



Deposited via The University of York.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/161461/>

Version: Accepted Version

---

**Article:**

Zhao, Shuai, Chang, Wanli, Wei, Ran et al. (Accepted: 2020) Priority Assignment on Partitioned Multiprocessor Systems with Shared Resources. IEEE Transactions on Computers. ISSN: 0018-9340 (In Press)

---

**Reuse**

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

**Takedown**

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing [eprints@whiterose.ac.uk](mailto:eprints@whiterose.ac.uk) including the URL of the record and the reason for the withdrawal request.

# Priority Assignment on Partitioned Multiprocessor Systems with Shared Resources

Shuai Zhao, Wanli Chang, Ran Wei, Weichen Liu, Nan Guan, Alan Burns and Andy Wellings

**Abstract**—Driven by industry demand, there is an increasing need to develop real-time multiprocessor systems which contain shared resources. The Multiprocessor Stack Resource Policy (MSRP) and Multiprocessor resource sharing Protocol (MrsP) are two major protocols that manage access to shared resources. Both of them can be applied to Fixed-Priority Preemptive Scheduling (FPPS), which is enforced by most commercial real-time systems regulations, and which requires task priorities to be assigned before deployment. Along with MSRP and MrsP, there exist two forms of schedulability tests that bound the worst-case blocking time due to resource accesses: the traditional ones being more widely adopted and the more recently developed holistic ones which deliver tighter analysis. On uniprocessor systems, there are several well-established optimal priority assignment algorithms. Unfortunately, on multiprocessor systems with shared resources, the issue of priority assignment has not been adequately understood. In this work, we investigate three mainstream priority assignment algorithms – Deadline Monotonic Priority Ordering (DMPO), Audsley’s Optimal Priority Assignment (OPA), and Robust Priority Assignment (RPA), in the context of partitioned multiprocessor systems with shared resources. Our contributions are multifold: First, we prove that DMPO is optimal with the traditional schedulability tests. Second, two counter examples are given as evidence that DMPO is not optimal with the tighter holistic schedulability tests. Third, we then analyse the pessimism arising from the adoption of OPA and RPA with the holistic tests. Lastly, we propose a Slack-based Priority Ordering (SPO) algorithm that minimises such pessimism, and has polynomial time complexity. Comprehensive experiments show that SPO outperforms (i.e., results in a larger number of schedulable systems) DMPO, OPA and RPA in general with the holistic schedulability tests, by up to 15%. With the theoretical contributions, this paper is a useful guide to priority assignment in real-time partitioned multiprocessor systems with shared resources.

## I. INTRODUCTION

Emerging embedded applications in various domains such as automotive, robotics, medical, communication and industrial automation [10]–[12], [23], requires the wide adoption of real-time multiprocessor systems [16]. Whilst more computational power can be obtained, the transition from uniprocessor to multiprocessor systems raises new challenges and breaks many well-practised real-time algorithms. One critical issue is

the management of logical and physical resources shared by tasks executing in parallel, such as data structures, I/O and network ports [7].

On uniprocessor systems, mature resource sharing protocols are well understood with several optimal solutions available, such as Priority Ceiling Protocol (PCP) [28], Stack Resource Policy (SRP) [3] and Deadline Floor Protocol (DFP) [8]. On multiprocessor systems, optimal resource sharing protocols are not available [33]. However, many solutions, e.g., Multiprocessor PCP (MPCP) [27], Multiprocessor SRP (MSRP) [19]), and the Multiprocessor resource sharing Protocol (MrsP) [9] have been proposed for managing globally shared resources, i.e., those resources shared between processors. In the mean time, new holistic schedulability tests for analysing these protocols are emerging, such as the framework proposed in [32] for several spin-based protocols (including MSRP) and the analysis proposed in [34], [35] for both MSRP and MrsP. They provide tighter schedulability results than the traditional schedulability tests reported together with the protocols.

