---

**Article:**
Zhao, Shuai, Garrido, Jorge, Wei, Ran et al. (3 more authors) (2020) A Complete Run-time Overhead-aware Schedulability Analysis for MrsP under Nested Resources. Journal of Systems and Software. ISSN 0164-1212

https://doi.org/10.1016/j.jss.2019.110449

---

# A Complete Run-time Overhead-aware Schedulability Analysis for MrsP under Nested Resources

Shuai Zhao[a], Jorge Garrido[b,*], Ran Wei[a,*], Alan Burns[a], Andy Wellings[a], Juan A. de la Puente[b]

[a]*Department of Computer Science, University of York, York, YO10 5GH, UK*
[b]*STRAST research group, Universidad Politécnica de Madrid (UPM), Spain*

**Abstract**

Multiprocessor Resource Sharing Protocol (MrsP) is a hard real-time multiprocessor resource sharing protocol for fully partitioned fixed-priority systems, and adopts a novel helping mechanism to allow task migrations during resource accessing. Previous research focusing on analysing MrsP systems have delivered two forms of timing analysis which effectively bound response time and migration cost of tasks under MrsP, and have demonstrated advantages of this protocol. An adjustable non-preemptive section is also introduced that effectively reduces the number of migrations needed during each resource access. However, these analysis methods are only applicable if a non-nested resource accessing model is assumed. In addition, there is no clear approach towards the configuration of the non-preemptive section length, and the computation cost for applying the analysis remains unknown.

In this paper, we extend the MrsP analysis for systems with nested resources. Major run-time costs incurred by MrsP tasks are also taken into account to form a complete run-time cost-aware schedulability analysis. In addition, recommendations towards non-preemptive section configuration are given from both analytic and empiric perspectives. Finally, a set of evaluations are conducted to investigate schedulability of MrsP under nested resources and the cost for applying the proposed analysis. As a result of this paper, the schedulability test for MrsP is complete and the computation costs of its use are now understood.

*Keywords:* hard real-time system; multiprocessor resource sharing protocol; schedulability test; nested resources; run-time overhead.

*Corresponding author

*Email addresses:* shuai.zhao@york.ac.uk (Shuai Zhao), jgarrido@dit.upm.es (Jorge Garrido), ran.wei@york.ac.uk (Ran Wei), alan.burns@york.ac.uk (Alan Burns), andy.wellings@york.ac.uk (Andy Wellings), jpuente@dit.upm.es (Juan A. de la Puente)

## 1. Introduction

Real-time applications have become more sophisticated with increased functionality demanding more computational power (Block et al., 2007). This has necessitated a transition from uniprocessor execution platforms to multiprocessor ones. The movement towards multiprocessor platforms has raised many theoretical and practical challenges for the real-time system developers, where matured uniprocessor real-time scheduling techniques cannot be directly applied (Davis and Burns, 2011). Resource sharing is one of the major problems in real-time multiprocessor systems, where the well-practised uniprocessor resource sharing protocols cannot be applied as these techniques assume that resources are accessed from a single processor (Brandenburg, 2011). Whilst both physical and logical resources are of research interest, they are differentiated research topics. In this work, we focus on the sharing of logical resources, such as shared data structures and I/O ports.

With multi-tasking, two or more tasks may request *exclusive access* to the same resource (i.e., a shared resource) simultaneously. The code related to a shared resource is called a *critical section*. Shared resources that are accessed only from one processor are termed *local resources*. In multiprocessor systems, resources can potentially be accessed from more than one processor in parallel, and are termed *global resources*. To avoid race conditions and protect data consistency, locks[1] are commonly adopted to protect shared resources in real-time systems. Each resource is protected by a designated lock, where the access to a shared resource is only permitted with the corresponding lock acquired. Deadlocks must be avoided to ensure system progress and to satisfy temporal requirements.

A task's behaviour while accessing a shared resource must be predictable and conclude within bounded time. With resource locks, tasks can incur additional delays due to accessing shared resources, which leads to *priority inversions*. A priority inversion happens when a high priority task is waiting for a shared resource but cannot proceed (i.e., is being *blocked*) because a low priority task is executing with that resource. Priority inversion cannot be completely eliminated due to the difficulty in controlling the exact time at which a given task can access a shared resource. However, it must be bounded to achieve a predictable blocking time for each resource access. With locks applied, deadlocks must be avoided to guarantee system execution progress, which is essential for real-time systems to meet temporal requirements.

For uniprocessor systems, resource sharing is successfully managed by matured resource control technology, which is well-understood and has been applied

[1] The lock-based approach is the dominant synchronisation approach in real-time systems (Davis and Burns, 2011). We acknowledge the existence of other synchronisation approaches, such as Lock-Free (Anderson et al., 1997) and Wait-Free (Sundell and Tsigas, 2000) algorithms. However, these algorithms rely on multiple snapshots of shared resources for monitoring state changes, which are not always feasible as memory spaces are often very limited in real-time applications.

for decades with several optimal resource sharing policies available (Davis and Burns, 2011), such as the Priority Ceiling Protocol (PCP) in (Sha et al., 1990). Standard schedulability analysis techniques, such as Response Time Analysis (RTA) (Audsley et al., 1993), have been extended to include the blocking time for these resource sharing protocols (Brandenburg, 2011; Burns and Wellings, 2013). However, these techniques cannot directly be applied on multiprocessors due to the simple fact that resource requests can be issued simultaneously from multiple processors. Although there are several multiprocessor resource sharing protocols available, there is no agreed best approach (Brandenburg and Anderson, 2010), because the performance of these protocols depends highly on the characteristics of the given applications, such as the length of critical sections (Zhao et al., 2017). Amongst existing multiprocessor resource sharing protocols, early efforts in locking protocols map mainly to extensions of uniprocessor protocols, such as MPCP by Rajkumar et al. (1988) and MSRP by Block et al. (2007). However, these protocols assume a restricted resource-accessing model, where nested accesses to shared resources is not allowed (Garrido et al., 2017b). Such restrictions are relaxed later on by protocols proposed explicitly for multiprocessors, which assume a more flexible resource-accessing model with nested resources allowed and deadlocks avoided (Ward and Anderson, 2012a; Burns and Wellings, 2013).

This article focuses on a multiprocessor resource sharing protocol proposed by Burns and Wellings (2013) for fully partitioned multiprocessor systems, namely the Multiprocessor resource sharing Protocol (MrsP). In MrsP, preemptive spin-locks are adopted and resource access requests are served in a FIFO order. Although targeting fully-partitioned systems, MrsP introduces a novel helping mechanism that allows task migration for resource accessing[2], where a preempted task that is accessing a resource can keep executing by migrating to a remote processor where there is a task spin-waiting for the resource. With this helping mechanism, resource-accessing tasks can keep progressing after being locally preempted and the blocking of high priority tasks can be minimised by the preemptive approach. As illustrated in (Burns and Wellings, 2013), the definition of this protocol yields a temporal behaviour very similar to that of the well-known Priority Ceiling Protocol (Rajkumar, 1991) for uniprocessor systems. With this feature, MrsP has attracted notable attention, as relevant previous research and practitioners' results from PCP (e.g., the deadlock-free mechanism and the analysis techniques (Sha et al., 1990; Audsley et al., 1993; Burns and Wellings, 2016)) can be easily applied to MrsP with minor modifications, thus easing the adoption of multiprocessor platforms for hard real-time systems (Burns and Wellings, 2013). The current version of MrsP contains a complete nested resource accessing model with a simple analysis that provides an upper blocking bound (Garrido et al., 2017b). A short configurable non-

[2]Fully partitioned scheduling requires each task is assigned with an allocation prior to execution, but allows temporary allocation changes made by resource sharing protocols (Davis and Burns, 2011)

preemptive section has been introduced in its helping mechanism (Zhao and Wellings, 2017) that effectively reduces the number of task migrations required for each resource access. In addition, a migration-cost aware schedulability analysis is supported for non-nested resource accessing in (Zhao et al., 2017), and can provide more accurate schedulability results than that of the original analysis given in (Burns and Wellings, 2013). As observed in (Garrido et al., 2017a; Zhao et al., 2017; Shi et al., 2017), MrsP outperforms other similar protocols (Craig, 1993; Anderson et al., 1998; Gai et al., 2001) under certain application characteristics based on either system schedulability or response time of tasks in real-world applications, especially with long critical sections.

However, with the presence of nested resource accesses, the current analysis of MrsP by Garrido et al. (2017b) carries considerable pessimism compared to the state-of-the-art analysis techniques (Wieder and Brandenburg, 2013; Zhao et al., 2017) due to the issue of over-calculating the time required to execute critical sections (see Section 3.4). In addition, as illustrated in (Zhao et al., 2017), the potential cost of task migrations has a non-trivial impact on MrsP schedulability, and this is not considered in the analysis for nested resources. However, as demonstrated in the above study, failing to bound such cost can lead to direct system failure due to deadline misses and should not be ignored by any form of schedulability tests. Further, although the NP-section adopted in the helping mechanism is shown to be effective in reducing the cost of migrations (Zhao and Wellings, 2017) in general, it is not clear how to set the NP-section length. Due to above issues, although the current version of MrsP is a promising multiprocessor resource control solution with several reference implementations available (Catellani et al., 2015; Zhao and Wellings, 2017), further research and development work still needs to be carried out for it.

In this article, we present a number of contributions all of which address the above concerns. These contributions can be summarised as follows:

- extensions to the schedulability analysis presented by Zhao et al. (2017) to support nested resource access, including the extensions to the migration cost analysis;
- a complete analysis approach for incorporating all run-time and implementation overhead: preemptions, context switches and locking operations;
- an approach for the configuration of NP-section aiming to reduce number of task migrations during each resource access;
- an extensive evaluation of the presented extensions, and a performance evaluation of the revised protocol's schedulability compared with other relevant FIFO spin-based protocols.

As the result of the work presented in this article, a complete MrsP run-time cost-aware schedulability test is now available that bounds the worst-case blocking time for accessing nested resources as well as run-time costs incurred by tasks from both the protocol and the underlying operating system. In addition, with NP-section configuration clarified, true performance (in terms of

schedulability) of this protocol under nested resources can be determined while its advantages over other relevant FIFO spin-based protocols are understood. Finally, the computation cost of applying the proposed analysis to a range of applications is investigated and is compared with other relevant forms of schedulability tests.

The rest of the article is organised as follows. Section 2 provides a review of resource sharing technology for real-time systems relevant for the present work. Section 3 describes MrsP and explains previous efforts made towards both the definition and schedulability tests of this protocol. In Section 4, the response-time analysis is extended to support the analysis of MrsP systems under presence of nested resources. Section 5 presents the complete run-time overhead-aware analysis for MrsP, which bounds the costs of migrations under nested resources and incorporates overheads imposed from the underlying operating system. Section 6 gives a comprehensive approach for bounding and reducing the migrations costs under MrsP with a fine-tuned configuration of short non-preemptive sections. Section 7 presents the evaluation results on the newly developed analysis and comparisons against relevant protocols. Finally, Section 8 concludes the article and supplies some interesting future work directions.

## 2. Resource Sharing in Real-time Systems

The research into shared resources in real-time systems can be traced back to late 1980's and is broad. The majority of these resource sharing techniques work in collaboration with a Fixed-Priority Scheduler (FPS), where the priority of each task must be statically assigned prior to execution (Brandenburg, 2011). In this sense, a higher priority value indicates higher execution eligibility. Task activation conforms to the general sporadic task model, in which each task can give rise to a potentially infinite sequence of invocations (i.e., task releases), but each release must be invoked after a minimum interval (e.g., period) has elapsed since its last arrival (Davis and Burns, 2011), where the worst-case scenario happens when all tasks are released immediately after such intervals.

This section focuses on the major real-time resource sharing protocols and their schedulability tests[3] (if they exist). The basic concepts and notions towards real-time systems (e.g., sporadic task model and fixed-priority scheduling policy) used in this article can be found in a survey paper on hard real-time scheduling for homogeneous multiprocessor systems by Davis and Burns (2011). The rest of this section is organised as follows: first, Section 2.1 reviews the major uniprocessor protocols that introduced the main techniques and associated system properties with regards to resource sharing adopted by modern multiprocessor protocols. Then Section 2.2 presents the most relevant multiprocessor

[3]Schedulability test is a mathematical tool that determines whether a given system is schedulable via an analytical approach. A common approach is to calculate the worst-case response time of all tasks in the system (Davis and Burns, 2011). The system is regarded as schedulable if all tasks can meet their deadlines (i.e., response time of a task is equal to or lower than its deadline).

resource sharing protocols to date, categorised by the novel features they introduced. Section 2.3 presents modern scheduling analysis techniques that can further improve schedulability results of existing protocols meeting certain criteria. Finally, the presented protocols and analysis techniques are summarised and compared in Section 2.4, as well as areas of improvement are identified, motivating the contributions of this paper.

### *2.1. Uniprocessor Resource Control Protocols*

Uniprocessor scheduling algorithms incorporating shared resource accesses have been successfully studied and implemented. The most relevant examples are, for this work, derivatives of the original work on priority inheritance protocols defined by Sha et al. (1990). These protocols coordinate the tasks' execution when accessing a shared resource and provide a blocking bound for each resource access.

The Priority Inheritance Protocol (PIP) (Sha et al., 1990) increases the *active priority* (i.e., the current priority value of a given task) of a task holding a resource when blocking a higher priority task[4]. By doing so, the blocking task (i.e., the low priority task that is executing with the resource) is assigned with a higher execution eligibility so that it will incur no interference from other unrelated tasks (e.g., a task with an intermediate priority that does not require the resource), and hence, the blocking time suffered by the higher priority task is reduced. However, PIP is not ideal as a task can be blocked more than once if it requests several resources and is not deadlock-free (see system execution examples in (Zhao, 2018)). This motivates the development of the optimal resource sharing protocols for uniprocessor systems, as described below.

The Priority Ceiling Protocol (PCP) proposed by Sha et al. (1990) increases the priority of a resource-requesting task to the highest among all tasks that (will) require the same resource, known as resource *ceiling priority*. Under PCP, a task that requests a resource raises its priority to the corresponding ceiling priority, and executes with this priority until it releases the resource. Executing with ceiling priority effectively delays executions of other tasks requesting the same shared resource, preventing the formation of a circular resource-requesting chain and hence, avoiding deadlocks. In addition, all resources requested by a task are guaranteed to be available when the task starts its execution, i.e., there is no task running at the ceiling priority for accessing those resources. By doing so, PCP provides an optimal blocking bound in a uniprocessor system (Davis and Burns, 2011), in which the blocking is bounded to solely one critical section and can only happen before the real execution of tasks (this is termed *arrival blocking*, being blocked upon task's arrival).

The Stack Resource Policy (SRP) (Baker, 1990) extends PCP and provides

[4]FPS does not impose any changes towards task priority, but allows priority changes made by resource control protocols. This does not break the working mechanism of FPS as this scheduler only requires that a priority is assigned to each task before execution (Davis and Burns, 2011).

an optimal blocking bound for both FPS and Earliest Deadline First scheduling[5]. As with PCP, the notion of resource ceiling is applied. In addition, this protocol introduces preemption levels based on the deadline monotonic scheme, where a task with a shorter relative deadline is assigned with a higher preemption level. With this static metric, SRP is able to work with dynamic scheduling policies using deadlines. Accordingly, the value of the resource ceiling for each resource is decided by the static preemption levels. On FPS, SRP demonstrates identical behaviour as PCP, where the preemption levels are mapped to resource ceiling via task priority instead of deadlines.

Table 1: Notations in the PCP/SRP Analysis

| | |
|---|---|
| $\tau_i$ | A task that is currently been studied. |
| $R_i$ | Response time of $\tau_i$. |
| $C_i$ | The pure worst-case computation time of $\tau_i$ without accessing any shared resources. |
| $T_i$ | Period of $\tau_i$. |
| $B_i$ | The maximum blocking time $\tau_i$ can incur in each release. |
| $\mathbf{hp}(i)$ | The set of tasks with a priority higher than that of $\tau_i$. |
| $Pri(r^k)$ | The resource ceiling priority of $r^k$. |
| $\hat{c}_i$ | The arrival blocking incurred by $\tau_i$. |
| $\hat{b}$ | Maximum length of NP-sections in the underlying Real-time operating system. |
| $c^k$ | Computation time of resource $r^k$. |
| $N_i^k$ | Number of accesses $\tau_i$ requests to $r^k$ during one release. |
| $F(\tau_i)$ | Resources that are requested by $\tau_i$. |

Both PCP and SRP systems can be analysed via the Response-Time Analysis (RTA) technique proposed by Audsley et al. (1993), in which the response time $R_i$ of a task $\tau_i$ is calculated iteratively by using Equation (1), where $C_i$ denotes $\tau_i$'s pure computation time, i.e., without the time waiting for and executing with shared resources, $\mathbf{hp}(i)$ gives the set of tasks with priorities higher than $\tau_i$, $B_i$ is the maximum blocking $\tau_i$ can incur due to resource accessing and $T_j$ gives the period of $\tau_j$,. The system is schedulable if the iteration reaches a fixed point, and the response time of all tasks is equal or lower their deadlines. The cost due to resource accessing is computed by function $\sum_{r^k \in F(\tau_i)} N_i^k c^k$, where $F(\tau_i)$ returns resources that are requested by $\tau_i$, $N_i^k$ denotes the number of access $\tau_i$ requests to $r^k$ during one release and $c^k$ gives the computation cost for executing with $r^k$.

$$R_i = C_i + \sum_{r^k \in F(\tau_i)} N_i^k c^k + B_i + \sum_{\tau_j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \sum_{r^k \in F(\tau_j)} N_j^k c^k) \qquad (1)$$

[5]Earliest Deadline First scheduling is a typical dynamic scheduling policy that does not rely on task priorities but absolute task deadlines, where the task with the closest deadline is assigned with the highest execution eligibility (Liu and Layland, 1973).

