

The DA-LC test below combines points (ii) and (iii).

DA-LC test*: for a fully non-pre-emptive task τ_k under gFPDS:*

$$D_k^* \geq C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(D_k^*, C_k^*) + \sum_{i \in MDB(k)} I_i^{DIFF-D}(D_k^*, C_k^*) \right) \right\rfloor \quad \text{(A.2)}$$

where $MDB(k)$ is the subset of the m tasks with the largest values of $I_i^{DIFF-D}(D_k^*, C_k^*)$ from the set of tasks $hp(k) \cup lpv(k)$ provided at least one of those tasks is from $lpv(k)$, otherwise $MDB(k)$ equates to the subset of at most m-1 tasks with the largest values of $I_i^{DIFF-D}(D_k^*, C_k^*)$ given by (8), from the set of tasks $hp(k)$, and the single virtual task from $lpv(k)$ that has the largest value of $I_i^{DIFF-D}(D_k^*, C_k^*)$.

RTA test: *The upper bound response time R_k^S for the start (first unit of execution) of a fully non-pre-emptive task τ_k under gFPDS, may be computed via the fixed point iteration given by (A.3) within Algorithm 1. The task is schedulable if $R_k^S \leq D_k^*$, where D_k^* is the task's effective deadline $D_k^* = D_k - (C_k - 1)$.*

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^R(R_k^S, C_k^*) + \sum_{\forall j \in MB(k)} (F_j - 1) \right) \right\rfloor \quad \text{(A.3)}$$

If the task is schedulable, then an upper bound on its worst-case response time is given by $R_k^{UB} = R_k^S + (C_k - 1)$.

RTA-LC test: *The upper bound response time R_k^S for the start (first unit of execution) of a fully non-pre-emptive task τ_k under gFPDS, may be computed via the fixed point iteration given by (A.4) within Algorithm 1. The task is schedulable if $R_k^S \leq D_k^*$, where D_k^* is the task's effective deadline $D_k^* = D_k - (C_k - 1)$.*

$$R_k^S \leftarrow C_k^* + \left\lfloor \frac{1}{m} \left(\sum_{\forall i \in hp(k)} I_i^{NC}(R_k^S, C_k^*) + \sum_{i \in MRB(k)} I_i^{DIFF-R}(R_k^S, C_k^*) \right) \right\rfloor \quad \text{(A.4)}$$

where $MRB(k)$ is the subset of the m tasks with the largest values of $I_i^{DIFF-R}(R_k^S, C_k^*)$ from the set of tasks $hp(k) \cup lpv(k)$ provided at least one of those tasks is from $lp(k)$, otherwise $MDB(k)$ equates to the subset of at most m-1 tasks with the largest values of $I_i^{DIFF-R}(R_k^S, C_k^*)$ given by (14), from the set of tasks $hp(k)$, and the single virtual task from $lpv(k)$ that has the largest value of $I_i^{DIFF-R}(R_k^S, C_k^*)$. If the task is schedulable, then an upper bound on its worst-case response time is given by $R_k^{UB} = R_k^S + (C_k - 1)$.

We note that the RTA and RTA-LC tests given by (A.3) and (A.4) do not depend on the upper bound response times of lower priority tasks, and so the iteration of Algorithm 1 is unnecessary if all tasks are fully non-pre-emptive. In that case, upper bound response times may be evaluated highest priority first.

We observe that the schedulability tests given in this section for the special case of a fully non-pre-emptive task dominate the equivalent tests for the general case of deferred pre-emption with $F_k = C_k$ given in section IV. This means that the DA and DA-LC tests retain their monotonic behaviour with respect to increasing values of F_k if in the special case of $F_k = C_k$ we use the specific tests given by (A.1) and (A.2) instead of the more general ones given by (4) and (9). This was done in our experimental evaluation.

[2] We refer to a lower priority task as having a carry-in job if it enters its final non-pre-emptive region (effectively releasing its corresponding virtual task) before the release of the task of interest (start of the busy window).