The majority of commercial real-time systems regulations enforce Fixed-Priority Preemptive Schedulers (FPPS), where priorities must be first assigned to tasks [15]. As an optimal priority assignment [16], Deadline Monotonic Priority Ordering (DMPO) has been widely used for decades on uniprocessor systems. In addition, search-based priority assignments such as Audsley’s Optimal Priority Assignment scheme (OPA) [2] and Robust Priority Assignment (RPA) [14] are optimal for a wider range of systems. Furthermore, these priority assignments have been proved to remain optimal on uniprocessor systems, in the presence of shared resources managed by those optimal uniprocessor resource sharing protocols, e.g., SRP and PCP [5]. However, on multiprocessor systems with shared resources, whether DMPO and these search-based priority assignments maintain their optimality (or even applicability) is unknown [5], [16].

**Main contributions:** In this paper, we investigate optimality and applicability of three mainstream priority assignment algorithms — DMPO, OPA and RPA — in the context of partitioned (i.e., fixed allocation of tasks to processors and no migration) multiprocessor systems with shared resources, based on two multiprocessor resource sharing protocols — MSRP and MrsP — and their schedulability tests. We prove that DMPO remains optimal under the traditional schedulability tests of both MSRP and MrsP. However, its optimality is undermined in the tighter holistic schedulability tests. In addition, we explain the pessimism resulting from applying OPA and RPA to the holistic tests with experimental evidence, which illustrates that these algorithms are no longer optimal.

S. Zhao, W. Chang, A. Burns and A. Wellings are with the Department of Computer Science, University of York, UK. E-mail: shuai.zhao@york.ac.uk, wanli.chang@york.ac.uk, alan.burns@york.ac.uk, andy.wellings@york.ac.uk

R. Wei is with the School of Artificial Intelligence, Dalian University of Technology, China. E-mail: ranwei@dlut.edu.cn

W. Liu is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: liu@ntu.edu.sg

N. Guan is with the Department of Computing, Hong Kong Polytechnic University, China. E-mail: nan.guan@polyu.edu.hk

W. Chang is the corresponding author.

A Slack-based Priority Ordering (SPO) with polynomial time complexity is then proposed to minimise the pessimism incurred when applying the holistic schedulability tests. Finally, experimental results show the impact of priority assignment on the schedulability of multiprocessor systems with shared resources, and the effectiveness of SPO.

**Organisation of this paper:** Section II describes the priority assignments studied in this paper. Section III gives the system model and explains the targeted resource sharing protocols, i.e., MSRP and MrsP. Section IV presents the traditional and holistic schedulability tests for MSRP and MrsP, respectively. A comprehensive investigation towards the optimality and applicability of DMPO, OPA and RPA under the considered schedulability tests is conducted in Sections V and VI. In Section VII, SPO is proposed and discussed in detail. Section VIII reports the experimental results and Section IX draws conclusions from the study and discusses future work.

## II. RELATED WORK

In this section, we provide a brief review of the major priority assignment algorithms related to this work, based on a survey reported in [16]. Descriptions of the targeted multiprocessor resource sharing protocols and their schedulability tests are given in later sections. Note that in this work we focus on the priority assignment problem rather than the resource sharing and analysis issues of multiprocessor systems.

DMPO proposed in [25] assigns higher priorities to tasks with shorter deadlines. As proved in [16], DMPO is optimal for sporadic tasks with constrained deadlines. Notably, DMPO remains optimal in the presence of shared resources managed by SRP, PCP and DFP on uniprocessors [5] whilst the Deadline Minus Release Jitter Monotonic (D-J) Priority Ordering is optimal in the presence of release jitters [37]. However, the optimality of DMPO can be easily undermined with minor changes to the system, such as in the presence of arbitrary deadlines [24], [25].