Notation $B_i$ can be further extended as shown in Equation (2), where $\hat{c}$ indicates the maximum critical section length of all resources that are requested by at least one task with priority less than $Pri(\tau_i)$ and at least one task with equal or higher priority, and $\hat{b}$ is the maximum length of NP-sections in the underlying Real-time operating system (RTOS). Table 1 summarises the notations in schedulability tests of PCP/SRP.

$$B_i = \max\{\hat{c}, \hat{b}\} \tag{2}$$

### *2.2. Multiprocessor Resource Sharing Protocols*

The research into multiprocessor resource sharing problem is broad, where there exist many locking protocols with unique features. Below we review eight major multiprocessor protocols in total, which are categorised by the novel features they introduced.

**Multiprocessor PCP versions** Due to the simple implementation and analysis of priority ceiling protocols, the PCP approach has been translated into different forms for multiprocessor systems. The most relevant, the Multiprocessor Priority Ceiling Protocol (MPCP) by Rajkumar et al. (1988) introduces the requirement to migrate to a synchronisation processor to access a resource, as a means to serialise accesses. Later in (Rajkumar et al., 1988), MPCP is extended to distributed systems (DPCP) with the notion of a remote agent that can execute a resource's critical sections on behalf of remote tasks. As an early multiprocessor resource sharing protocol, MPCP (and DPCP) manages shared resources in a uniprocessor fashion (via the synchronisation processor) so that matured uniprocessor resource sharing techniques can be directly applied. This was highly appreciated in 1980s, given that there existed no multiprocessor resource sharing solutions.

**Multiprocessor SRP** Another relevant approach to control shared resources in multiprocessor systems is the Multiprocessor Stack Resource Policy (MSRP) by Gai et al. (2001), developed as an extension of SRP. Resources under MSRP are accessed from the task host processor in a non-preemptable fashion and any not immediately satisfied requesting task keeps spinning also non-preemptably until the access is granted. A FIFO queue is used to grant access to the resource allowing the spin-waiting time to be bounded by the number of processors with tasks that request the resource. However, MSRP only supports a limited nested resource accessing model, where nested accesses between global resources are not allowed. With the spin-waiting time taken into account in the execution time of resources, MSRP systems can be analysed by Equation (1) with minor modifications to reflect the potential parallel accesses to shared resources (Gai et al., 2001), where $hpl(i)$ returns $\tau_i$'s local high priority tasks. In Equation (3), $\overline{C_i}$ is the worst case execution time of $\tau_i$, including the time it spends when waiting (spinning) for and executing with each required resource. Equation (4) presents $\overline{C_i}$ calculation, where notation $c^k$ in Equation (1) is replaced by $e^k$ to reflect potential parallel accesses to globally shared resources, including $\tau_i$'s spin-waiting time for the resource. Accordingly,

notation $\hat{c}$ in Equation (2) is also replaced by $\hat{e}$ to include the potential parallel accesses to resources that can cause $\tau_i$ to incur arrival blocking, as given in Equation (5).

$$R_i = \overline{C_i} + B_i + \sum_{\tau_j \in \mathbf{hpl}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \overline{C_j} \tag{3}$$

$$\overline{C_i} = C_i + \sum_{r^k \in F(\tau_i)} N_i^k e^k \tag{4}$$

$$B_i = \max\{\hat{e}, \hat{b}\} \tag{5}$$

With the above equations, the response time of $\tau_i$ is bounded by its pure executing cost $C_i$, the waiting time $\tau_i$ takes waiting for (i.e., being blocked) and executing with each required resource, arrival blocking $B_i$ and total higher priority interference (i.e., computation time and resource-accessing time $\overline{C_j}$) from each of $\tau_i$'s local higher priority task (denoted as $\tau_j$).

As requests to a resource under MSRP are served in a non-preemptive FIFO order, $e^k$ is effectively bounded by the number of processors containing requests to $r^k$, as given in Equation (6), where $G(r^k)$ gives the set of tasks that require $r^k$, function $map()$ returns a set of processors where the given tasks are assigned to and $||$ returns the size of the given set. Table 2 summarises the notations in schedulability analysis of MSRP.

$$e^k = |map(G(r^k))| * c^k \tag{6}$$

Table 2: Notations in the MSRP Analysis

| | |
|---|---|
| $\mathbf{hpl}(i)$ | A set of local tasks with a priority higher than that of $\tau_i$. |
| $\overline{C_i}$ | The complete execution time of $\tau_i$, including its resource accessing time. |
| $e^k$ | Worst-case accessing time to resource $r^k$, including the delay due to parallel resource accesses. |
| $\hat{e}$ | Worst-case arrival blocking of $\tau_i$, including potential parallel accesses to resources that can cause $\tau_i$ to incur this blocking. |
| $\|\|$ | The size of a given set. |
| $G(r^k)$ | Tasks that request $r^k$. |
| $map()$ | Processors where the given tasks are assigned to. |

**Waiting and accessing schemes** While the use of FIFO queues for granting access to globally shared resources has been widely adopted, different waiting and access schemes have been proposed. Notable examples are the O($m$) Locking Protocol (OMLP) proposed by Brandenburg and Anderson (2010), where tasks first contend for acquiring a common $m$-exclusion priority lock

and then are suspended until they become the head of the FIFO queue associated to the required resource, finally, the task becomes non-preemptable under OMLP when accessing the resource; or the preemptive resource sharing approach (PWLP) (Anderson et al., 1998; Craig, 1993). Under the latter protocol tasks spin-wait for shared resources at their base priority. If the scheduler preempts a spinning task, it cancels the task's current resource request and the task is placed back at the end of the FIFO queue when rescheduled. Once granted the resource lock, a task becomes non-preemptable during the entire execution of the critical section.

**Helping protocols** Progress on shared resources upon local preemption of tasks can also be achieved via the notion of *helping*. In Spinning Processor Executes for Preempted Processor (SPEPP) proposed by Takada and Sakamura (1997), tasks insert in a FIFO queue the action to be performed on the resource as an operation block. When the task responsible of the action is locally preempted having been granted access to the resource, the queued operation block is executed non-preemtptively by a waiting task (if any is available). Another notable helping mechanism is proposed in the Multiprocessor Bandwidth Inheritance (M-BWI) protocol by Faggioli et al. (2010), for soft real-time systems. M-BWI is an *execution-time server* based protocol where a resource-holding task that is locally preempted or runs out of budget can be helped by other tasks waiting for that resource by being migrated to the helper's processor and consuming the budget of the helper.

**Resource grouping** Support for nested resources can be generally achieved by grouping resources together. The Flexible Multiprocessor Locking Protocol (FMLP) proposed by Block et al. (2007) uses this notion to manage each group with different approaches based on specific semantics of the group. In particular, FMLP distinguishes between short and long resources, and resource groups can only include either short or long resources. Short resources are accessed in FIFO order and both the spinning and access is done non-preemptively, while long resources are also serviced in FIFO order but tasks suspend until they are granted access to the resource. That is, FMLP advocates that spin-based locks are not favourable for long critical sections (Block et al., 2007). Nested access is only allowed between resources of the same group, and also from long resources to short ones. This same approach of resource grouping is used in other multiprocessor protocols to partially support nested resource accesses, such as OMLP and PWLP. Grouping resources, however, has the side effect of undermining the degree of parallelism and system composability (Davis and Burns, 2011), where in an extreme case parallelism can be reduced to the uniprocessor case as resources requested by different tasks are guarded by one lock, and only serialised access is allowed.

**Fine grained nested resources access** Fine-grained blocking bounds for nested resources was first achieved by Takada and Sakamura (1995) and refined in the Real-time Nested Locking Protocol (RNLP) by Ward and Anderson (2012b), which only requires a partial order on the resource nesting to avoid deadlocks. Under RNLP, concurrency is limited by a $k$-exclusion token and a number of satisfaction (access granting) mechanisms are defined, providing

sub-optimal results under different system configurations. Unfortunately, these optimal results are only possible for certain specific system configurations, and are therefore impractical (from the scheduling and system complexity point of view) to support all possible combinations of tokens and satisfaction mechanisms on a given system to address real-world scenarios.

### *2.3. Schedulability Tests in the Presence of Blocking*

In addition to Response Time Analysis, further analysis techniques have been proposed that can be applied to MSRP directly, or to similar protocols (such as PWLP) with modifications. A holistic analysis is presented in (Brandenburg, 2011) that analyses the exact number of remote requests being issued for a shared resource that can actually cause blocking of a task accessing the same resource in one release. By avoiding the assumption that each time a task tries to access a resource, it can be blocked once from each remote processor that contains tasks requesting the same resource, the holistic analysis is able to provide more accurate blocking time bounding than that of the original tests.

In addition, the mixed-integer linear programming (ILP) technique is introduced to the schedulability analysis in (Brandenburg, 2013a; Wieder and Brandenburg, 2013). This ILP-based analysis preserves the advantage of the holistic analysis in (Brandenburg, 2011) (i.e., computing the exact number of requests) and further improves schedulability results by guaranteeing that each critical section is accounted for only once. By defining up to 30 spin locks constraints, this ILP-based analysis framework can provide schedulability tests to 8 spin-based resource sharing protocols, including MSRP and PWLP as described above. Later, this analysis framework was extended by Biondi et al. (2016) to support fine-grained nested access analysis based on graph abstraction that reflects conflicts and transitive delays between shared resources.

### *2.4. Summary and Discussion*

As described in Section 2.2, each multiprocessor resource sharing protocol has a unique combination of resource classification, queuing techniques and resource-accessing rules. In addition, some protocols (e.g., PWLP) also contain an additional mechanism (e.g., request cancellation) that further reduces blocking in a multiprocessor environment, and contains different approach for supporting nested resources (if they exist). Table 3 summarises the main features of each reviewed protocol, where "-" denotes that the feature is not supported by a given protocol. A detailed comparison of these protocols is reported in (Zhao, 2018). Below we summarise the comparison and emphasis the major observations.

As shown in this table, most protocols classify resources as either global or local, and manage global resources with multiprocessor resource control techniques. An exception to this is FMLP, that manages shared resources by their length of critical sections (denoted as short and long resources). The advantage of doing so is that each resource can be managed by the most appropriate lock, and hence, unnecessary system overheads (i.e., avoiding the use of suspension-based locks on short resources) as well as processor idle time (i.e., avoiding

Table 3: Features of Reviewed Multiprocessor Resource Sharing Protocols

| Protocol | Resources | Accessing Rule | Queuing Technique | Additional Facility | Nested Resource |
|---|---|---|---|---|---|
| MPCP | Global & Local | Priority Ceiling | - | Synchronisation processor | Ordered Locks |
| MSRP | Global & Local | Non-Preemptive | FIFO | - | - |
| OMLP | Global & Local | Priority Inheritance & Non-Preemptive | FIFO & Priority Ordered | - | Group Locks |
| PWLP | Global & Local | Base Priority for Waiting; Non-Preemptive for Holding | FIFO | Cancel | Group Locks |
| SPEPP | Global & Local | Base Priority for Waiting; Non-Preemptive for Holding | FIFO | Operation Blocks | - |
| M-BWI | - | Base Priority of Servers | FIFO | Execution Server | Ordered Locks |
| FMLP | Short & Long | Non-Preemptive for Short; Suspension for Long | FIFO | - | Group Locks |
| RNLP | - | $k$-exclusion token | Satisfaction mechanism | - | Ordered Locks |
| MrsP | Global & Local | Priority Ceiling | FIFO | Migration-based Helping | Ordered Locks |

spinning too long) could be decreased. However, applying FMLP requires the support of both suspension-based and spin locks from the underlying operating system. In addition, there exists no clear instruction for classifying resources under FMLP, which increases the difficulty of adopting this protocol.

Among the reviewed protocols, there exist three major resource-accessing rules, where a task can access a shared resource with a) its base priority; b) priority boosting (either priority inheritance or ceiling priority) or c) in a non-preemptive fashion. The non-preemptive approach provides the strongest execution progress guarantee, but can impose extra blocking to high priority tasks, where they can be prevented from executing due to a low priority task is executing non-preemptively with a shared resource. In contrast, accessing resources with base priority cannot provide any protection towards resource accessing tasks (which can be preempted at any time), but has minimised blocking to high priority tasks. Finally, the priority ceiling approach provides a trade-off

between the non-preemptive and base priority approaches, where unrelated high priority tasks can still execute while the resource-accessing tasks are protected from tasks with an intermediate priority. MPCP adopts priority ceiling and uses a synchronisation processor for resource-accessing. However, this approach serialises the execution with any resources, decreases the degree of parallelism in multiprocessor systems and does not support nested resources between global resources (Davis and Burns, 2011).

Two mainstream approaches are commonly adopted for queuing resource-accessing tasks. These are to queue tasks either by their base priority or following a FIFO order. With priority-ordered queues, the waiting time of low priority tasks can be prolonged significantly. In turn, FIFO queues lead to the theoretical bounding of $|map(G(r^k))|$ for a given resource $r^k$. Thus, FIFO queuing is generally preferred and adopted by the majority.

Moreover, additional facilities are introduced by certain protocols to reduce the blocking in multiprocessor systems. In particular, PWLP presents the cancellation mechanism to facilitate the resource accessing routine, where preempted resource-waiting tasks cancel their requests and rejoin into the resource contention later on when being resumed. This approach minimises the arrival blocking to one critical section only (same as the uniprocessor case), but introduces extra overhead to low priority tasks, which could be preempted frequently and have a prolonged resource-waiting time. The helping mechanism in SPEPP accelerates resource-accessing via wasted CPU cycles (i.e., when a task is spinning). However, its adoption is limited to atomic operations only and cannot support nested resources.

Finally, nested resource accesses are supported by some of the reviewed protocols via either group locks and ordered locks, where group locks decrease the degree of parallelism while ordered locks impose restrictions towards the resource model. With group locks, more than one resource are managed by the same lock so that accesses towards these resources are serialised. Ordered locks mandate that accesses towards nested resources must be one way only, in order to prevent deadlocks, but this does not undermine the performance of the system, and hence, is more favourable (Ward and Anderson, 2012b).

From the above discussion, each resource sharing approach has its unique advantages and limitations. As concluded by Davis and Burns (2011); Zhao (2018), there exists no optimal resource sharing solution in real-time multiprocessor systems. As reported by Wieder and Brandenburg (2013), spin locks are mandated in majority of real-world safety-critical systems due to its low runtime overhead. However, spin locks are known to be not favourable for long critical sections (Brandenburg et al., 2008), as valuable CPU cycles are wasted while tasks are spinning for shared resources. Consequently, helping mechanisms are adopted by spin-based protocols (e.g., SPEPP), which utilise the wasted cycles to facilitate resource accesses and executions. However, these protocols impose strong limitations towards the resource-accessing model, where operations must be atomic and nested resources accesses are not allowed. To advance the state of art, MrsP (Burns and Wellings, 2013) (see Section 3) introduces a migration-based helping mechanism, in which wasted CPU cycles are utilised for

executing critical sections. Compared with the existing helping-based protocols, MrsP supports a less restrictive resource-accessing model, where atomic operations towards shared resources are no longer mandated. In addition, this protocol demonstrates the capability for supporting nested resource access (Burns and Wellings, 2013; Garrido et al., 2017b). In this paper, a complete definition is presented for MrsP to fully support fine-grained nested resources via order locks with corresponding schedulability analysis.

As for schedulability tests reviewed in Section 2.3, the ILP-based analysis proposes advanced analysis techniques and can provide schedulability results with minimised pessimism when compared to traditional RTA. Unfortunately, this analysis does not consider any helping-based protocols due to the focus on generic spin-based frameworks, and hence, cannot be directly applied to MrsP. In addition, it is difficult to extend the ILP-based analysis to bound the potential migration cost under MrsP due to the complexity of the migration cost analysis. Further, as the ILP-based analysis relies on an optimisation process (which gradually decreases the blocking bound via iterations until the constraints are met), it presents a high computation cost. This higher cost further discourages the adoption of this approach for analysing MrsP systems. Therefore, the analysis presented in this paper is based on the RTA due to its high expandability nature (Audsley et al., 1993) but incorporates lessons learnt from ILP-based analysis (i.e., the issues of inflating task computation time with resource-accessing time and over-calculating critical sections, see Section 3.4 for details).

## 3. MrsP: Definition and Analysis

MrsP (Burns and Wellings, 2013) is a multiprocessor resource sharing protocol aimed at fully-partitioned systems with fixed priority scheduling, that provides a safe upper bound of the accessing cost to resources shared among tasks executing on different processors. This protocol has a novel migration-based helping mechanism and PCP-like temporal behaviour. Research efforts have been made towards completing and improving its definition and analysis (Garrido et al., 2017b; Zhao et al., 2017), as well as implementing and evaluating this protocol in real-world operating systems (Garrido et al., 2017a; Catellani et al., 2015; Zhao and Wellings, 2017; Shi et al., 2017). This section describes the original definition proposed by Burns and Wellings (2013) and summarises our previous work on this protocol.

### *3.1. Original Protocol Definition*

In MrsP, each shared resource has a set of ceiling priorities, one for each processor that contains tasks requesting the resource. The ceiling priority for a given processor is set to the highest priority among tasks requesting the resource on that processor. Once a task requests a resource, it raises its priority to the ceiling of the resource on its processor and executes with the ceiling priority during its entire access. With this mechanism, tasks requesting the same

resource in one processor do so at the same ceiling priority. Thus, only one task per processor can request access to the same shared resource at a time[6]. Access requests are satisfied in FIFO order, waiting actively to be served (i.e., spinning), preventing lower priority tasks to be released during the busy-waiting time, consequently reducing context switch overhead. With FIFO spinning, the total access cost of a request could be bounded and would be equal to the maximum number of possible simultaneous access requests multiplied by the access time of the resource (i.e., the bounding given by Equation (6)). However, such a safe upper bound is only achievable if the resource-accessing task (either holding or waiting for the resource) is executed in a non-preemptive manner. Otherwise, both the resource-holding task and other waiting tasks (on remote processors) can be locally preempted by higher priority tasks, which interferes the resource-accessing routine and prolongs the cost for accessing a shared resource.

To achieve the previously mentioned bounded blocking time, a *helping mechanism* is introduced in this protocol where any task waiting to access a resource is able to undertake the associated computation (i.e., critical section) "on behalf of" any other task accessing the same resource. Hence, when a resource-accessing task is preempted, it can be helped by other waiting tasks to ensure progress. In the worst-case, a task needs to execute on behalf of all other tasks in the FIFO queue (according to the FIFO order) whenever it tries to access a resource. This situation happens when this task is placed at the end of FIFO queue and all other tasks in the queue are locally preempted. This will not increase the worst-case response time of the helping task as it has to wait for all other tasks in the FIFO queue to execute with the resource before it can do so (i.e., the FIFO resource-accessing order). In contrast, this helping mechanism speedups the resource-accessing routine as all these resource-accessing tasks do not need to wait for potential preemptors before executing with the resource. The choice of the helping task is not forced by MrsP and is up to implementations. For instance, MrsP implementation provided in (Zhao and Wellings, 2017) always searches for a valid (i.e., actively spinning on its processor) helping task from the top of the FIFO queue.

The helping mechanism is realised by task migrations, where the locally preempted resource-accessing task is migrated to a processor where a task is actively spin-waiting to access the resource. After migration, the task is assigned the priority of the helping task and then resumes its execution with the resource. If the task is preempted again, it can either migrate to another remote processor with a waiting task spinning for the resource or to its original processor if the initial preemptor is finished. After the task releases the resource, it migrates back to its original processor if necessary, i.e., it was being helped on a remote processor.

Accordingly, the mentioned blocking bound (i.e., Equation (6)) of each resource access is obtained, and MrsP is compatible with the MSRP schedulability

[6]Under FPS, a First Come First Serve strategy is used when two tasks have the same priority (Burns and Wellings, 2016).

test described in Section 2, with $\hat{e}$ in Equation (5) replaced by $\hat{e_i}$ given in Equation (8) for bounding the arrival blocking due to the use of the priority ceiling facility. A task $\tau_i$ incurs arrival blocking if there exist any local lower priority task that requests a resource $r^k$, which has a resource ceiling priority on $\tau_i$'s processor that is equal or higher than the priority of $\tau_i$, where $F^A(\tau_i)$ denote such resources, $N_{ll}^k$ denotes the number of such local low priority tasks, $P(\tau_i)$ gives the processor of $\tau_i$, $Pri(\tau_i)$ denotes $\tau_i$'s priority and $Pri(r^k, P_m)$ returns the ceiling priority of $r^k$ on processor $P_m$.

$$F^A(\tau_i) \triangleq \{r^k | N_{ll}^k > 0 \wedge Pri(r^k, P(\tau_i)) \geq Pri(\tau_i)\} \tag{7}$$

$$\hat{e_i} = max\{e^k | F^A(\tau_i)\} \tag{8}$$

In the presence of nested accesses to resources, MrsP exhibits the same behaviour as PCP. However, simply following PCP on a multiprocessor can raise the issue of deadlocks (i.e., circular chain of requests to resources). To avoid deadlocks, two approaches that are commonly adopted by other multiprocessor protocols are discussed in (Burns and Wellings, 2013): *group locks* and *resource ordering*. A group lock can be applied to serialise the access of nested resources so that the circular chain can be prevented. However, this approach could impose prolonged resource accessing time to the tasks that access resources in a non-nested fashion. The other approach is to order the resources statically and to only allow a task to access a resource with a higher order index than that of any other currently held resource. Such an approach imposes restrictions to the resource-accessing model but can be more expressive than group locks. Therefore, there is a trade-off between the use of group locks and nested access. However, assigning resources with orders is preferred and is assumed by Burns and Wellings (2013); as with group locks nesting is essentially avoided rather than tolerated. As noted earlier, this is the approach adopted by RNLP (Ward and Anderson, 2012b). Equation 9 gives the bounding for one access to $r^k$ with potential access requests to $r^k$'s via a nested fashion (denote as $V(r^k)$).

$$e^k = (|V(r^k)| + |map(G(r^k))|)c^k \tag{9}$$

### *3.2. Nested Resource Access*

Although two possible approaches and a simple analysis for supporting nested resources are provided by Burns and Wellings (2013), it has been demonstrated in (Garrido et al., 2017b) that the support for nested resource access in MrsP's original definition is insufficient as tasks can incur extra local blocking due to nested access when used together with the helping mechanism. For instance, with nested access, a task can access further resources while being helped on a remote processor, which could lead to a priority boost according to PCP. However, with the boosted priority, the task could block the current executing task on its original processor when migrating back with the resources (i.e., after being preempted again on the remote processor), which breaks the property of PCP that a task can incur local blocking (i.e., blocking from the task's hosting

processor) only once during each release. To preserve this property, Garrido et al. (2017b) proposed a specific set of rules that tasks should follow when accessing nested resources, including a dynamic priority assignment scheme.

As defined by Garrido et al. (2017b), a task does not update its active priority when accessing inner resources while being helped in a remote processor. Thus, the migrated task can not benefit from the helping mechanism as its priority remains at the priority of the helping task. In turn, tasks are re-dispatched on their host processor with the priority they had when they were locally preempted. We will refer to this priority as the *Leaving Priority* for the rest of the paper. Migrated tasks do update their active priorities when they are re-dispatched on their host processor.

In the presence of nested resources, the helping mechanism can be initiated if the spinning task requires a resource that is held by a preempted task regardless of the nesting level of the resource (i.e., not necessarily the immediately inner resource). As mentioned, the migrated task keeps executing with the priority of the helping task. In this case, obtaining or releasing resources does not cause any priority update. The helping mechanism is finished when the resource holder releases the resource that the helping task requires. In addition, "transitive helping" is allowed to cope with the case where a task that is being helped (say $\tau_h$) requires a resource that is held by another preempted task (say $\tau_p$). In this case, $\tau_h$ can help $\tau_p$ in its current processor until $\tau_h$ releases the required resource.

In addition, the definition of the existing helping functions is clarified in (Garrido et al., 2017b). In the nested access case, function $F(\tau_i)$ should only return the outer-most resources that are directly accessed by $\tau_i$ without any nested access while $G(r^k)$ should give the set of tasks that access $r^k$ directly as the outer-most resource. This will not affect the non-nested analysis presented above as tasks can execute with only one resource (i.e., the outer-most resource) at any given time. Then, Equation (9) is modified to account for the potential transitive blocking incurred when accessing nested resources, as given in Equation (10), where $V(r^k)$ returns the set of outer resources that access $r^k$ as an inner resource and $\mathbf{U}(r^k)$ gives a set of resources that are accessed by $r^k$ directly (i.e., without further nesting).

$$e^k = (|V(r^k)| + |map(G(r^k))|) * (c^k + \sum_{r_q \in \mathbf{U}(r_k)} N_i^{kq} e^q) \tag{10}$$

This analysis, which includes the whole cost of accessing a nesting of resources from a specific resource ($F(\tau_i)$ in Equation (4), returns the set of outermost resources requested by $\tau_i$) has a similar form to the non-nested analysis shown in Equation (6). The left-hand side of this equation gives the longest possible queue to access a resource, and the right-hand side gives the maximum accessing time for that resource and all inner resources it requires.

### *3.3. Controlled Migrations in Helping Mechanism*

With the helping mechanism, MrsP theoretically guarantees an identical resource-accessing time bound to the non-preemptive protocol (recall analysis

in Section 3.1). However, as our previous work suggests, allowing migrations on-demand in a real-time system could decrease the predictability of task behaviours (Zhao et al., 2017). Consider, in the case where a resource accessing task is preempted and there exists a large number of potential migration targets that reside in processors with one or more high priority tasks with very short periods. Under such cases, the resource accessing task can suffer frequent migrations. In extreme situations, the task can spend significantly more time migrating than executing in critical sections, which greatly undermines the usability of the protocol.

To prevent excessive migrations and to offer a more efficient resource-accessing behaviour, a short and tuneable non-preemptive section (NP-section) is introduced by Zhao et al. (2017), where a newly migrated task is allowed to execute non-preemptively for a short time before it inherits the priority of its helper. As demonstrated by Zhao et al. (2017), this simple approach can provide guaranteed progress to the resource-accessing tasks and effectively reduce the number of migrations. The only side effect of the NP-section is that any newly released high priority task has to cope with the cost of one NP section before it can preempt the holder and execute. However, the length of the NP section can be configured so that high priority tasks are still able to meet their deadlines. We denote the length of the NP section as $C_{np}$. With the NP-section adopted, the local blocking, $B_i$ is updated as Equation (11), where $\hat{np}_i$ denotes the blocking time imposed by this facility.

$$B_i = max\{\hat{e}_i, \hat{np}_i, \hat{b}\} \tag{11}$$

For a task $\tau_i$, it can incur such blocking as long as $\tau_i$ has a priority equal or higher than the lowest ceiling priority of global resources on its processor. Otherwise $np_i$ is 0 as the task cannot preempt any task accessing a shared resource, and, consequently, is not affected by the NP-section:

$$\hat{np}_i = \begin{cases} C_{np}, & \text{if } Pri(\tau_i) \geq min_{\{r^k \text{ is global}\}} Pri(r^k, P(\tau_i)) \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

### *3.4. Response Time Analysis*

The analysing technique given above (see Section 2) is attractive due to its simplicity and elegancy, and can be applied with limited knowledge of the application. However, as shown by Wieder and Brandenburg (2013), the analysis can over calculate critical sections as it assumes each resource request will be blocked $|map(G(r^k))|$ times regardless the actual number of remote requests being issued during that period. Such an assumption can lead to response time boundings that are much higher than the actual worst-case values, and hence, is considered to be pessimistic. In addition, this analysis does not consider the potential blocking caused by a phenomenon named "back-to-back" hits, which was firstly reported in (Brandenburg, 2013a) and is fully elaborated with details in (Zhao et al., 2017). Such a phenomenon happens where a given task $\tau_i$ is

released only once during the release of $\tau_2$ (i.e., $\left\lceil \frac{R_1}{T_2} \right\rceil = 1$), yet can cause one more blocking due to the resource access made in its last release ($\left\lceil \frac{R_1+R_2}{T_1} \right\rceil = 2$).

Consequentially, Zhao et al. (2017) proposed new MrsP response time analysis that addresses the above issues. Under this analysis, the response time of $\tau_i$ is bounded by Equation (13). $E_i$ is the total resource accessing time of $\tau_i$ with *direct spin delay* (i.e., being blocked directly by remote tasks for accessing a shared resource) accounted for. $I_{i,h}$ indicates the *indirect spin delay*, where $\tau_i$ is blocked indirectly by its local higher priority, which preempts $\tau_i$ but is blocked for requesting a locked resource. $B_i$ denotes the *arrival blocking* (i.e., occurs upon $\tau_i$'s arrival), where it cannot execute because a local lower priority task is accessing a resource with an active priority equal to or higher than $\tau_i$'s priority $Pri(\tau_i)$.

$$R_i = C_i + E_i + B_i + \sum_{\tau_h \in hpl(i)} \left( \left\lceil \frac{R_i}{T_h} \right\rceil \cdot C_h + I_{i,h} \right) \tag{13}$$

The resource accessing time by $\tau_i$ itself ($E_i$) and indirect spin delay ($I_{i,h}$) are computed by the same function but require different input parameters, as shown in Equations (14) and (15), where $e_x^k(l, \mu)$ gives the accessing time to resource $k$ that task $\tau_x$ can incur within the duration $l$ and a release jitter $\mu$. By giving different duration and jitter length, the function returns a different bounding as $\tau_x$ can be released a different number of times (and generate a different number of requests) within the given duration.

$$E_i = \sum_{r^k \in F(\tau_i)} e_i^k(R_i, 0) \tag{14}$$

$$I_{i,h} = \sum_{r^k \in F(\tau_h)} e_h^k(R_i, R_h) \tag{15}$$

Equation (14) gives the total resource accessing time of $\tau_i$ itself, including the direct spin delay. For the direct spin delay, we consider $l = R_i$ and $\mu = 0$ so that only the resource requests in $\tau_i$'s one release will be accounted for (we enforce that $R_i \leq D_i$). As for the indirect spin delay incurred by $\tau_i$ from local high priority tasks $\tau_h$ (Equation 15), $l = R_i$ and $\mu = R_h$ so that the potential delay due to the back-to-back hit can be accounted for when computing the total number of requests issued from a high priority task $\tau_h$ to $r^k$ in the context of $\tau_i$ (i.e., during $\tau_i$'s release).

For a given task, the blocking time incurred for each access to a resource may vary as there may not exist resource requests on each remote processor that can cause the delay for every access of the given task. Thus, function $e_x^k(l, \mu)$ is computed via analysing the resource accessing time of a task in each individual access, as given in Equation (16), where $e_x^k(l)(n)$ gives the resource accessing time of $\tau_x$'s $n$th access to $r^k$ within the duration $l$.

$$e_x^k(l, \mu) = \sum_{n=1}^{N_x^k(l,\mu)} e_x^k(l)(n) \tag{16}$$

To include the "back-to-back" hits in the number of requests issued from other tasks during $\tau_i$'s release, a helper function $N_x^k(l,\mu)$ (see Equation (16)) is introduced, where $N_x^k(l,\mu) = \left\lceil \frac{l+\mu}{T_x} \right\rceil \cdot N_x^k$. For a given task $\tau_x$, function $N_x^k(l,\mu)$ gives the number of requests $\tau_x$ can issue to resource $k$ within the given duration $l$ and a jitter $\mu$, and $N_x^k$ gives the number of requests $\tau_x$ can issue to $r^k$ in one release. For instance, it can compute the number of requests issued from $\tau_1$ to a resource during $\tau_2$'s release, $l = R_2$ and $\mu = R_1$ so that the back to back hit can be accounted for. In addition, two additional functions are introduced to compute the number of resource requests issued from a group of tasks, where $Nh_x^k(l) = \sum_{\tau_h \in hpl(x)} N_h^k(l, R_h)$ gives the number of requests issued by local high priority tasks of $\tau_x$ during the time $l$ and $Np_m^k(l) = \sum_{\tau_j \in \tau(P_m)} N_j^k(l, R_j)$ gives the number of requests issued from a remote processor $m$ within the duration $l$.

Then, the worst case bounding of $e_x^k(l,\mu)$ is formed by two theorems proved by Zhao et al. (2017), where notation $(f(x))_a^b$ denotes $min\{max\{f(x), a\}, b\}$, where $a$ and $b$ are positive integers with $a \leq b$, summarised as follows.

**Theorem 1.** *The maximum number of requests on a remote processor $m$ that **may** block $\tau_x$ directly for accessing $r^k$ within the duration $l$ is bounded by $NS_{x,m}^k(l) = (Np_m^k(l) - Nh_x^k(l))_0$.*

**Theorem 2.** *The number of direct spin delays that $\tau_x$ **can** incur for accessing $r^k$ from a remote processor $m$ within the duration $l$ and jitter $\mu$ is bounded by $min\{NS_{x,m}^k(l), N_x^k(l,\mu)\}$.*

With the above theorems, $e_x^k(l)(n)$ is constructed as shown in Equation (17), where $n$ takes the values $1, \ldots, N_x^k(l,\mu)$ (defined by Equation (16)) and one extra $c^k$ is accounted for the access by $\tau_x$ itself. In $\tau_x$'s $n$th access, requests from a remote processor $m$ can block $\tau_x$ only if there still exists unaccounted requests on that processor, i.e., $(NS_{x,m}^k(l) - n + 1)_0 \geq 1$. Upon one access, there can be at most one request on a remote processor that can cause the spin delay, and hence $(NS_{x,m}^k(l) - n + 1)_0^1$.

$$e_x^k(l)(n) = \sum_{P_m \neq P(\tau_x)} (NS_{x,m}^k(l) - n + 1)_0^1 \cdot c^k + c^k \tag{17}$$

The arrival blocking caused by a given resource (say $r^k$) is effectively bounded by identifying the set of processors that contain unaccounted requests to $r^k$ that can cause arrival blocking to $\tau_i$, denoted as $\alpha_i^k$ and is bounded in Equation (18). Thus, the arrival blocking time due to $c^k$ is $|\alpha_i^k| \cdot c^k$, which forms a new equation for $\hat{e_i}$, as shown in Equation (19). Finally, by integrating this $\hat{e_i}$ to Equation (11), the arrival blocking $B_i$ for $\tau_i$ is then safely bounded.

$$\alpha_i^k \triangleq \{P_m | NS_{i,m}^k(R_i) - N_i^k > 0 \wedge P_m \neq P(\tau_i)\} \cup P(\tau_i) \tag{18}$$

$$\hat{e_i} = max\{|\alpha_i^k| \cdot c^k | r^k \in F^A(\tau_i)\} \tag{19}$$

As proved in (Zhao et al., 2017), this analysis guarantees that each critical section is accounted for only once and the blocking time is safely bounded with the back-to-back hit considered. As such, this analysis delivers less pessimistic as well as more accurate response time boundings than that of the original one given in (Burns and Wellings, 2013). Furthermore, the analysis is independent of the priority assignment scheme and is not fixed to any specific hardware architecture. With an initial response time, say $C_i$, the analysis computes the blocking variables and then updates the response time of all tasks in the system iteratively and alternately until a fixed-point is reached.

### *3.5. Cost of Migrations*

As described in Section 3.1, migrations are required in MrsP due to the helping mechanism, which imposes extra overhead due to necessary updates in the underlying operating system, cache misses and pipeline stalls. As measured by Zhao and Wellings (2017), a full migration operation (i.e., from the point the task is queued for migration until the time it is scheduled for execution on a remote processor) under the Litmus$^{\mathrm{RT}}$ system (Calandrino et al., 2006a) costs 8.4 microseconds on average. Such costs are non-negligible to real-time systems and should not be ignored by MrsP analysis. To bound the cost of migrations, a migration cost analysis is developed in (Zhao et al., 2017). This analysis treats the cost of one migration as a constant upper bound (i.e., $C_{mig}$) and computes the maximum number of migrations a task can perform during each release due to accessing shared resources under MrsP.