In [2], a search-based priority assignment algorithm, OPA, is proposed. Given a set of tasks with priorities unassigned, this algorithm starts from the lowest priority level and checks whether there exists a priority-unassigned task that is schedulable at that priority level; assuming all other unassigned tasks have a higher priority. If such a task is found, it is assigned this priority. The algorithm then moves on to the next priority level and checks the rest of the unassigned tasks. The algorithm returns a schedulable solution if each task is assigned a priority. OPA is an optimal priority assignment as it guarantees that a schedulable priority ordering can be found if there are any, with a worst-case of  $n(n+1)/2$  iterations. In addition, OPA is proved to be optimal for a wider range of application semantics than DMPO, such as systems with offset release times [1], arbitrary deadlines [30], non-preemptive execution [22] and mixed criticalities [31]. Later, several extensions based on OPA have been developed to further optimise priority ordering based on different metrics, such as minimising the number of priority levels [2], or minimising the lexicographical distance [13].

However, with OPA, tasks are assigned an arbitrary priority if there exists more than one schedulable task at a given priority level. As described in [14], this can result in a system that is merely schedulable, but fragile to minor changes of task parameters, execution budgets overrun or under-estimated interference. To address this concern, RPA was developed in [14] as an extension of OPA, with an approach to specify the exact task that should be assigned at each priority level. In RPA, an interference function is introduced to model the amount of potential interference that tasks can incur on each priority level whilst remaining schedulable. With this function, RPA aims to produce a priority ordering that can tolerate the maximum amount of additional interference.

Similar to OPA, RPA starts with the lowest priority level and requires  $n(n+1)/2$  binary searches to find the maximum additional interference for all priority levels. On a given priority level, the task that can tolerate the maximum amount of additional interference will be assigned that priority. The algorithm then iterates to the next priority level until all tasks are assigned a priority. If a feasible priority ordering can be found, this system is guaranteed to be schedulable and is able to cope with the minimal tolerable additional interference among all priority levels.

## III. SYSTEM MODEL AND RESOURCE SHARING PROTOCOLS

As partially discussed in Section I, most commercial real-time systems regulations enforce FPPS and explicitly mandate the use of spin locks for managing shared resources (e.g., the AUTOSAR framework for automotive systems [17], [18]). In this work, we focus on the general sporadic task model in fully-partitioned multiprocessor systems with FPPS and spin-based resource sharing protocols adopted.

The multiprocessor system under study contains  $M$  processors ( $P_1$  to  $P_M$ ). We consider a set of tasks  $\Gamma$  and the tasks allocated to each processor are fixed before execution (i.e., fully-partitioned). Tasks can present periodic or sporadic activation patterns following the general *sporadic task model*. For a given task in the system (denoted by  $\tau_x$ ), it has a period  $T_x$ , a relative deadline  $D_x$ , a priority  $pri(\tau_x)$ , a pure worst-case computation time  $C_x$ , and a response time  $R_x$ . The pure computation time of  $\tau_x$  indicates the time it takes to execute without waiting for, or accessing, shared resources. Deadlines are constrained, that is:  $D_x \leq T_x$ . The index value of a task also indicates its priority (i.e.,  $pri(\tau_x) = x$ ), and each task is assigned a unique priority. A higher priority value indicates a higher execution eligibility.

Within the system, there also exists a set of resources  $\mathbb{R}$  that are shared among tasks. These resources are accessed in the mutual exclusion manner, by their associated *critical sections*, for data consistency. For each shared resource  $r^k$ ,  $c^k$  denotes the worst-case cost for executing its associated critical section. During each release, a task  $\tau_x$  may request a resource  $r^k$  a number of times, denoted by  $N_x^k$ . The relation between tasks and resources is described by two functions:  $F(\tau_x)$  returns the set of resources requested by the task  $\tau_x$ , and  $G(r^k)$  returns the set of tasks accessing the resource  $r^k$ . Accesses to shared

resources are assumed to be managed in compliance with either MSRP or MrsP. The reason for such a choice is that MSRP and MrsP represent two mainstream resource-accessing approaches (the non-preemptive and the resource ceiling) with schedulability tests well supported and understood.