The analysis is constructed based on the following theorem (see complete proof in (Zhao et al., 2017)), which identifies potential migration targets for a resource accessing task via examining whether there exist requests to the same resource during the given period.

**Theorem 3.** *In $\tau_x$'s n-th access to $r^k$ within a duration $l$, the set of migration targets for $\tau_x$ is $mt_x^k(l)(n) \triangleq \{P_m | P_m \neq P(\tau_x) \wedge NS_{x,m}^k(l) - n + 1 > 0\} \cup P(\tau_x)$.*

When $\tau_x$ is blocked by a low priority task upon its arrival, that low priority task may also incur migration cost due to resource access, which in turn delays $\tau_x$. The migration targets of the low priority task are identified directly by the set $\alpha_x^k$ (the set of remote processors with requests that can cause $\tau_x$ to incur arrival blocking) in Equation (18).

In addition, for a given set of migration targets ($mt$) and a resource $r^k$, the set of migration targets with potential preemptors is identified in Equation (20), where $hpt(r^k, P_m)$ gives a set of tasks on processor $m$ that have a priority higher than the resource ceiling of $r^k$.

$$mtp(mt, r^k) \triangleq \{P_m | P_m \in mt \wedge hpt(r^k, P_m) \neq \emptyset\} \tag{20}$$

With migration targets determined, Zhao et al. (2017) firstly presents three obvious situations where no (or limited number of) migrations can occur when a request is issued from processor $P_m$ to resource $r^k$ with a given set of migration targets $mt$, summarised as follows. $Nmig$ denotes the number of potential migrations.

- $Nmig = 0$ if $P_m \notin mtp(mt, r^k)$.
- $Nmig = 0$ if $\{P_m\} = mt$.
- $Nmig = 2$ if $\{P_m\} = mtp(mt, r^k) \wedge |mt| > 1$.

However, in a more general case (i.e., $P_m \in mtp(mt, r^k) \wedge |mt| > 1$), there could exist more than one migration targets with potential preemptors. Thus, the number of migrations is bounded by the releases of all potential preemptors, and each release could cause a preemption to a resource-accessing task, as computed by Equation (21). This equation accounts for the total number of releases of all the potential preemptors on each migration target within the duration of one resource computation time, including the migration time (i.e., $c^k + Mhp(mt, r^k)$). To cope with the situation where the next holder needs to wait for the current holder to migrate away before it can acquire the resource, one extra migration is included.

$$Mhp(mt, r^k) = C_{mig} \cdot \Big( \sum_{P_m \in mtp(mt, r^k)} \big( \sum_{\tau_h \in hpt(r^k, P_m)} \left\lceil \frac{c^k + Mhp(mt, r^k)}{T_h} \right\rceil \big) + 1 \Big) \tag{21}$$

On the other hand, with the NP-section adopted, the migration cost in a single access can also be bounded by the length of the NP-sections, denoted by $Mnp^k$, as given by Equation (22), where $C_{np}$ represents the length of the NP-section.

$$Mnp^k = C_{mig} \cdot \left( \left\lceil \frac{c^k}{C_{np}} \right\rceil + 1 \right) \tag{22}$$

In the case where the holder can be preempted frequently, this equation can give a more acceptable number of migrations that a MrsP resource holder can incur. Unlike Equation (21), this equation does not rely on iterations as the NP-section is for the resource execution only and does not include the cost of migrations. Therefore, $\left\lceil \frac{c^k}{C_{np}} \right\rceil$ can provide a safe bounding on the number of migrations with NP section applied. Combing Equations (21) and (22), Zhao et al. (2017) gives the migration cost bounding for the last situation, where

- $Nmig = min\{Mhp(mt, r^k), Mnp^k\}$ if $P_m \in mtp(mt, r^k) \wedge |mtp(mt, r^k)| > 1$.

Combining the above, Equation (23) gives the complete migration cost bounding that a task can incur. In the worst case, the task has to cope with the migration cost of all the requests in the FIFO queue, including the migration cost of those resource requests, where $Mig(mt, r^k)$ denotes the total migration cost that a task can incur for accessing $r^k$ with a given set of migration targets $mt$.

$$Mig(mt, r^k) = \sum_{P_m \in mt} \begin{cases} 0, & \text{if } P_m \notin mtp(mt, r^k) \vee \{P_m\} = mt \\ 2 \cdot C_{mig}, & \text{if } \{P_m\} = mtp(mt, r^k) \wedge |mt| > 1 \\ min\{Mhp(mt, r^k), Mnp^k\}, & \text{otherwise} \end{cases} \tag{23}$$

This concludes the migration cost analysis. This analysis can be easily integrated into the schedulability test proposed by Zhao et al. (2017) to form a complete migration-aware schedulability analysis for MrsP. The integration is performed by embedding function $Mig(mt, r^k)$ into Equations (16) and Equation (19), as given below.

$$e_x^k(l,\mu) = \sum_{n=1}^{N_x^k(l,\mu)} \left( e_x^k(l)(n) + Mig(mt_x^k(l)(n), r^k) \right) \tag{24}$$

$$\hat{e_i} = max\{|\alpha_i^k| \cdot c^k + Mig(\alpha_i^k, r^k) | r^k \in F^A(\tau_i)\} \tag{25}$$

## 4. Analysing Nested Resource Accesses under MrsP

Section 3 describes the original version of MrsP proposed by Burns and Wellings (2013) and our previous contributions that improve both the definition and schedulability analysis of this protocol. With these efforts, the migration-based helping mechanism is analysable and the definition of supporting nested resource access is complete. However, the schedulability analysis presented in Sections 3.4 and 3.5 is constructed based on the assumption that resource accesses must be non-nested. That is, a task can only access one resource at a given time. However, this assumption imposes strong application restrictions and greatly undermines the practicability of the protocol. In the following sections, this restriction is removed with new extensions for the schedulability analysis proposed to achieve an analysable MrsP system model in wider application scenarios.

The fundamental approach of analysing nested resource accesses is described in (Garrido et al., 2017b), including the techniques of bounding the transitive blocking incurred by inner resources. In this section, new approaches that support the analysis of nested accesses are developed as adaptations to the improved MrsP response time analysis, and aim to provide less pessimistic results than that of the sufficient analysis of nested access presented in Section 3.2.

As described in Section 3.2, a task can incur additional blocking while accessing a resource, say $r^k$, due to requests to $r^k$'s outer resources (which in turn access $r^k$ as an inner resource). In addition, the task can also incur additional transitive blocking while holding $r^k$ due to the access to $r^k$'s inner resources, which are requested by other tasks in the system. Accordingly, the total blocking time of a task for accessing $r^k$ in the nested case is determined as the delay to the concurrent accesses to $r^k$'s outer resources, $r^k$ itself and access requests to $r^k$'s inner resources issued from other tasks in the system. Let $E_x^k(l)(n)$ denote such total blocking in $\tau_x$'s $n$th access to $r^k$ within the duration $l$, and hence Equation (14) and (15) are updated as follows:

$$E_i = \sum_{r^k \in F(\tau_i)} \sum_{n=1}^{N_i^k(R_i,0)} E_i^k(R_i)(n) \tag{26}$$

$$I_{i,h} = \sum_{r^k \in F(\tau_h)} \sum_{n=1}^{N_h^k(R_i,R_h)} E_h^k(R_i)(n) \tag{27}$$

To bound the variable $E_x^k(l)(n)$, we examine the blocking time of $\tau_x$ for accessing $r^k$ and each of $r^k$'s inner resources. When waiting for a resource, $\tau_x$ can incur blocking from the requests to $r^k$'s outer-most resources as well as the requests to $r^k$ itself. One fundamental difference of the nested resource case is the increased number of possible concurrent requests, as a result of migrations. While for the sufficient analysis the general safe upper bound was provided $|V(r^k)| + |map(G(r^k))|$ (see Section 3.2), a more exact bound is required for the improved analysis. For this new analysis, we define $\Gamma(r^k)$ as a set of tasks that can access a resource $r^k$ regardless of the nesting level. This number can be lower than the bounding of $|V(r^k)| + |map(G(r^k))|$. If it is, this value should be used as the safe bound for the maximum number of concurrent access requests $S_{max}^k$ (i.e., the maximum length of the FIFO queue) to such a resource, as shown in Equation (28):

$$S_{max}^k = \begin{cases} |map(G(r^k))| & \text{when } |V(r^k)| = 0 \\ min\{|\Gamma(r^k)|, |V(r^k)| + |map(G(r^k))|\} & \text{otherwise} \end{cases} \tag{28}$$

*Proof.* If the resource has no outer resource, it can only be accessed directly (without nesting) by tasks from their host processors, PCP rules ensuring only one task per processor requests access at a time (i.e., $|map(G(r^k))|$) (Burns and Wellings, 2013). If the resource has outer resources, the concurrent nested access to this resource is bounded by its outer resources due to the mutual exclusion required to those outer resources (Burns and Wellings, 2013; Garrido et al., 2017b). Together with the bounding of direct accesses, accessing a nested resource is safely bounded by $|V(r^k)| + |map(G(r^k))|$. In addition, as the system model does not allow more than one job of a task to be active at a time, there cannot be more resource requests pending so that tasks can issue resource requests up to $(|\Gamma(r^k)|)$ regardless of the nesting level. Thus, a more precise bounding can be constructed, where the maximum blocking due to accessing a nested resource is bounded by $min\{|\Gamma(r^k)|, |V(r^k)| + |map(G(r^k))|\}$ for accessing a nested resource. □

Another difference between nested and non-nested resource accesses is that with nested access, resource requests for a given task $\tau_x$ can be issued from potentially all processors it can migrate to (i.e., being helped). In addition, the migration targets for $\tau_x$ now can be all the processors with tasks requesting any resource that $\tau_x$ may have locked. Consequently, the tasks with resource requests that may delay $\tau_x$ are tracked regardless of their host processors, as given by Equation (29), where $Nr_x^k(l)$ gives the total number of requests to $r^k$

issued by tasks except for $\tau_x$ within the given duration $l$.

$$Nr_x^k(l) = \sum_{\tau_j \neq \tau_x} N_j^k(l, R_j) \tag{29}$$

With $Nr_x^k(l)$ calculated, the resource requests that can cause $\tau_x$ to incur spin delay can be identified by $NS_x^k(l) = (Nr_x^k(l) - Nh_x^k(l) \times S_{max}^k)_0$, where $Nh_x^k(l) \times S_{max}^k$ denotes the requests that block higher priority tasks of $\tau_x$ and should be accounted for as the high priority tasks' interference.

With $S_{max}^k$ and $NS_x^k(l)$ identified, the amount of spin delay for accessing a given resource $r^k$ (without the cost of accessing its inner resources) can be bounded, denoted as $S_x^k(l)(n)$, as shown in Equation (30). This is, from all the contending requests that could lead to spin delay to $\tau_x$ for accessing $r^k$ (i.e., $NS_x^k(l)$), we subtract n-1 times (already performed accesses) the maximum number of concurrent accesses ($S_{max}^k$) minus one (which is the access performed by the analysed task itself), with a minimum of 0 and max of $S_{max}^k - 1$ as the task can be blocked as many times as $S_{max}^k - 1$ (i.e., the FIFO queue length).

$$S_x^k(l)(n) = (NS_x^k(l) - ((n-1) \cdot (S_{max}^k - 1)))_0^{S_{max}^k - 1} \tag{30}$$

*Proof.* As demonstrated for Equation (28), only up to $S_{max}^k$ requests can be issued to a resource $r^k$ at a time. If task $\tau_x$ issues a request, only up to $S_{max}^k - 1$ requests can be ahead in the resource's FIFO queue. Given that $NS_x^k(l)$ returns the maximum number of potential access requests that could directly block a $\tau_x$ request to $r^k$ during a $\tau_x$ activation, Equation (30) safely upper bounds the maximum number of contending access requests to the resource in the $n$th access. □

Let $e_x^k(l)(n)$ be the total amount of time that $\tau_x$ executes with $r^k$, including the time waiting and executing with each $r^k$'s inner resource. With $S_x^k(l)(n)$ bounded, the total blocking time for $\tau_x$ accessing $r^k$ can be determined by Equation (31), where $(S_x^k(l)(n) + 1)$ reflects all potential concurrent access to $r^k$.

$$E_x^k(l)(n) = (S_x^k(l)(n) + 1) \cdot e_x^k(l)(n) \tag{31}$$

Similar to the approach adopted in Equation (10), the accessing time for each $r^k$'s inner resource can be bounded by $\sum_{r^j \in \mathbf{U}(r^k)} \sum_{n=1}^{N_k^j} E_x^j(l)(n)$ iteratively until the inner-most resource is identified, where $\mathbf{U}(r^k)$ returns an empty set.

$$e_x^k(l)(n) = c^k + \sum_{r^j \in \mathbf{U}(r^k)} \sum_{n=1}^{Nr_k^j} E_x^j(l)(n) \tag{32}$$

Thus, the total cost for accessing a resource and its inner resources (including the spin delay and the transitive blocking) can be safely bounded by the above analysis. As for the arrival blocking, the set of resources that could cause

such blocking remains identical to Equation (7). However, as for function $\hat{e}_i$, modifications are required to reflect the transitive blocking for accessing the inner resources. With nested resource allowed, $\hat{e}_i = max\{E_i^k(R_i)(N_i^k + 1)|r^k \in F^A(\tau_i)\}$. Note that there could be the case where $\tau_i$ does not access $r^k$. In this instance, $N_i^k = 0$ so that the remote requests that have not blocked $\tau_i$'s higher priority tasks will be accounted for as the arrival blocking. Otherwise, where $N_i^k > 0$, we track the remote requests that have not blocked $\tau_i$ or its higher priority tasks yet.

*Example*

In order to illustrate the analysis approach, consider the example depicted in Figure 1. The example includes a 3 processor platform with four relevant tasks $\tau_1$ .. $\tau_4$, and two resources accessed by them, $r^1$ and $r^2$. Resource $r^2$ is accessed by $\tau_2$ and $\tau_3$ while holding $r^1$, i.e., in a nested way. The priorities of these tasks are $Pri(\tau_1) = 4$, $Pri(\tau_2) = 3$, $Pri(\tau_3) = 2$ and $Pri(\tau_4) = 1$. That is, $\tau_1$ has the highest execution eligibility among all these tasks while $\tau_4$ has the lowest such eligibility.

Table 4 summarises the relevant task and resource data for the example. A first step to analyse the system is to upper bound the maximum number of concurrent access requests each resource can receive, i.e., $S_{max}^k$, by applying Equation (30). For $r^1$, since it is an outermost resource, this value is equal to $|map(G(r^1))|$ which is the number of processors from where it is directly accessed. Resource $r^2$, on the contrary, is accessed both directly by task $\tau_1$ and $\tau_4$ and in a nested fashion by tasks $\tau_2$ and $\tau_3$. In this case, the number of tasks that access the resource is $\Gamma(r^2) = 4$ but the number of outer serialising entities is just $|V(r^2)| + |map(G(r^2))| = 3$, and, as a result, the maximum number of concurrent access requests to the resource is $S_{max}^{r^2} = 3$. As $\tau_2$ and $\tau_3$ access the resource while having locked $r^1$ first, only one of them can issue an access request at a time.

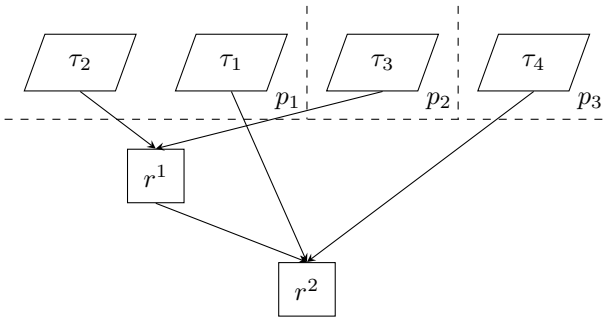


Figure 1: Graphical representation of resource usage in running example.

To compute the response time of $\tau_1$, its access cost to $r^2$ ($E_{\tau_1}^{r^2}$) is calculated. To do so, Equation (29) is used to identify the number of accesses issued by other tasks to the resource. This is, applying $N_x^k(l, \mu)$ for each other task:

Table 4: Relevant task parameters and resource usage of running example.

| | | | | $r^1$ | | $r^2$ | |
|---|---|---|---|---|---|---|---|
| Task | P | T | C | N | c | N | c |
| $\tau_1$ | 1 | 50 | 5 | | | 1 | 2 |
| $\tau_2$ | 1 | 60 | 3 | 3 | 1 | 3 | 2 |
| $\tau_3$ | 2 | 50 | 4 | 1 | 1 | 1 | 2 |
| $\tau_4$ | 3 | 40 | 3 | | | 1 | 2 |

| Resource | $\Gamma$ | $\lvert V\rvert$ | $\lvert map(G)\rvert$ | $S_{max}$ |
|---|---|---|---|---|
| $r^1$ | 2 | 0 | 2 | 2 |
| $r^2$ | 4 | 1 | 2 | 3 |

$\lceil \frac{5+3}{60} \rceil \cdot 3 = 3$ for $\tau_2$, $\lceil \frac{5+4}{50} \rceil \cdot 1 = 1$ for $\tau_3$ and $\lceil \frac{5+3}{40} \rceil \cdot 1 = 1$ for $\tau_4$, resulting in a total of $Nr^{r^2}_{\tau_1}(R_{\tau_1}) = 5$. As $\tau_1$ does not have any local higher priority task, i.e., $Nh^{r^2}_{\tau_1}(R_{\tau_1}) = 0$. Consequently $NS^{r^2}_{\tau_1}(R_{\tau_1}) = 5$, meaning that all remote accesses are to be considered, since they have not been accounted for yet. Then, for the first and only access to $r^2$, the potential number of tasks that can cause direct spin delay to $\tau_1$ according to Equation (30) is $S^{r^2}_{\tau_1}(R_{\tau_1})(1) = (5 - ((1-1)\cdot(3-1)))^{3-1}_0 = 2$, i.e., one direct access from $p_3$ and a nested access from $r_1$. Since $r^2$ does not have any inner resource, its access time $e^{r^2}$ is equal to its execution time $c^{r^2}$, i.e., 2. Then we can substitute $e^{r^2}$ access time in Equation (31) to obtain the final access cost to $r^2$ as $E^{r^2}_{\tau_1}(R_{\tau_1})(1) = (2+1)\cdot 2 = 6$. This is to be added to the base execution time of 5 units to obtain a preliminary response time of 11 time units.

Then, response time of $\tau_2$ can be addressed. As shown in Table 4, it accesses 3 times to $r^1$ and $r^2$ in a nested way. Then, for each access to $r^1$, the cost of accessing $\tau_2$ is calculated first. The number of accesses to $r^2$ from other tasks is calculated via $N^k_x(l,\mu)$ yielding a value of $\lceil \frac{3+5}{50} \rceil \cdot 1 = 1$ for $\tau_1$, $\lceil \frac{3+4}{50} \rceil \cdot 1 = 1$ for $\tau_3$ and $\lceil \frac{3+3}{40} \rceil \cdot 1 = 1$ for $\tau_4$, and a total of $Nr^{r^2}_{\tau_2}(R_{\tau_2}) = 3$. In contrast to $\tau_1$ analysis, $\tau_2$ presents higher priority tasks accessing $r^2$ ($\tau_1$). This number of accesses is $Nh^{r^2}_{\tau_2}(R_{\tau_2}) = N^{r^2}_{\tau_1}(R_{\tau_2}) = \lceil \frac{3+5}{50} \rceil \cdot 1 = 1$. As the resource has already been accessed by a higher local priority task, the direct delay to be accounted for due to remote tasks can be reduced: $NS^{r^2}_{\tau_2}(R_{\tau_2}) = 3 - (1 \cdot 3) = 0$. With this value, the access to $r^2$ does not need to account for any direct spin delay according to Equation (30): $S^{r^2}_{\tau_2}(R_{\tau_2})(1) = (0 - ((1-1)\cdot(3-1)))^{3-1}_0 = 0$ and thus its access cost is equal to its execution time, $E^{r^2}_{\tau_2}(R_{\tau_2})(1) = (0+1)\cdot 2 = 2$. With this value, the access cost to $r^1$ can be calculated. The resource only has one other task accessing the resource, $\tau_3$: $Nr^{r^1}_{\tau_2}(R_{\tau_2}) = N^{r^1}_{\tau_3}(R_{\tau_2}) = \lceil \frac{3+4}{30} \rceil \cdot 1 = 1$, and no local higher priority task accesses the resource, i.e., $Nh^{r^1}_{\tau_2}(R_{\tau_2}) = 0$, resulting in a number of potential spin delay remote accesses of $NS^{r^1}_{\tau_2}(R_{\tau_2}) = 1 - (0 \cdot 2) = 1$. Then, for the first access, this direct spin delay is accounted for according by Equation (30): $S^{r^1}_{\tau_2}(R_{\tau_2})(1) = (1 - ((1-1)\cdot(2-1)))^{2-1}_0 = 1$. Being the access

time including the inner resource $e^{r^1}_{\tau_2}(R_{\tau_2})(1) = 1 + 2 = 3$, the total access cost for the first access is $E^{r^1}_{\tau_2}(R_{\tau_2})(1) = (1 + 1) \cdot 3 = 6$. Successive accesses do not have to account for any direct spin delay as the potential access from $\tau_3$ has been already accounted for. As a result, accesses 2 and 3 have an access cost of $E^{r^1}_{\tau_2}(R_{\tau_2})(2, 3) = (0+1) \cdot 3 = 3$ each. Then, the total direct access cost to shared resources is the sum of the three accesses $E_{\tau_2} = E^{r^1}_{\tau_2}(1) + E^{r^1}_{\tau_2}(2) + E^{r^1}_{\tau_2}(3) = 6 + 3 + 3 = 12$. With this value, a preliminary response time of 15 time units can be used for further calculations.

The analysis of $\tau_2$ response time needs to be completed with the study of the interference suffered due to $\tau_1$. This interference includes $\tau_1$ execution time plus the indirect spin delay caused by $\tau_1$, as identified in Equation (27). The first step is to calculate the number of accesses that $\tau_1$ can issue during an activation of $\tau_2$, including the back to back hit. This is calculated as $N^{r^2}_{\tau_1}(R_{\tau_1}, R_{\tau_2}) = \left\lceil \frac{11+15}{50} \right\rceil \cdot 1 = 1$. Then, the indirect spin delay to account for is the only access of $\tau_1$ that was already identified to be of $E^{r^2}_{\tau_1}(R_{\tau_1}) = 6$ time units, and a final $\tau_2$ response time of $R_{\tau_2} = C_{\tau_2} + E_{\tau_2} + \left\lceil \frac{R_{\tau_2}}{T_{\tau_1}} \right\rceil \cdot C_{\tau_1} + I_{\tau_1} = 3 + 12 + 5 + 6 = 31$

The last step to complete the analysis of tasks on processor $P_1$ is to compute the arrival blocking suffered by $\tau_1$. The only resource identified by function $F^A_{\tau_1}$ is $r^2$, as is the only resource accessed by both $\tau_1$ and a lower priority task. The arrival blocking is calculated as the cost of the $E^k_i(R_i)(N^k_i + 1)$ access to the resource, i.e., $B_{\tau_1} = E^{r^2}_{\tau_1}(R_{\tau_1})(2)$. The number of remote access requests that can interfere with that of this second access to the resource is recalculated with the updated response times, being $\left\lceil \frac{11+31}{60} \right\rceil \cdot 3 = 3$ for $\tau_2$, $\left\lceil \frac{11+4}{50} \right\rceil \cdot 1 = 1$ for $\tau_3$ and $\left\lceil \frac{11+3}{40} \right\rceil \cdot 1 = 1$ for $\tau_4$, resulting in a total of $Nr^{r^2}_{\tau_1}(R_{\tau_1}) = 5$. As there are no higher priority tasks accessing the resource $Nh^{r^2}_{\tau_1}(R_{\tau_1})$ is still equal to 0. The number of potential remote contenders for the resource is calculated as $S^{r^2}_{\tau_1}(R_{\tau_1})(2) = (5 - ((2 - 1) \cdot (3 - 1)))^{3-1}_0 = 2$ and so the access cost is equal to $E^{r^2}_{\tau_1}(R_{\tau_1})(2) = (2 + 1) \cdot 2 = 6$, resulting in a final response time of $R_{\tau_1} = C_{\tau_1} + E_{\tau_1} + B_{\tau_1} = 5 + 6 + 6 = 17$.

Processors $P_2$ and $P_3$ present a simple analysis, since they only host one task each. Following the same procedures as those presented previously, an access cost for $\tau_3$ to $r_1$ and $r_2$ can be calculated as $E^{r^1}_{\tau_3}(R_{\tau_3})(1) = 14$ yielding a response time of $R_{\tau_3} = C_{\tau_3} + E_{\tau_3} = 4 + 14 = 18$. Similarly, for $\tau_4$ in $p_3$ an access cost of $E^{r^2}_{\tau_4}(R_{\tau_4})(1) = 6$ and a response time of $R_{\tau_4} = C_{\tau_4} + E_{\tau_4} = 3 + 6 = 9$. With these results, any pair of response times is bigger than the period of any of the two tasks and consequently the analysis is finished.

## 5. A complete Run-time Cost Analysis for MrsP

Section 3.5 describes our first attempt to include run-time overhead (more precisely, the cost of migrations) of MrsP system into mathematical schedulability analysis, based on the assumption of non-nested accesses. As this assumption is removed in this paper for more realistic application scenarios, the migration-cost analysis should be revised to cope with the case where tasks hold multiple

resources at the same time. In addition, as shown in (Burns and Wellings, 2016), context switches introduce non-trivial run-time overhead and manipulating locks also imposes extra costs, which should not be ignored for any form of run-time cost-aware tests. In this section, the migration cost analysis is extended to support applications with nested resource access and to include the above cost.

### *5.1. Cost of Migrations under Nested resources*

Similar to the non-nested case, the number of migrations during nested resource access directly depends on the amount of time during which a resource is locked. However, two differences need to be taken into account. Firstly, a task having locked a non-outermost resource can migrate not only to processors from where the locked resource can be accessed directly, but also to those from where its outer resources can be accessed. In other words, a task can migrate to any processor where a task accesses the resource regardless of the nesting level.

$$mt(r^k) \triangleq \{P_m | P_m \in map(\Gamma(r^k))\} \tag{33}$$

Note that Equation (33) also includes the host processor of task $\tau_x$ under analysis. Recall that, with nested resources, it is perfectly possible for a task to be migrated to a remote processor when accessing a $r^k$ outer resource (say $r^j$) and while accessing $r^j$ on that remote processor access $r^k$ and require help. During the time $\tau_x$ was migrated, another task $\tau_y$ satisfying $Pri(r^j) < Pri(\tau_y) \leq Pri(r^k)$ could have been released and spinning for access to $r^k$, making $\tau_x$ host a valid migration target.

The second difference is due to the transitive helping rule, a resource can be helped with the priority of an outer resource rather than its local ceiling priority. Following general PCP rules, an outer resource must have an equal or lower priority than any inner resource accessed. As a result, the priority to be considered for migration calculation purposes on each helping processor is the lowest of those outer resources on that processor (if any), denoted as $lcp_m^k$ in Equation (34). If resource $r^k$ is an outermost resource on that processor, its $lcp$ is equal its local ceiling priority.

$$lcp_m^k = \min_{r^y \in V(r^k) \wedge P_m \in map(G(r^y))} \{Pri(r^y)\} \tag{34}$$

This function is then used to redefine the calculation of migration targets with preemptors ($mtp(mt, r^k)$ in Equation (20)) as:

$$mtp(mt, r^k) \triangleq \{P_m | P_m \in mt \wedge hpt(lcp_m^k, P_m) \neq \emptyset\} \tag{35}$$

Finally, the migration cost bounded by the number of preemptor releases is refined as that can be used to finally obtain the migration cost by also redefining Equation (21) as:

$$Mhp(mt, r^k) = C_{mig} \cdot \Big( \sum_{P_m \in mtp(mt, r^k)} \big( \sum_{\tau_h \in hpt(lcp_m^k, P_m)} \left\lceil \frac{c^k + Mhp(mt, r^k)}{T_h} \right\rceil \big) + 1 \Big) \tag{36}$$

Alternatively, if non-preemptive sections are implemented, the cost of migrations can also be calculated using Equation (22). The final value $Mig(mt, r^k)$ to be taken into account for the analysis is still defined by Equation (23). This value is integrated in the accessing time to a nested resource with migrations accounted for.

$$e_x^k(l)(n) = c^k + Mig(map(\Gamma(r^k)), r^k) + \sum_{r^j \in \mathbf{U}(r^k)} \sum_{n=1}^{N_k^j} E_x^j(l)(n) \tag{37}$$

*Example*

Consider the resource access graph of the example provided Section 4, and let the taskset be enriched with tasks $\tau_x$, $\tau_y$ and $\tau_z$ with parameters presented in Table 5. With this newly considered tasks, the migration cost analysis of $r^2$ would be as follows. The first step is to consider the potential migration targets of the resource by applying Equation (33). This, for $r^2$ is all processors, including $p_2$, as, although not directly accessed, it is an inner resource of $r^1$ accessed by $\tau_3$. The second step is to consider on which processors it has preemptors, using Equations (34) and (35). Following Equation (34), $lcp_1^2$ and $lcp_2^2$ are equal to 2, as is the lowest priority of its outer resources on each processor. In processor $p_3$, the value of $lcp_3^2$ its, 1, equal to its local ceiling priority. Then applying Equation (35), $mtp(mt, r^2) = p_1, p_3$ as on those processors there is at least a task with higher priority than $lcp^2$ on that processor. Finally, we compute the migration cost (lets consider 0.1 as the cost of each migration $C_{mig}$): on the first iteration $M_{hp_0} = 0.1 \cdot (\lceil \frac{2+0}{40} \rceil) + \lceil \frac{2+0}{25} \rceil) = 0.2$. Then, feeding back the $M_{hp}$ value in the iteration $M_{hp_1} = 0.1 \cdot (\lceil \frac{2+0.2}{40} \rceil) + \lceil \frac{2+0.2}{30} \rceil) = 0.2$ that is the final value to be considered for tasks allocated to $p_1$ and $p_3$. Note that $\tau_3$ does not suffer migration costs as, as mentioned in section 3.5, $\tau_3$ cannot be locally preempted. As denoted by Equation (37) this value is to be added to the resource access time in order to obtain $e^2$ value $e^2 = 2 + 0.2 = 2.2$.

Table 5: Relevant task parameters and resource usage of enriched running example.

| | | | | | $r^1$ | | $r^2$ | |
|---|---|---|---|---|---|---|---|---|
| Task | P | Pri | T | C | N | c | N | c |
| $\tau_x$ | 1 | 3 | 40 | 5 | | | | |
| $\tau_1$ | 1 | 2 | 50 | 5 | | | 1 | 2 |
| $\tau_2$ | 1 | 1 | 60 | 3 | 3 | 1 | 3 | 2 |
| $\tau_3$ | 2 | 2 | 50 | 4 | 1 | 1 | 1 | 2 |
| $\tau_y$ | 2 | 1 | 60 | 5 | | | | |
| $\tau_z$ | 3 | 2 | 25 | 5 | | | | |
| $\tau_4$ | 3 | 1 | 30 | 3 | | | 1 | 2 |

### *5.2. Costs of Context Switches and Protocol Implementation*

As described in (Burns and Wellings, 2016), the major run-time costs that tasks incur when using a resource sharing protocol include the cost of obtain-

ing and releasing a lock (i.e., the time required for executing the `lock()` and `unlock()` functions), and the context switches due to task releases and preemptions. Figure 2 illustrates the major scheduling events occurring in the underlying operating system during the lifetime of a periodic task's release (say $\tau_i$).

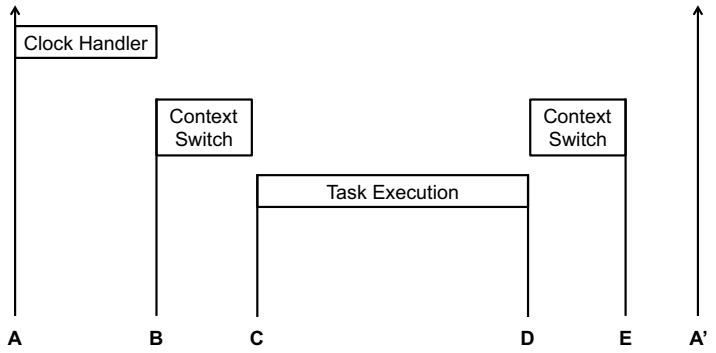


Figure 2: Events from the Operating System During a Task's Release (Burns and Wellings, 2016).

When $\tau_i$'s release time arrives, the corresponding clock interrupt will be fired and the interrupt handler will move $\tau_i$ from the sleeping queue to the ready queue, where it waits to be scheduled (i.e., event A). Assuming $\tau_i$ has the highest priority among all the ready tasks, the scheduler will be invoked to release $\tau_i$ (i.e., event B). If there is an executing task, this task will be switched away. $\tau_i$ starts its execution at event C and finishes at event D, during which it could be preempted several times by newly-released higher priority tasks. When $\tau_i$ is finished, it will be cleaned up and switched away by the scheduler (event E). Then, the system schedules the next ready task to execute (if any) and keeps waiting for the next clock interrupt (i.e., event A'). If $\tau_i$ is preempted during the interval C to D, it incurs the overhead from all events given in Figure 2.

According to the description above, to account for the cost due to the potential context switches $\tau_i$ can suffer during each release, Equation (13) is extended to Equation (38), as given below.

$$R_i = CX_1 + C_i + E_i + B_i + \sum_{\tau_h \in \mathbf{hpl}(i)} \left( \left\lceil \frac{R_i}{T_h} \right\rceil \cdot (CX_2 + C_h) + I_{i,h} \right) \tag{38}$$

where $CX_1$ denotes context switch overhead associated with $\tau_i$ itself. Note, $CX_1$ only includes the costs for events A and B as $\tau_i$ finishes its execution at even D. That is, only the cost of events A and B will occur during its release, assuming no preemptions. If $\tau_i$ is preempted while executing, it will incur extra overhead caused by the events A, B and E, which is denoted as $CX_2$. The reason to include event E here is that $\tau_i$ has to cope with the associated cost for switching the preemptor away before being resumed. With these two variables

determined, the run-time overhead incurred by $\tau_i$ due to major scheduling events from the underlying system can be bounded.

The cost for obtaining and releasing a MrsP lock mainly includes the overhead for raising and restoring the priorities of the resource accessing tasks (Zhao and Wellings, 2017), and manipulating the FIFO queues, which are performed in the function `lock()` and `unlock()`. Such costs are denoted as $C^{lock}_{MrsP}$ and $C^{unlock}_{MrsP}$ respectively, where they can be easily integrated into the cost for accessing a resource via a new notation $C^k$, as given below.

$$C^k = C^{lock}_{MrsP} + c^k + C^{unlock}_{MrsP} \tag{39}$$

Accordingly, notation $c^k$ is replaced by $C^k$ in Equations (17), (19), (21) and (32) to incorporate the overhead of the locking protocol. However, note that $c^k$ in Equation (22) remains as the NP-section is applied only inside the critical section. With the above equations, the run-time overhead incurred by tasks in MrsP systems due to the underlying operating system and the protocol implementation can be bounded. Note that the above equations only provide an overall approach for incorporating the run-time overhead.