In this work, we assume that a task can hold at most one resource at any time instant. Nested critical sections can be trivially supported via group locks [33] — where nested resources are grouped together and managed by one lock — without affecting any analysis or conclusion made in this paper. The finer-grained ordered lock requires some fundamental changes on the underlying schedulability analysis [21] and will be investigated in future work. In addition, the resource accessing time  $c^k$  of a resource  $r^k$  is taken as the worst-case accessing time among all tasks that require the resource. This assumption (also applied in [9]) eases the presentation without affecting the foundations of the schedulability analysis in this paper.

MSRP [19], [20] is developed as the extension of SRP [3] for uniprocessor systems. Under MSRP, resources are accessed from the task's host processor in a non-preemptive fashion. Resource requesting tasks that are not immediately handled will keep spinning non-preemptively 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, as for local resources, PCP is applied so that these resources are executed in a preemptive fashion.

For MrsP [9], a priority ceiling is used instead of the non-preemptive approach for both local and global resources. Under this protocol, a global resource is served in a FIFO order, and has a set of ceiling priorities (one for each processor with tasks requesting that resource). The ceiling priority of a resource  $r^k$  on a given processor  $P_m$  is the maximum priority of all tasks on  $P_m$  that use  $r^k$ . For each request to  $r^k$  on  $P_m$ , the requesting task inherits that ceiling priority during the entire resource access, including the time it spin waits for the resource. Notably, MrsP introduces a helping mechanism where a preempted resource-accessing task can be helped by other tasks waiting (spinning) for the resource, to keep executing rather than waiting in the run queue. This helping mechanism can be realised through either duplicate execution or the task migration approach [9], [21]. In the worst case, a resource-waiting task has to execute on behalf of all other tasks in the FIFO queue each time it requests a resource. This leads to the same worst-case resource accessing time as MSRP [9].

#### IV. SCHEDULABILITY TESTS FOR MSRP AND MRSP

With a resource sharing protocol enforced, a schedulability test  $S$  must be supported to bound the worst-case blocking time due to resource access. With the strong guarantee of resource execution progress (i.e., the non-preemptive fashion) in MSRP and the helping mechanism in MrsP, the schedulability tests of these two protocols are similar, but with different approaches to capture the blocking that tasks can incur upon their arrival [32], [34].

#### A. Traditional Schedulability Tests

Along with the development of MSRP and MrsP, a schedulability test was developed for both protocols to provide a safe bound for blocking due to accessing shared resources [9], [19]<sup>1</sup>. Under both protocols, the response time ( $R_i$ ) of a given task  $\tau_i$  is bounded by Equation (1).  $\overline{C}_i$  is the total computation time of  $\tau_i$  (including the time  $\tau_i$  waits (spins) for and executes with each requested resource).  $B_i$  denotes the amount of blocking  $\tau_i$  can incur upon its arrival. Function  $\mathbf{lh}p(i)$  returns a set of local tasks with a priority higher than the priority of  $\tau_i$ , i.e.,  $pr_i(\tau_i)$ . In addition,  $T_h$  is the period of  $\tau_h$  and  $\overline{C}_h$  is the total computation time of  $\tau_h$  (with the same principals as  $\overline{C}_i$ ).

$$R_i = \overline{C}_i + B_i + \sum_{\tau_h \in \mathbf{lh}p(i)} \left\lceil \frac{R_i}{T_h} \right\rceil \overline{C}_h \quad (1)$$

Notation  $\overline{C}_i$  is bounded through Equation (2). Function  $\sum_{r^k \in F(\tau_i)} N_i^k e^k$  denotes the total time  $\tau_i$  spends on waiting and executing with each requested resource (obtained through  $F(\tau_i)$ ) in one release, where  $N_i^k$  is the number of times  $\tau_i$  requests  $r^k$  in one release and  $e^k$  is the worst-case accessing time to  $r^k$ .

$$\overline{C}_i = C_i + \sum_{r^k \in F(\tau_i)} N_i^k e^k \quad (2)$$

As the requests to a resource under MSRP are served in a non-preemptive FIFO order,  $e^k$  is effectively bounded by the number of processor containing requests to  $r^k$ , as given in Equation (3), in which  $G(r^k)$  denotes the set of tasks that require  $r^k$ ,  $c^k$  for worst-case cost for executing with  $r^k$ , function  $map()$  returns a set of processors where the given tasks are assigned to, and  $|\cdot|$  returns the size of the given set.