To complete this analysis, the underlying hardware and a real-world operating system must be provided and the cost for each event in the worst case should be measured to provide a safe upper bound. The exact measuring approach of $CX_1$ and $CX_2$ largely depends on the scheduling structure of the given operating system while the costs of `lock()` and `unlock()` depends on the real implementation of the protocol. In Section 7, the above run-time cost variables are measured under the Litmus$^{RT}$ Real-Time Operating System (Calandrino et al., 2006a; Brandenburg, 2011) based on a MrsP implementation realised in the Preemptable-Fixed Priority scheduler for fully-partitioned systems.

This concludes our extensions to MrsP schedulability analysis. Combining the response time analysis and the migration cost analysis together, we provide an improved and more complete schedulability analysis tool for MrsP with the awareness of implementation and run-time costs, which is capable for analysing systems with the presence of nested resource accesses.

## 6. NP-Section Length Configuration

As described by Zhao et al. (2017), the number of migrations, and thus its costs, can be limited if a short non-preemptive Section is enforced after a migration. As discussed in Section 3.3 and formalised in Equation (11), the length of the NP-section might affect the response time of tasks, constituting the local blocking term ($B_i$). In consequence, the $C_{np}$ length configuration can have an impact on the protocol performance.

In (Zhao et al., 2017), an initial analytic approach to the $C_{np}$ length setting is given. In particular, the constant, platform-dependent $\hat{b}$ value can be safely used as an initial $C_{np}$ value. In this section, a complete set of recommendations for NP-section configuration is proposed to achieve further schedulability improvement, which is not possible via the simplistic setting of $C_{np}$ to $\hat{b}$.

From an analytic perspective, the objective is to reduce the number of potential migrations without increasing task response time. That is, reduce $E$ and $I$ values without increasing $B$ for any task in the system. As suggested in (Zhao et al., 2017), $\hat{b}$ can be safely used as $C_{np}$ as now demonstrated in Theorem 4.

**Theorem 4.** *No task local blocking time is increased when $C_{np}$ is set to $\hat{b}$, i.e., $C_{np} = \hat{b}$.*

*Proof.* By definition of $B_i = max\{\hat{e_i}, n\hat{p}_i, \hat{b}\}$. As $n\hat{p}_i$ can take only two values $[0, C_{np}]$, $B_i$ is independent of $C_{np}$ as long as $C_{np} \leq \hat{b}$. □

While Theorem 4 gives a safe value for $C_{np}$, it can provide a little benefit on systems where $\hat{b}$ is trivial to critical section length. This raises the interest of deriving $C_{np}$ configurations based on the length of shared resources.

**Theorem 5.** *Task local blocking time will not increase when $C_{np}$ is set so that $C_{np} = \{\min(B'_i) | Pri(\tau_i) \geq min_{\{r^k \text{ is global}\}} Pri(r^k, P(\tau_i))\}$, where $B'_i = max\{\hat{e_i}, 0, \hat{b}\}$.*

*Proof.* Condition of $C_{np}$ construction limits $B'_i$ consideration to only those tasks affected by $C_{np}$. Then, if the minimum value of local blocking of those tasks without considering $n\hat{p}_i$ is taken as $C_{np}$, it follows from $B_i$ and $n\hat{p}_i$ definition that $max\{\hat{e_i}, n\hat{p}_i, \hat{b}\} \leq max\{\hat{e_i}, 0, \hat{b}\}, \ \forall \tau_i$. □

Note that $C_{np}$ obtained from Theorems 4 and 5 can take the same value if for a given task $\tau_i$ that might be affected by the non-preemptive section it holds that $\hat{e_i} \leq \hat{b}$, i.e., the non-preemptive section imposed by the platform is longer than the access time of potentially local blocking resources. In any case, the resulting $C_{np}$ of applying Theorem 5 is optimal without increasing local blocking values of any task.

*Proof.* Suppose Theorem 5 does not provide an optimal $C_{np}$ with the restriction of not increasing local blocking values of any tasks. Then there should exist a $C^+_{np}$ value satisfying $C^+_{np} > C_{np}$. Then, for the task $\tau_x$ yielding the $min(B')$ value on Theorem 5 $C'_{np} > max\{\hat{e_i}, \hat{b}\}$ and hence $B_x > B'_x$. □

Values for $C_{np}$ obtained from Theorems 4 and 5 might not be sufficient to make the system schedulable, or provide a sufficient margin for certain tasks within systems including this requirement. In this case, the empirical approach shown in Algorithm 1 can be applied, at the cost of potentially increasing the local blocking of certain tasks and thus their response times. Note that this can make the latter tasks unschedulable. In consequence, the algorithm is only intended to be used when the system does not already meet its temporal requirements.

The algorithm initially sets as a first tentative $C_{np}$ value the one proposed in Theorem 5. Then this value is gradually increased until either a value for $C_{np}$ is found allowing every task to meet its temporal requirements or the system is finally deemed unschedulable.

**Algorithm 1** $C_{np}$ empiric configuration.

1: do_RTA;
2: $C_{np} \leftarrow \{\min(B'_i) | Pri(\tau_i) \geq \min_{\{r^k \text{ is global}\}} Pri(r^k, P(\tau_i))\}$, where $B'_i = max\{\hat{e_i}, 0, \hat{b}\}$
3: **loop**
4: **if** System_Unschedulable **then**
5: **if** $\{R_i \leq D_i \vee Mig(\tau_i) \geq R_i - D_i | \forall \tau_i\}$ **then**
6: $C_{np} \leftarrow \min\{C_{np}^{new} | \lceil \frac{c^k}{C_{np}^{new}} \rceil = \lceil \frac{c^k}{C_{np}^{old}} \rceil - 1 \wedge r^k \in G(\text{Unsched\_Tasks})\}$
7: **if** $C_{np}^{new} = C_{np}^{old}$ **then**
8: **return** System_Unschedulable;
9: **else**
10: do_RTA;
11: **end if**
12: **else**
13: **return** System_Unschedulable;
14: **end if**
15: **else**
16: **return** System_Schedulable;
17: **end if**
18: **end loop**

The algorithm prerequisite (necessary, but not sufficient) is that, for every task in the system, its initial response time is either lower or equal than its deadline (the tasks meets its temporal requirements), or the sum of the migration costs suffered by the task (denoted by $Mig(\tau_i)$) is greater than the response time reduction needed by the task to meet its deadline, i.e., reducing the task migration costs can actually make the task schedulable.

Then a new tentative value $C_{np}^{new}$ is selected. This new value is the minimum that reduces the number of potential migrations that can happen during the access to a resource by one. Only resources used by at least one task not meeting its temporal requirements ($G$(Unsched_Tasks)) are considered. If no new value is found then the system cannot be made schedulable by $C_{np}$ tuning. Otherwise, the response time analysis is reconducted, and system schedulability rechecked. If the system is found to be schedulable, the algorithm ends. Elseways, the process is repeated.

The presented algorithm efficiently checks every possible $C_{np}$ value in the solution space. Only $C_{np}$ values that can actually improve response times of tasks not meeting their temporal requirements are checked. Any other values would increase arrival blocking times without reducing worst-case migration costs of unschedulable tasks. By finding, if any, the minimum $C_{np}$ value making the system schedulable, the algorithm provides the lowest possible local blocking to higher priority tasks by construction. Finally, it should be noted that the algorithm complexity only depends on the number of shared resources in the

system and the number of their associated access time divisors greater than the initial $C_{np}$ value. To this end, the time complexity of this algorithm can be determined. Let $W$ denote the non-polynomial time complexity of the schedulability test (Audsley, 2001), the time complexity of this algorithm is bounded by $O(W \cdot \sum_{\forall \tau^k} \left\lceil \frac{c^k}{C_{np}^{init}} \right\rceil)$, where $C_{np}^{init}$ denotes the initial length of the NP-section provided by the algorithm.

## 7. Evaluation

In this section, we investigate the efficiency of the proposed schedulability analysis extension for nested resources and evaluate the performance of this protocol under the studied resource accessing model with various system settings (e.g., the degree of parallelism, the frequency of resource access and the length of critical sections). To achieve this, a set of experiments are conducted to investigate (1) the schedulability of the original test and the new test of MrsP; (2) the schedulability of MrsP and the other major FIFO spin-based locking protocols (i.e., MSRP and PWLP reviewed in Section 2.2); (3) the impact of the run-time overhead to the schedulability results and (4) the computation costs of the proposed schedulability analysis itself and the ILP-based analysis (see Section 2.3). Schedulability tests proposed in the above sections have been implemented for the experiments conducted in this section and are accessible via `https://github.com/RTSYork/SchedulabilityTestEvaluation`.

To compare the results of the schedulability tests for MrsP developed in this paper and the tests for MSRP and PWLP, the ILP-based analysis from the SchedCAT project (Brandenburg, 2013b; Biondi et al., 2016) is integrated into the testing program via JNI. Note, the run-time overhead for MSRP and PWLP is accounted for by approaches similar to those described in Section 5.2 in the ILP-based analysis, which directly supports the analysis of MSRP and PWLP with minimised pessimism (i.e., they guarantee that each critical section will be accounted for only once). To achieve fair comparison, a system generation tool (Zhao et al., 2017; Zhao, 2018) is developed to generate random systems with various application semantics and resource characteristics configurations (but within given boundaries). The algorithms and configuration settings applied in the generation tool is described below. This tool firstly generates tasks that conform to the sporadic model and a set of resources, then the resource usage is produced based on the given parameter settings.

The experimental setup for investigating the schedulability tests in this paper is described as follows, and covers a wide range of system settings in real-time automotive and safety-critical applications (Fürst et al., 2009; Cavalcanti et al., 2016; Garrido et al., 2015; Buttle, 2012). We consider platforms with $M = [2, 24]$ processors, where systems with $M \leq 8$ are widely available nowadays while $M > 8$ gives the forward-looking scenario. The system contains $n$ tasks with a total utilisation $U$ and $U = 0.1n$, where $n$ denotes the number of tasks in the system. Periods of tasks on each processor are given randomly between $[1ms, 1000ms]$ in a log-uniform distribution fashion. Deadlines of tasks are set

to be constrained (i.e., equal to their periods, where $D_x = T_x$ for $\tau_x$). The utilisation of each task is given based on the UUnifast-Discard algorithm (Bini and Buttazzo, 2005; Emberson et al., 2010). With task utilisation obtained, the total computation time (denoted as $C'_x$ for $\tau_x$) for each task can be computed, where $C'_x = U_x \times T_x$. Note, $C'_x$ is the sum of the pure computation time $C_x$ and the total resource computation time $C^r_x$ (i.e., the time $\tau_x$ spends on executing with each resource it requests), where $C'_x = C_x + C^r_x$. The system supports 1000 priority levels. The priorities of the tasks in a given system is assigned via the DMPO algorithm (Leung and Whitehead, 1982) prior to allocation, which assigns a higher priority to a task with a shorter deadline. At last, tasks are allocated to each processor via the Worst-Fit heuristic (Johnson, 1973), which allocates each task to the processor that has the lowest utilisation.

In addition, similar to the evaluation settings applied in (Wieder and Brandenburg, 2013; Zhao et al., 2017), tasks in each system share $M$ resources. For each resource, a wide range of critical section lengths ($L$) is supported, including $[1\mu s, 15\mu s]$, $[15\mu s, 50\mu s]$, $[50\mu s, 100\mu s]$, $[100\mu s, 200\mu s]$, $[200\mu s, 300\mu s]$ and $[1\mu s, 300\mu s]$. In addition, a real value parameter $\kappa$ is introduced to specify the number of tasks on each processor that can have access to resources (i.e., $\lfloor \kappa \cdot allocated\_tasks \rfloor$), where $\kappa \in$ [0.0,1.0]. Once a task is set to access a resource, it can issue requests to a number of randomly chosen resources, but limited by $[1, M]$. To control resource access frequency, the number of requests that a task can issue to a resource is randomly decided between $[1, A]$, where $A$ takes value ranged from 1 to 41. Such settings are sufficient to cover most scenarios in practice (Wieder and Brandenburg, 2013; Fürst et al., 2009). Once $\tau_x$'s resource usage is generated, the total resource computation time $C^r_x$ can be obtained so that $C_x$ can also be computed, where $C_x = C'_x - C^r_x$ ($C^r_x = 0$ for tasks that do not access any shared resources). We enforce that $C'_x - C^r_x \geq 0$. There exists a large number of possible combinations of the system settings with variables given above (i.e., $n$, $M$, $L$ and $A$). In the interest of brevity, we only present experiments that effectively demonstrate the main trends and performance difference between evaluated schedulability tests.

### *7.1. Schedulability Comparison*

This section investigates and compares the schedulability performance of the selected protocols in the presence of nested resources. In particular, the protocol versions considered are the original MSRP analysis (denoted as "MSRP-original") for nested resources and the ILP-based nested analysis presented in (Biondi et al., 2016) for both MSRP and PWLP ("MSRP-new" and "PWLP-new"), the original MrsP analysis ("MrsP-original") for nested resources by Garrido et al. (2017b) and the newly presented MrsP improved analysis for nested resources ("MrsP-new"). Protocols with the label "new" indicates their associated state-of-art schedulability tests (ILP-based analysis for MSRP, PWLP and our new analysis for MrsP) while "original" denotes the early analysis described in Section 2. As with in (Wieder and Brandenburg, 2013; Zhao et al., 2017), 1000 systems are generated for each combination of system settings. To generate nested resource accesses the following process has been adopted: after

all resources have been generated, the use of inner resources is calculated in order of creation. To this end, in order to avoid circular dependencies, a task holding a resource can only access another resource that is generated later. The probability of accessing each inner resource is 20%. The number of times that a task can access an inner resource is also defined by $A$.

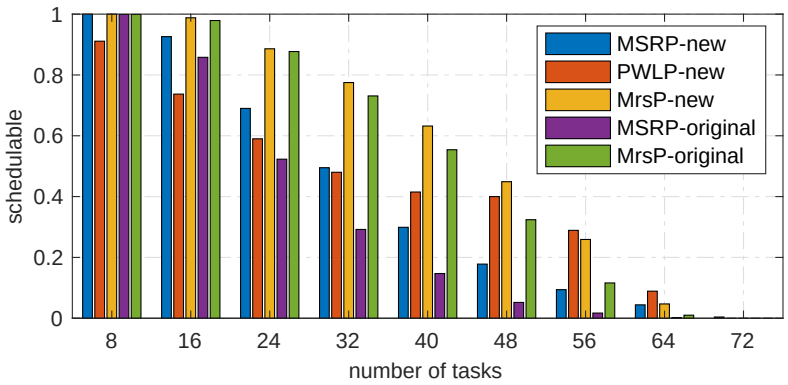


Figure 3: Schedulability in the presence of nested resources for $M = 8$, $U = 0.1n$, $\kappa = 0.4$, $A = 2$, $L = [50\mu s, 100\mu s]$, and $M$ Shared Resources.

(a) *Increasing workload n:* With a realistic number of processors ($M = 8$), low resource contention (the number of accesses to the same resource is limited to $A = 2$ and a resource sharing factor of $\kappa = 0.4$) and medium critical section length ($L = [50\mu s, 100\mu s]$), MrsP presents a better schedulability in general terms with the improved analysis, as shown in Figure 3. Firstly, similar with the observation obtained in (Zhao et al., 2017), the state of art schedulability tests of both MSRP and MrsP provide better results than that of their original tests in all cases, which indicates that the new schedulability tests remain less pessimistic in the presence of nested resources. With the increase of $n$, the original MSRP schedulability test presents the most pronounce fall as a consequence of its higher arrival blocking regardless of the analysis employed, due to its non-preemptive nature. The original test of MrsP, on the contrary, presents a similar trend to that of the new analysis for PWLP and MrsP until almost 60% of system workload (48 tasks in the system), as the inter-dependency between tasks is still low to reflect the pessimism in the analysis.

As for the new schedulability tests, both MSRP and MrsP outperform PWLP with $n < 40$ (i.e., with a low schedulability pressure) due to the extra resource-waiting time introduced into by the cancellation mechanism. However, with $n \geq 32$, the schedulability of MSRP decreases significantly and is outperformed by both MrsP and PWLP due to its non-preemptive approach, which leads to prolonged arrival blocking. As for MrsP, it provides the best schedulability in most cases with the priority ceiling approach, which achieves the $|map(G(r^k))|$ bounding as MSRP theoretically (i.e., without run-time overhead) and has a much lower arrival blocking. Interestingly, PWLP presents a leaner

curve, somewhat different from the non-nested results given in (Zhao et al., 2017). This is the result of the difference between grouping and non-grouping resources: in the cases where the workload is low, and thus the total number of accesses to shared resources, the reduced parallelism of the resource grouping approach present in PWLP highly affects the overall system performance compared to the fine-grained nesting of MSRP and MrsP. It is also relevant to note that PWLP presents better results on the higher end (above 70% of utilisation). This result is somehow expected since, as the overall utilisation of resources increases, it is more likely that resource access requests to inner resources cannot be directly satisfied under fine-locking approaches, where each resource is protected by its own lock and tasks have to compete with each other when accessing an inner resource. However, with group locks, such resource contention is described as the access to an inner resource is granted as long as its outer resource is being held by the task.

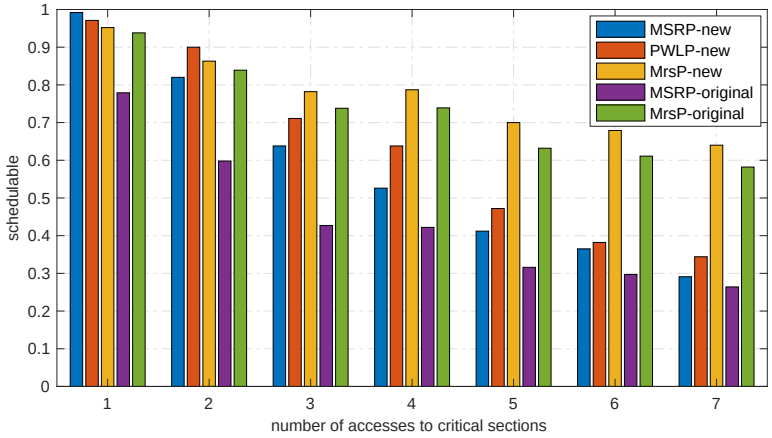


Figure 4: Schedulability for $M = 8$, $n = 32$, $U = 0.1n$, $\kappa = 0.4$, $L = [15\mu s, 50\mu s]$, and $M$ Shared Resources.

(b) *Increasing resource contention A:* As with the non-nested case in (Zhao et al., 2017), PWLP under the nested case has similar results with a low resource access frequency, where it demonstrates a strong schedulability and outperforms other protocols with $A = 2$. Again this result is a consequence of the trade-off between arrival blocking and the cost of being preempted while busy-waiting of PWLP (requiring to requeue the request). However, as the number of times a resource can be accessed increases, the number of times a lower priority task can be preempted while busy-waiting is increased and, as a consequence, the time spent occupying the processor without making progress is increased. A similar analysis can be derived from the MSRP results: while it is the best performing protocol with the ILP analysis with low number of accesses to critical sections (and thus probabilities of higher priority tasks suffering arrival blocking), as tasks increase their use of critical sections the arrival blocking becomes a limiting

factor, highly reducing the protocol performance. As for MrsP, it demonstrates strong schedulability in this experiment due to the priority ceiling approach (which yields limited arrival blocking compared to MSRP) without no extra blocking period.

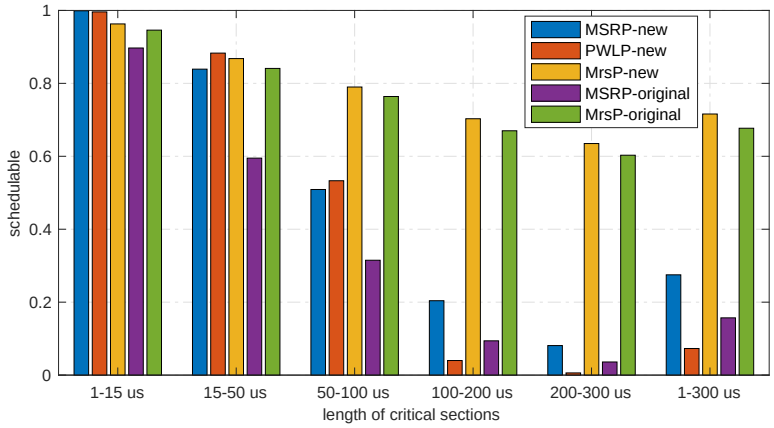


Figure 5: Schedulability in the presence of nested resources for $M = 8$, $n = 32$, $U = 0.1n$, $\kappa = 0.4$, $A = 2$ and $M$ Shared Resources.

(c) *Increasing critical section length L:* The results presented in Figure 5 show that both MSRP and PWLP are highly affected by the length of critical sections. It is important to note that, as resource accesses can be nested, the effect of the length increase is multiplied. In the case of MSRP, as the arrival blocking includes the busy-waiting and access time of lower priority tasks for each resource accessed in a nest of resources, clearly affecting the schedulability of higher priority tasks. Regarding PWLP, lower priority tasks are those specially affected by the critical section length increase, since the potential amount of busy-waiting time lost upon preemption before acquiring the lock is increased. This effect is again multiplied by the resource grouping used to analyse PWLP nested resources. Regarding MrsP, both approaches are clearly less affected. As higher priority tasks can preempt both busy-waiting and lock-holder lower priority tasks, the reduction in the schedulability is only a consequence of the proportionally longer spinning times, with better performance of the new, less pessimistic on resource contention analysis approach.

### *7.2. Run-time overhead*

Now we study the impact of the run-time overhead to the schedulability results with the analysis developed in Section 5. For the rest of the paper we will only consider "-new" approaches as denoted in Section 7.1, so we will now omit the suffix to avoid repetition. The experiment is conducted by varying the critical section length $L$. In addition, we present evidence of improved efficiency of MrsP by the controlled migration behaviours due to the NP-section. To conduct

this experiment, MSRP, PWLP and MrsP are implemented into the Preemptable Fixed Priority scheduler under the Litmus$^{RT}$ system (Calandrino et al., 2006a) for overhead measuring, with the support of nested resource accessing. Standard POSIX threads scheduled by a preemptive fixed-priority scheduler in Litmus$^{RT}$ are mapped to a set of tasks requesting certain shared resources. These threads share the same lock (implemented by a *struct* in Litmus$^{RT}$ kernel with a size of 114 bytes), and operates on a simple *strcut* with the size of 76 bytes inside the critical section. The implementation can be accessed via `https://github.com/RTSYork/Litmus_MSRP_PWLP_MrsP`. Table 6 summarises the worst-case bounding of the run-time cost variables introduced in the newly-developed schedulability tests measured from the implementations, and will be adopted in this experiment. The $C_{retry}$ notation denotes the worst-case run-time cost of the cancellation mechanism carried in PWLP.

Table 6: The Run-time Costs of the Candidate Protocols under Litmus$^{RT}$

| Variables | Worst-case Cost | Variables | Worst-case Cost |
|---|---|---|---|
| $CX_1$ | 5606 ns | $C^{unlock}_{MSRP}$ | 602 ns |
| $CX_2$ | 10,240 ns | $C^{lock}_{PWLP}$ | 1255 ns |
| $C_{retry}$ | 1663 ns | $C^{unlock}_{PWLP}$ | 602 ns |
| $C_{mig}$ | 8378 ns | $C^{lock}_{MrsP}$ | 1272 ns |
| $C^{lock}_{MSRP}$ | 979 ns | $C^{unlock}_{MrsP}$ | 1642 ns |

The schedulability analysis examined in this experiment includes (1) ILP-based MSRP test without run-time overhead (MSRP); (2) ILP-based MSRP test with run-time overhead (MSRP*); (3) ILP-based PWLP test without run-time overhead (PWLP); (4) ILP-based PWLP test with run-time overhead (PWLP*); (5) new MrsP analysis without run-time overhead (MrsP); (6) new MrsP analysis with run-time overhead, including the cost of migrations but without the protection of the NP-section (MrsP*); and (7) new MrsP analysis with NP section adopted, including run-time overhead and the NP-section adopted (MrsP-np*). The analysis "MrsP*" is modified from the analysis in Section 5 by taking the functions $Mnp^k$ and $n\hat{p}_i$ out of Equations (23) and (11) respectively. When "MrsP-np" is in use, the length of the NP-sections (i.e., $C_{np}$) is configured differently for each given system based on the approach given in Section 6.

From the experiment in Figure 6, we observed that, as with the observations obtained in (Zhao et al., 2017; Zhao, 2018), the schedulability tests with the run-time overhead accounted for (i.e., "MSRP*", "PWLP*", "MrsP*" and "MrsP-NP*") demonstrate lower schedulability results than the theoretical response time analysis do (i.e., "MSRP", "PWLP" and "MrsP") respectively, especially for MrsP, where schedulability results "MrsP*" and "MrsP-NP*" are such lower than that of "MrsP" in all cases. This observation reveals that run-time cost imposes non-trivial impact towards the schedulability of these protocols in the presence of nested resources, and again, illustrates the necessity of incorporating the run-time overhead into corresponding schedulability tests to provide more accurate and realistic schedulability results.

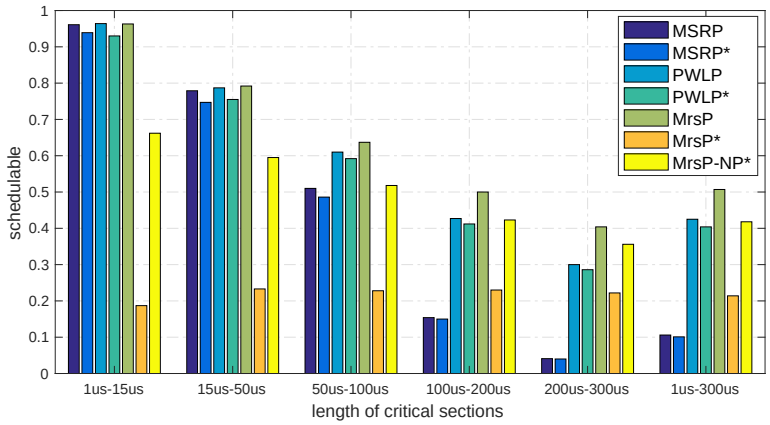


Figure 6: Schedulability for $M = 16$, $n = 48$, $U = 0.1n$, $\kappa = 0.4$, $A = 3$ and $M$ Shared Resources.

Now we focus on the schedulability tests with run-time overheads accounted for (i.e., bars labelled with "*"). Firstly, without the protection of the NP-sections, the cost of migrations in MrsP (i.e., "MrsP*") impose a huge impact to the schedulability of this protocol, and leads to the protocol being impractical. However, with the NP-section adopted (i.e., "MrsP-NP*"), the efficiency of the helping mechanism is significantly improved. This observation illustrates that the proposed NP-section effectively reduce the cost of migrations in MrsP. Compared to "PWLP*", "MrsP-np*" is less favourable when applied to short critical sections as one single migration has a cost of $8.378\mu s$ in this experiment, where "MrsP-np*" provides a low schedulability with $L = [1\mu s, 50\mu s]$. However, when $L = [100\mu s, 200\mu s], [200\mu s, 300\mu s]$ or $[1\mu s, 300\mu s]$, MrsP with the NP-section adopted shows a better schedulability than both "MSRP*" and "PWLP*", which again leads to the conclusion that MrsP works better with longer critical sections while MSRP and PWLP are favourable if critical sections are short. By taking the run-time overhead into account while analysing all candidate locking protocols, we have improved the accuracy of schedulability of MrsP in the presence of nested resources and proved the necessity for incorporating the run-time costs into the schedulability test of MrsP, especially the non-trivial costs for task migrations.

In summary, results obtained from the above evaluation clearly show that there is no silver-bullet with regards to resource sharing among processors. Yet, we have revealed a clear relationship between the performance of the studied protocols and various critical section length ranges, where MSRP and PLWP are more favourable with short critical section and MrsP is desirable with either long resources or mixed length of critical sections. While longer critical sections can have an impact on migrations costs, these are bounded by the NP-section configuration presented in Section 6; in this respect, the enhanced MrsP

protocol based on this aspect clearly outperforms MSRP and PWLP as it exhibits both lower arrival blocking and no re-queuing costs. With a mixed length of critical sections (i.e., $L = [1\mu s, 300\mu s]$ in Figure 6), both MrsP and PWLP demonstrate strong schedulability, where MrsP is slightly better in general. The lower schedulability of MSRP showcases that arrival blocking is the most relevant factor in scheduling performance when varying critical section lengths. However, compared to either MSRP and PWLP, the cost of migrations in MrsP becomes significant with short critical sections and hence, greatly undermines MrsP schedulability. On the other hand, MSRP and PWLP impose much less run-time cost towards the system, and hence are more favourable under such cases.

### *7.3. Time Consumption for Analysing Spin-based Locking Systems*

The last experiment conducted in this article is to investigate and to compare the time consumption of our newly-developed schedulability tests and the ILP-based analysis i.e., the time that a schedulability test spends to produce schedulability results. As the computing time of a given test largely depends on the exact system being generated, there can be huge differences between the computation times under the same system setting. To illustrate the overall time consumption of the schedulability tests in general, 10000 systems are generated for each given system setting and an average computing time of each analysis is reported in Tables 7 and 8. Note that this section aims at comparing the run-time cost of the proposed analysis and other schedulability tests, the computation time presented is for schedulability tests only. That is, the cost of Algorithm 1 is not included in this measurement. New schedulability tests of MSRP, PWLP used in this experiment is developed by Zhao (2018) based on the new MrsP schedulability test by Zhao et al. (2017), which remove the need of the ILP solver but achieve identical scheduability results with ILP-based tests (Zhao, 2018).

Table 7: The Average Time Consumption (in $ms$) and Standard Deviation (std.) for Analysing Systems with $M = 16$, $U = 0.1n$, $\kappa = 0.4$, $A = 2$, $L = [15\mu s, 50\mu s]$, and $M$ Shared Resources.

| $n$ | MSRP | | PWLP | | MrsP | | MSRP-ILP | | PWLP-ILP | |
|---|---|---|---|---|---|---|---|---|---|---|
| | avg. | std. | avg. | std. | avg. | std. | avg. | std. | avg. | std. |
| 48 | 1.24 | $8.4{\cdot}10^5$ | 0.96 | $6.9{\cdot}10^5$ | 3.33 | $4.4{\cdot}10^6$ | 139.65 | $8.2{\cdot}10^7$ | 137.93 | $8.3{\cdot}10^7$ |
| 64 | 1.93 | $1.3{\cdot}10^6$ | 1.59 | $1.2{\cdot}10^6$ | 5.69 | $8.0{\cdot}10^6$ | 228.17 | $1.4{\cdot}10^8$ | 232.97 | $1.6{\cdot}10^8$ |
| 80 | 1.37 | $3.1{\cdot}10^6$ | 0.87 | $2.0{\cdot}10^6$ | 2.21 | $4.8{\cdot}10^6$ | 252.84 | $3.0{\cdot}10^8$ | 328.91 | $3.5{\cdot}10^8$ |
| 96 | 1.85 | $4.6{\cdot}10^6$ | 1.00 | $2.6{\cdot}10^6$ | 3.16 | $1.1{\cdot}10^7$ | 318.64 | $4.6{\cdot}10^8$ | 441.86 | $4.6{\cdot}10^8$ |
| 112 | 1.83 | $4.6{\cdot}10^6$ | 1.01 | $2.5{\cdot}10^6$ | 3.35 | $7.3{\cdot}10^6$ | 347.40 | $5.3{\cdot}10^8$ | 618.20 | $5.3{\cdot}10^8$ |

Table 8: The Average Time Consumption (in $ms$) and Standard Deviation (SD) for Analysing Systems with $n = 5M$, $U = 0.1n$, $\kappa = 0.4$, $A = 2$, $L = [1\mu s, 15\mu s]$, and $M$ Shared Resources.

| $M$ | MSRP | | PWLP | | MrsP | | MSRP-ILP | | PWLP-ILP | |
|---|---|---|---|---|---|---|---|---|---|---|
| | avg. | std. | avg. | std. | avg. | std. | avg. | std. | avg. | std. |
| 4 | 0.19 | $5.2{\cdot}10^4$ | 0.17 | $5.0{\cdot}10^4$ | 1.98 | $1.6{\cdot}10^6$ | 37.11 | $3.1{\cdot}10^7$ | 28.67 | $3.2{\cdot}10^7$ |
| 8 | 1.72 | $4.9{\cdot}10^5$ | 1.22 | $3.5{\cdot}10^5$ | 13.33 | $1.3{\cdot}10^7$ | 361.85 | $3.2{\cdot}10^8$ | 359.55 | $3.7{\cdot}10^8$ |
| 12 | 4.86 | $2.8{\cdot}10^6$ | 3.42 | $1.8{\cdot}10^6$ | 10.96 | $2.4{\cdot}10^7$ | 972.11 | $7.8{\cdot}10^8$ | 1167.84 | $9.4{\cdot}10^8$ |
| 16 | 6.83 | $7.9{\cdot}10^6$ | 4.72 | $5.1{\cdot}10^6$ | 5.61 | $2.1{\cdot}10^7$ | 1299.37 | $1.5{\cdot}10^9$ | 1700.07 | $1.7{\cdot}10^9$ |

As given in both tables, ILP-based tests (i.e., "MSRP-ILP" and "PWLP-ILP") require much more computation costs (in terms of both average computing time and standard deviation) than those that do not rely on such a technique (i.e., "MSRP" and "PWLP") under all tested system settings. Among the first three analysis, where the requirement of ILP solver is removed, MrsP takes more time to compute the response times due to the additional migration cost analysis (up to 13.3 milliseconds) while MSRP and PWLP consume similar computation time to deliver the results. In contrast, the ILP-based ones require up to 1700 milliseconds with $M = 16$ in Table 8. One interesting observation is that with a shorter $L$, all tests require larger computation time to deliver the schedulability results, see $n = 80$ in Table 7 and $M = 16$ in Table 8 (with $L = [15\mu s, 50\mu s]$ and $[1\mu s, 15\mu s]$ respectively). This is due to the fact that with a shorter critical section length, the response time of tasks will have a smaller increment under each recursion calculation so that more recursions could be required to either get fixed response times or reach the deadlines of tasks (i.e., where the tests are terminated).

Table 9: The Increase Rate of the Time Consumption in Table 7.

| $n$ | MSRP | PWLP | MrsP | MSRP-ilp | PWLP-ilp |
|---|---|---|---|---|---|
| $48 \rightarrow 64$ | 155 % | 166 % | 171 % | 164 % | 170 % |
| $64 \rightarrow 80$ | 71 % | 55 % | 39 % | 111 % | 141 % |
| $80 \rightarrow 96$ | 135 % | 114 % | 143 % | 126 % | 134 % |
| $96 \rightarrow 112$ | 107 % | 102 % | 106 % | 109 % | 140 % |

Table 10: The Increase Rate of the Time Consumption in Table 8.

| $M$ | MSRP | PWLP | MrsP | MSRP-ilp | PWLP-ilp |
|---|---|---|---|---|---|
| $4 \rightarrow 8$ | 905 % | 718 % | 672 % | 975 % | 1253 % |
| $8 \rightarrow 12$ | 283 % | 280 % | 83 % | 267 % | 325 % |
| $12 \rightarrow 16$ | 141 % | 138 % | 121 % | 134 % | 146 % |

Tables 9 and 10 present the increase rate of the computation cost of each analysis given in Tables 7 and 8. In general, the increasing ratio of computation cost of the ILP-based analysis demonstrates a slightly higher increase rate compared to ones without ILP technique. Note, we observe that the computation

cost of the new analysis does not increase monotonically with the increase of the system parameter settings (i.e., $n$ from 64 $\rightarrow$ 80 in Table 9). This is because with $n > 64$, the evaluated schedulability tests can hardly schedule any of the input systems (recall Figure 3). For schedulable systems, these tests need to iterate through all tasks in the system and compute a fixed response time for each task via recursive calculations, and hence, returns the result that the system is schedulable. However, for systems that cannot be schedulable by a given test, it is not necessary to compute response time for all tasks, and the test returns immediately as soon as an unschedulable task (i.e., its response time higher than deadline) is being found. Therefore, a computation time decrease for all new tests is observed. In addition, the input systems for analysis are generated randomly, including the usage of shared resources. Thus, systems with low resource contention could be generated under high system setting parameters, which requires less time for analysing such systems, and hence, causes the computation time decrease under the new tests in Table 9 with $n$ from 64 $\rightarrow$ 80. However, even under such case, the costs of the ILP-based analysis keep increasing due to the use of ILP solver, which needs to establish the optimisation problem based on the constraints (e.g., 8 constraints for analysing MSRP systems) for each task and each resource access before computing response time, and hence, requires a large amount of calculations.