As for MrsP, although it adopts a preemptive approach for resource accessing, in the worst case, a task executes on behalf of all other tasks in the FIFO queue before it can execute with the resource (see [9] for MrsP's helping mechanism). Thus, the worst-case blocking time that a MrsP task can incur is also bounded by Equation (3), as discussed in [9].

$$e^k = |map(G(r^k))| * c^k \quad (3)$$

The arrival blocking ( $B_i$ ) of  $\tau_i$  is computed through Equations (4) and (5), where  $\hat{e}_i$  is the arrival blocking  $\tau_i$  can incur with potential remote delay,  $\hat{b}$  is the maximum non-preemptive section in the underlying operating system and  $F^A(\tau_i)$  returns the set of resources that can cause  $\tau_i$  to incur blocking upon its arrival.

$$B_i = max\{\hat{e}_i, \hat{b}\} \quad (4)$$

$$\hat{e}_i = max\{e^k | r^k \in F^A(\tau_i)\} \quad (5)$$

However, as MSRP and MrsP use different priority levels for accessing globally shared resources (i.e., non-preemptive and priority ceiling), the approaches for identifying such resources are different under these protocols. For MSRP, a

<sup>1</sup>We refer to these tests as the traditional schedulability tests for MSRP and MrsP.

global resource  $r^k$  can cause  $\tau_i$  to incur arrival blocking if  $r^k$  is requested by a local lower priority task of  $\tau_i$  (denoted by  $\tau_{ll}$ ). As for a local resource, PCP [28] is enforced so that a resource can cause such blocking if it has a higher priority ceiling than  $\text{pri}(\tau_i)$  and is requested by  $\tau_{ll}$ . Equation (6) yields the set of resources that can cause arrival blocking under MSRP (denote by  $F_{\ddagger}^A(\tau_i)$ ), where  $N_{ll}^k$  gives the number of requests from  $\tau_{ll}$  to  $r^k$  in one release.

$$F_{\ddagger}^A(\tau_i) = \{r^k | N_{ll}^k > 0 \wedge (r^k \text{ is global} \vee \text{pri}(r^k) \geq \text{pri}(\tau_i))\} \quad (6)$$

With MrsP, as both local and global resources are accessed with ceiling priorities,  $F_{\ddagger}^A(\tau_i)$  is simply bounded by finding the resources that have a higher ceiling than  $\text{pri}(\tau_i)$  on  $P(\tau_i)$  (i.e.,  $\tau_i$ 's hosting processor) and are requested by  $\tau_{ll}$ , as given in Equation (7).

$$F_{\ddagger}^A(\tau_i) = \{r^k | N_{ll}^k > 0 \wedge \text{pri}(r^k, P(\tau_i)) \geq \text{pri}(\tau_i)\} \quad (7)$$

The above schedulability tests provide a simple analysis in which a safe blocking bound can be obtained. Notably, with this approach, the worst-case blocking of tasks in one processor can be obtained regardless of the exact resource usage on remote processors (i.e., it is less vulnerable to changes in the system). These highly decoupled schedulability tests are favourable at early design and development stages of systems, where tasks' parameters and execution behaviours may change frequently.

## B. Holistic Schedulability Tests

Despite the advantages of the original schedulability tests described above, these tests rely on the assumption that *each time a task (denoted by  $\tau_i$ ) request a resource, there will always be a remote request on each remote processor in  $\text{map}(G(r^k))$  that can block  $\tau_i$ , regardless of the actual number of possible requests that can be issued within the time period  $R_i$* . In addition, as described in [32], these tests rely on the inflation of blocking time to task execution time, which introduce extra pessimism. Subsequently, [6] proposed an improved schedulability test for several spin-lock protocols (including MSRP) based on Integer Linear Programming<sup>2</sup>. Later on, [34] reformatted the ILP-based analysis to remove the need for any optimisations, and proposed a holistic schedulability test in the context of MrsP, which analyses the total number of resource requests that can be issued within a given time period. This holistic analysis can be directly applied to both MSRP and MrsP with the corresponding  $F^A(\tau_i)$  functions (i.e., Equations (6) and (7) in Section IV-A) adopted.

In the holistic analysis by [34], the response time of  $\tau_i$  is bounded by Equation (8), where  $C_i$  is the pure worst-case computation time of  $\tau_i$  (i.e., without accessing any shared resource),  $E_i$  is the total resource accessing time of  $\tau_i$  with the potential spin delay accounted for and the indirect spin delay (i.e., the transitive blocking) incurred by  $\tau_i$  from each local high priority task  $\tau_h$  (which preempts  $\tau_i$  but is blocked for

requesting a locked resource), and  $B_i$  is the arrival blocking of  $\tau_i$ .

$$R_i = C_i + E_i + B_i + \sum_{\tau_h \in \text{hpl}(i)} \left\lceil \frac{R_i}{T_h} \right\rceil \cdot C_h \quad (8)$$

$E_i$  is obtained through Equation (9), where  $\zeta_i^k$  yields the total number of requests to  $r^k$  issued by  $\tau_i$ 's local higher priority tasks and  $\xi_{i,m}^k$  gives the number of requests to  $r^k$  from a remote processor  $P_m$ . Note that this analysis uses a holistic approach on bounding the blocking time, where the maximum blocking that  $\tau_i$  can incur from a remote processor  $P_m$  due to  $r^k$  during  $\tau_i$ 's release is bounded by the minimal value between  $N_i^k + \zeta_i^k$  (the total number of requests to  $r^k$  from  $\tau_i$  and its local higher priority tasks within the duration of  $R_i$ ) and  $\xi_{i,m}^k$ . By doing so, this analysis is less pessimistic than the traditional test as 1) the analysis computes the exact number of remote requests that can cause the blocking (i.e., avoiding the assumption used by the original tests in Section IV-A) and 2) each critical section (i.e., resource request) is accounted for only once (avoiding inflating tasks' computation times).

$$E_i = \sum_{r^k \in \mathbb{R}} (N_i^k + \zeta_i^k + \sum_{P_m \neq P(\tau_i)} \min\{N_i^k + \zeta_i^k, \xi_{i,m}^k\}) \times c^k \quad (9)$$

$\zeta_i^k$  and  $\xi_{i,m}^k$  are obtained from Equations (10) and (11) respectively. As shown in the equations, a jitter<sup>3</sup> interval (i.e.,  $R_h$  and  $R_j$  respectively) is included to extend the duration of  $\tau_i$ 's release to provide a safe upper bound. As described and proved in [6] (see Lemma 5.1 in [6]), in multiprocessor systems, at most  $\lceil \frac{t+R_x}{T_x} \rceil$  jobs of  $\tau_x$  can be released within a given duration  $t$ . This lemma forms the fundamental approach for calculating the number of requests being issued within a given duration, and is applied by the improved schedulability tests in [32]–[34].

$$\zeta_i^k = \sum_{\tau_h \in \text{hpl}(i)} \left\lceil \frac{R_i + R_h}{T_h} \right\rceil N_h^k \quad (10)$$

$$\xi_{i,m}^k = \sum_{\tau_j \in \Gamma_{P_m}} \left\lceil \frac{R_i + R_j}{T_j} \right\rceil N_j^k \quad (11)$$