Based on these tables, the schedulability test proposed in this paper requires less computation cost (and has lower increase rates by giving higher systems setting parameters) than the ILP-based ones. Admittedly, the time consumed by the ILP-based analysis for executing once is definitely acceptable, as such tests are usually performed off-line (i.e., before the execution of the system). For systems with a set of locking protocols pre-defined, the ILP-based analysis provides a valuable unified analysing tool that can be adopted to analyse 8 spin-based protocols. However, for more complex application scenarios, (e.g., as the fitness function of a genetic algorithm adopted in (Zhao, 2018)), where a vast amount of invocation towards the analysis is performed, practising this analysis can lead to significant consumption costs; thus, it would be more favourable to use a different technique than an ILP-based one in a practical viewpoint.

## 8. Conclusions

Multiprocessor platforms are becoming the system configuration of choice in the area of real-time embedded systems (Davis and Burns, 2011). The development of hard real-time systems over these platforms presents a number of challenges, being one of them the scheduling of tasks under the presence of globally shared resources (Brandenburg, 2011). Although initially few results from uniprocessor scheduling were expected to be translated to multiprocessor systems (Liu, 1969), notable multiprocessor resource sharing approaches have built on SRP (Baker, 1990) and PCP (Sha et al., 1990) uniprocessor lock-based approaches. Some of these relevant multiprocessor locking protocols are MSRP (Block et al., 2007), PWLP (Anderson et al., 1998; Craig, 1993) and MrsP (Burns and Wellings, 2013).

In this article, MrsP is selected due to its promising features (i.e., priority ceiling, migration-based helping and full support for nested resource access) and its wide adoption in real-world operating systems and applications (e.g., in Litmus$^{RT}$ (Calandrino et al., 2006b) and the UPMSat-2 satellite (Garrido et al., 2015)). MrsP was first proposed in 2013 and has been improved over time by a number of different contributions providing missing features or improving existing ones. These include a full approach to fine-grained nested resources support (Garrido et al., 2017b), a modified helping mechanism (Zhao and Wellings, 2017), migration costs analysis and an improved schedulability analysis (Zhao et al., 2017) still based on RTA. In this paper, extensions towards MrsP schedulability test for supporting nested resource accesses are proposed for conducting analysis over both theoretical response time and run-time cost. With the proposed analysing techniques, we provide a complete run-time overhead-aware schedulability analysis for MrsP. Then, a set of NP-section configuration approaches are also addressed that provide recommendations for NP section length setting. Finally, we evaluate the MrsP protocol by comparing its schedulability ratio under different configurations with other relevant spin-lock protocols, such as MSRP and PWLP.

The results of this evaluation show that there is no silver-bullet with regards to resource sharing among processors. Although MrsP strictly dominates MSRP if no migrations cost is considered, it has been demonstrated that, in practice (when these costs are considered) the situation is not a so clear cut. However, thanks to the contributions presented in this paper, MrsP still produces overall better results on realistic platforms and test cases where there exist both short and long critical sections. Our new, less pessimistic analysis exhibits the main MrsP strong points: while it maintains the spin-lock efficiency on short resource access, it also reduces the effect on arrival blocking of long resource access. With regards to the comparison against PWLP, this latter protocol has only shown better performance with very short critical sections or low resource access frequency, while MrsP has proven to be a better all-round approach capable of proving consistently better and more scalable results when increasing resource length and access frequency. Again, the combination of our new analysis and MrsP inherited PCP properties provide an effective compromise between local and global blocking.

There are a number of topics that will be addressed as future work. Firstly, as with in Zhao et al. (2017), the analysis developed in this article depends on several assumptions that impose resections towards the system and task model (i.e., homogeneous multiprocessor systems with sporadic tasks). In future, such assumptions could be removed to improve the applicability of the proposed analysis. Then, as demonstrated in Section 7, although the migration-based helping mechanism is worthwhile with long critical sections, the cost of migrations becomes significant and greatly undermines MrsP schedulability if critical sections are short. In future, a flexible helping mechanism could be desirable, which migrates a resource-holding task only if it is worthwhile to do so (e.g., when the cost of migrations is less than the remaining computation of the critical section). By doing so, an overall schedulability improvement of MrsP could

be achieved and better scalability of MrsP towards varied critical section length could be obtained. With the adoption of the configurable NP-section for controlled migrations, approaches that optimise the NP-section configuration to achieve improved robustness of MrsP systems (i.e., remains schedulable with extra unexpected inference taken by the system) would be desirable. In addition, as described in Section 7, the schedulability tests are evaluated via their pessimism and accuracy, which have conflicted metrics based on the same measurement (i.e., the percentage of schedulable systems). In the future, approaches that identify the optimal schedulability point of resource sharing protocols would be desirable, which can greatly facilitate the evaluation of corresponding schedulability tests and could provide further motivations for improving the existing analysis to be close or equal to the ideal point.

Further, with the proposed schedulability tests, the response time of a given task depends on potentially all other tasks in the system. As proved by Zhao (2018), due to this feature, DMPO is no longer an optimal priority-ordering algorithm under the new MrsP analysis while the majority of the existing search-based priority assignments (e.g., Optimal Priority Assignment by Bletsas and Audsley (2006) and Robust Priority Assignment by Davis and Burns (2007)) are not applicable. As for task allocation, the traditional utilisation-based task allocation algorithms (e.g., the Worst-Fit and Best-Fit heuristics) do not consider shared resource access costs, and hence, can lead to long blocking times. In addition, existing resource-oriented task allocation schemes (e.g., the SPA algorithm in (Lakshmanan et al., 2009) and the BPA algorithm in (Nemati et al., 2010)) are incompatible with the new schedulability test applied due to the aforementioned response dependency (i.e., the response time of a task depends potentially on the response time of all other tasks in the system). Therefore, novel priority ordering and task allocation algorithms (e.g., search-based heuristics and evolutionary algorithms (Lee and Lee, 2005; Zhao et al., 2019b,a)) that can further improve the schedulability of MrsP systems are desirable. In this respect, the analysis presented in this paper exhibits a speedup factor ranging from 20 to 200 times when compared to the ILP-based analysis, thus constituting a notable state of the art advancement, enabling the efficient use of the aforementioned priority ordering and task allocation algorithms.

Finally, although fully functional MrsP implementations are available in several real-world operating systems, the requirement for migrations could impose difficulty to the protocol implementation in practice (Catellani et al., 2015; Shi et al., 2017) and raise race conditions (Zhao and Wellings, 2017). Therefore, an implementation guide for MrsP (in particular, a guide for realising the helping mechanism) could greatly prompt the use of this protocol in real-world systems and applications (Chang et al., 2019).

## Acknowledgements

This work has been partially funded by the Spanish National R&D&I plan (project M2C2, TIN2014-56158-C4-3-P and project PRECON-I4, TIN2017-

86520-C3-2-R). UPM authors would also like to acknowledge the contributions of Juan Zamorano and Alejandro Alonso.

## References

Anderson, J. H., Jain, R., Jeffay, K., 1998. Efficient object sharing in quantum-based real-time systems. In: Real-Time Systems Symposium, 1998. Proceedings. The 19th IEEE. IEEE, pp. 346–355.

Anderson, J. H., Ramamurthy, S., Jeffay, K., 1997. Real-time computing with lock-free shared objects. ACM Transactions on Computer Systems (TOCS) 15 (2), 134–165.

Audsley, N., Burns, A., Richardson, M., Tindell, K., Wellings, A. J., 1993. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal 8 (5), 284–292.

Audsley, N. C., 2001. On priority assignment in fixed priority scheduling. Information Processing Letters 79 (1), 39–44.

Baker, T. P., 1990. A stack-based resource allocation policy for realtime processes. In: [1990] Proceedings 11th Real-Time Systems Symposium. IEEE, pp. 191–200.

Bini, E., Buttazzo, G. C., 2005. Measuring the performance of schedulability tests. Real-Time Systems 30 (1-2), 129–154.

Biondi, A., Brandenburg, B. B., Wieder, A., 2016. A blocking bound for nested FIFO spin locks, 291–302.

Bletsas, K., Audsley, N., 2006. Optimal priority assignment in the presence of blocking. Information processing letters 99 (3), 83–86.

Block, A., Leontyev, H., Brandenburg, B. B., Anderson, J. H., 2007. A flexible real-time locking protocol for multiprocessors. In: Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on. IEEE, pp. 47–56.

Brandenburg, B. B., 2011. Scheduling and locking in multiprocessor real-time operating systems. Ph.D. thesis, The University of North Carolina at Chapel Hill, https://cs.unc.edu/~anderson/diss/bbbdiss.pdf.

Brandenburg, B. B., 2013a. Improved analysis and evaluation of real-time semaphore protocols for p-fp scheduling. In: Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th. IEEE, pp. 141–152.

Brandenburg, B. B., 2013b. "SchedCAT: Schedulability test collection and toolkit". http://www.mpi-sws.org/bbb/projects/schedcat, accessed: 2016-11-1.

Brandenburg, B. B., Anderson, J. H., 2010. Optimality results for multiprocessor real-time locking. In: Real-Time Systems Symposium (RTSS), 2010 IEEE 31st. IEEE, pp. 49–60.

Brandenburg, B. B., Calandrino, J. M., Block, A., Leontyev, H., Anderson, J. H., 2008. Real-time synchronization on multiprocessors: To block or not to block, to suspend or spin? In: 2008 IEEE Real-Time and Embedded Technology and Applications Symposium. IEEE, pp. 342–353.

Burns, A., Wellings, A., 2016. Analysable Real-Time Systems: Programmed in Ada. CreateSpace Independent Publishing Platform.

Burns, A., Wellings, A. J., 2013. A schedulability compatible multiprocessor resource sharing protocol – MrsP. In: Real-Time Systems (ECRTS), 25th Euromicro Conference on. IEEE, pp. 282–291.

Buttle, D., 2012. Real-time in the prime-time, etas gmbh, germany. In: Keynote talk at 24th Euromicro Conference on Real-Time Systems, Pisa, Italy.

Calandrino, J. M., Leontyev, H., Block, A., Devi, U. C., Anderson, J. H., 2006a. Litmus^ rt: A testbed for empirically comparing real-time multiprocessor schedulers. In: Real-Time Systems Symposium. RTSS. 27th IEEE International. IEEE, pp. 111–126.

Calandrino, J. M., Leontyev, H., Block, A., Devi, U. C., Anderson, J. H., 2006b. Litmus$^{RT}$: A testbed for empirically comparing real-time multiprocessor schedulers. In: Real-Time Systems Symposium, 2006. RTSS'06. 27th IEEE International. IEEE, pp. 111–126.

Catellani, S., Bonato, L., Huber, S., Mezzetti, E., 2015. Challenges in the implementation of MrsP. In: Ada-Europe International Conference on Reliable Software Technologies. Springer, pp. 179–195.

Cavalcanti, A., Miyazawa, A., Wellings, A., Woodcock, J., Zhao, S., 2016. Java in the safety-critical domain. In: School on Engineering Trustworthy Software Systems. Springer, pp. 110–150.

Chang, W., Zhao, S., Wei, R., Wellings, A. J., Burns, A., 2019. From java to real-time java: A model-driven methodology with automated toolchain (invited paper): From java to real-time java: A model-driven methodology. In: 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems. York.

Craig, T. S., 1993. Queuing spin lock algorithms to support timing predictability. In: Real-Time Systems Symposium, 1993., Proceedings. IEEE, pp. 148–157.

Davis, R. I., Burns, A., 2007. Robust priority assignment for fixed priority real-time systems. In: Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International. IEEE, pp. 3–14.

Davis, R. I., Burns, A., 2011. A survey of hard real-time scheduling for multiprocessor systems. Acm Computing Surveys 43 (4), 1–44.

Emberson, P., Stafford, R., Davis, R. I., 2010. 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.

Faggioli, D., Lipari, G., Cucinotta, T., 2010. The multiprocessor bandwidth inheritance protocol. In: Real-Time Systems (ECRTS), 22nd Euromicro Conference on. IEEE, pp. 90–99.

Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K., Lange, K., 2009. Autosar–a worldwide standard is on the road. In: 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden. Vol. 62. p. 5.

Gai, P., Lipari, G., Natale, M. D., 2001. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In: Proceedings of the 22nd IEEE Real-Time Systems Symposium. IEEE Computer Society.

Garrido, J., Zamorano, J., Alonso, A., de la Puente, J. A., 2017a. Evaluating MSRP and MrsP with the multiprocessor ravenscar profile. In: Ada-Europe International Conference on Reliable Software Technologies. Springer, pp. 3–17.

Garrido, J., Zamorano, J., de la Puente, J. A., Alonso, A., Salazar, E., 2015. Ada, the programming language of choice for the upmsat-2 satellite. Data Systems in AerospaceDASIA 2015.

Garrido, J., Zhao, S., Burns, A., Wellings, A., 2017b. Supporting nested resources in MrsP. In: Ada-Europe International Conference on Reliable Software Technologies. Springer, pp. 73–86.

Johnson, D. S., 1973. Near-optimal bin packing algorithms. Ph.D. thesis, Massachusetts Institute of Technology.

Lakshmanan, K., de Niz, D., Rajkumar, R., 2009. Coordinated task scheduling, allocation and synchronization on multiprocessors. In: Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE. IEEE, pp. 469–478.

Lee, Z.-J., Lee, C.-Y., 2005. A hybrid search algorithm with heuristics for resource allocation problem. Information sciences 173 (1-3), 155–167.

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

Liu, C. L., 1969. Scheduling algorithms for multiprocessors in a hard real-time environment. JPL Space Programs Summary, 1969.

Liu, C. L., Layland, J. W., 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) 20 (1), 46–61.

Nemati, F., Nolte, T., Behnam, M., 2010. Partitioning real-time systems on multiprocessors with shared resources. Principles of Distributed Systems, 253–269.

Rajkumar, R., 1991. Synchronization in real-time systems: a priority inheritance approach. Vol. 151. Springer Science & Business Media.

Rajkumar, R., Sha, L., Lehoczky, J. P., 1988. Real-time synchronization protocols for multiprocessors. In: IEEE Real-Time Systems Symposium.

Sha, L., Rajkumar, R., Lehoczky, J. P., 1990. Priority inheritance protocols: An approach to real-time synchronization. IEEE Trans. Comput. 39 (9).

Shi, J., Chen, K.-H., Zhao, S., Huang, W.-H., Chen, J.-J., Wellings, A., 2017. Implementation and evaluation of multiprocessor resource synchronization protocol (MrsP) on LITMUSRT.

Sundell, H., Tsigas, P., 2000. Space efficient wait-free buffer sharing in multiprocessor real-time systems based on timing information. In: Real-Time Computing Systems and Applications, 2000. Proceedings. Seventh International Conference on. IEEE, pp. 433–440.

Takada, H., Sakamura, K., 1995. Real-time scalability of nested spin locks. In: Real-Time Computing Systems and Applications. Proceedings., Second International Workshop on. IEEE, pp. 160–167.

Takada, H., Sakamura, K., 1997. A novel approach to multiprogrammed multiprocessor synchronization for real-time kernels. In: Real-Time Systems Symposium. Proceedings., The 18th IEEE. IEEE, pp. 134–143.

Ward, B., Anderson, J., 2012a. Nested multiprocessor real-time locking with improved blocking. In: Proceedings of the 24th Euromicro conference on real-time systems.

Ward, B. C., Anderson, J. H., 2012b. Supporting nested locking in multiprocessor real-time systems. In: 24th Euromicro Conference on Real-Time Systems. IEEE, pp. 223–232.

Wieder, A., Brandenburg, B. B., 2013. On spin locks in autosar: Blocking analysis of fifo, unordered, and priority-ordered spin locks. In: Real-Time Systems Symposium (RTSS), 2013 IEEE 34th. IEEE, pp. 45–56.

Zhao, S., 2018. A FIFO Spin-based Resource Control Framework for Symmetric Multiprocessing. Ph.D. thesis, The University of York, http://etheses.whiterose.ac.uk/21014/.

Zhao, S., Dziurzanski, P., Przewozniczek, M., Komarnicki, M., Indrusiak, L. S., 2019a. Cloud-based dynamic distributed optimisation of integrated process planning and scheduling in smart factories. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, pp. 1381–1389.

Zhao, S., Garrido, J., Burns, A., Wellings, A., 2017. New schedulability analysis for MrsP. In: Embedded and Real-Time Computing Systems and Applications (RTCSA), 2017 IEEE 23rd International Conference on. IEEE, pp. 1–10.

Zhao, S., Mei, H., Dziurzanski, P., Przewozniczek, M. W., Soares Indrusiak, L., 2019b. Cloud-based integrated process planning and scheduling optimisation via asynchronous islands.

Zhao, S., Wellings, A., 2017. Investigating the correctness and efficiency of MrsP in fully partitioned systems. In: The 10th York Doctoral Symposium on Computer Science and Electronics (YDS 2017). The University of York.