The arrival blocking ( $B_i$ ) of  $\tau_i$  (in Equation 8) is computed through Equation (4), but with a different approach for bounding, as provided in Equations (12) and (13).

$$\hat{e}_i = \max\{|\alpha_i^k| \cdot c^k | r^k \in F^A(\tau_i)\} \quad (12)$$

in which  $\alpha_i^k$  denotes the set of processors that contain unaccounted requests to  $r^k$  (i.e., requests that are not accounted for in  $E_i$ ) and is computed as:

$$\alpha_i^k \triangleq \{P_m | \xi_{i,m}^k - \zeta_i^k - N_i^k > 0 \wedge P_m \neq P(\tau_i)\} \cup P(\tau_i) \quad (13)$$

For each  $r^k$  that belongs to  $F^A(\tau_i)$ , this equation identifies the remote processors that contain unaccounted requests (i.e.,  $\xi_{i,m}^k - \zeta_i^k - N_i^k > 0$ ), where a request to  $r^k$  in each of these processors can block  $\tau_i$  upon its arrival. Therefore, by bounding the number of such processors, Equation (12) yields the maximum blocking that  $\tau_i$  can incur among all resources in  $F^A(\tau_i)$ .

<sup>3</sup>An extra time period to safely bound the number of executable jobs a task can have within a given duration.

<sup>2</sup>Referred to as the ILP-based analysis hereafter.

TABLE I: Schedulability tests for MSRP and MrsP

Protocols	traditional test	holistic test
MSRP( $\dagger$ )	$S_{\dagger}$	$S_{\dagger}^{\circ}$
MrsP( $\ddagger$ )	$S_{\ddagger}$	$S_{\ddagger}^{\circ}$

To summarise, in this paper we consider the above schedulability tests (the traditional and the holistic test) for MSRP and MrsP, respectively, to investigate the priority assignment problem in multiprocessor systems. Table I shows the notations used of the resource sharing protocols considered and their schedulability tests. Note that the behaviour of the cache has no impact on resource accesses, and only influences priority assignment via the pure worst-case computation time parameter  $C$ . Our work focuses on resource sharing and assumes that  $C$  is known. Estimating  $C$  needs to consider cache, e.g., in preemption and migration. Runtime overhead of both MSRP and MrsP has been studied in terms of real implementations and schedulability tests [29], [33], [36], where the overhead of context switches and potential migrations is considered. In future work, the analysis of cache-related costs and runtime overheads will be integrated to the schedulability tests, and the impact of such costs on the priority assignment investigated.

## V. DMPO WITH TRADITIONAL SCHEDULABILITY TESTS

To the best of our knowledge, the optimality of DMPO in the presence of blocking holds only for uniprocessor systems [5], [16]. However, whether this algorithm remains optimal in multiprocessor systems with shared resources is unknown. This section investigates the optimality of DMPO in multiprocessor systems with shared resources managed by these protocols. Schedulability tests considered in this section are the traditional schedulability tests described in Section IV-A. The term *Optimal Priority Ordering* is defined in [16].

**Definition 1.** A priority assignment algorithm  $\Lambda$  is optimal with a task model, a scheduling algorithm  $G$ , and a schedulability test  $S$ , if and only if every set of tasks that is compliant with the task model is deemed schedulable with  $G$  by  $S$  under other priority assignments is also schedulable with  $\Lambda$ .

The following base case and inductive step (similar to the proof in [16]) for proving the optimality of DMPO is conducted in a multiprocessor system, with tasksets that comply with the system and task model described in Section III, using schedulability tests  $S_{\dagger}$  and  $S_{\ddagger}$ .

**Base case:** A priority assignment algorithm  $\Lambda^x$  is assumed to be schedulable for a given taskset  $\Gamma$  with  $M$  processors under schedulability test  $S_{\dagger}$  or  $S_{\ddagger}$  (see Figure 1), where  $\Lambda_{\Gamma}^x$  denotes the schedulable priority order for the taskset  $\Gamma$ .

**Inductive step:** under  $\Lambda_{\Gamma}^x$ , a pair of adjacent tasks not in DMPO order are chosen with their priorities swapped to form a new priority ordering, denoted as  $\Lambda_{\Gamma}^{x-1}$ . Then, proof is presented to demonstrate that no tasks have missed their deadlines due to this priority swapping. For a taskset with  $n$  tasks, at most  $x = n(n+1)/2$  priority

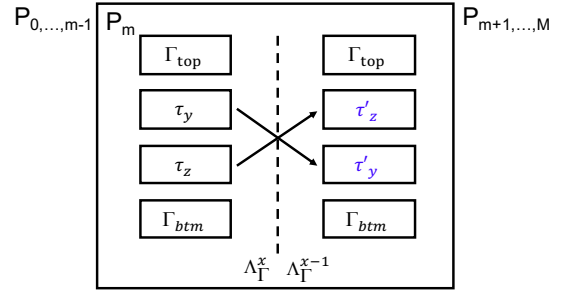


Fig. 1: A priority swap

swapping are required to transfer the priority ordering from  $\Lambda_{\Gamma}^x$  to DMPO in one processor (i.e.,  $\Lambda_{\Gamma}^1 = \text{DMPO}_{\Gamma}$ ). If no tasks have missed their deadlines under  $S_{\dagger}$  and  $S_{\ddagger}$  respectively during the entire priority reordering for all processors, there will be no task sets that are schedulable with  $\Lambda_{\Gamma}^x$  but are not schedulable with DMPO, and hence, proving the optimality of DMPO.

The system shown in Figure 1 is adopted to conduct the proof below. A priority exchange is performed between  $\tau_y$  and  $\tau_z$  on  $P_m$ , where  $\Gamma_{top}$  and  $\Gamma_{btm}$  denote the set of local higher and lower priority tasks respectively. In addition, the system contains a set of shared resources that are managed by MSRP or MrsP, and functions  $F(\tau_x)$  and  $G(r^k)$  given in Section III are adopted here to describe the usage of shared resources. Hence, unlike the proof presented in [5] (which relies on explicit resource usage), the proof in this paper is conducted with generalised resource usage i.e., without any resource sharing assumptions imposed. In addition, it is important to highlight that, under the traditional analysis of both MSRP and MrsP, swap the priorities of two adjacent tasks does not affect the response time of the tasks with a higher (or lower) priority. This is because in the traditional analysis, the response time of a task may depends on the independent properties of higher (or lower) priority tasks, but does not depend on their relative priority ordering [14].

Note that the priority ordering  $\Lambda_{\Gamma}^x$  does not comply with the DMPO algorithm. With DMPO adopted, increasing priorities are assigned in the reverse ordering of deadlines. However, with the priority ordering policy  $\Lambda^x$  adopted, there exists at least one pair of tasks (say  $\tau_1$  and  $\tau_2$ ) that  $D_1 < D_2$  with  $pri(\tau_1) < pri(\tau_2)$ . In this proof, we assume  $D_y > D_z$  and  $pri(\tau_y) > pri(\tau_z)$  under  $\Lambda_{\Gamma}^x$ . To differentiate the response time of  $\tau_y$  and  $\tau_z$  under both priority orderings,  $R'_y$  and  $R'_z$  are used to denote the response time of  $\tau_y$  and  $\tau_z$  after the priority swap i.e., with the priority ordering  $\Lambda_{\Gamma}^{x-1}$ .

**Theorem 1.** DMPO is optimal in fully partitioned multiprocessor deadline-constrained systems in the presence of blocking under  $S_{\dagger}$  (MSRP) or  $S_{\ddagger}$  (MrsP).

*Proof.* For both  $S_{\dagger}$  and  $S_{\ddagger}$ , the response time of a given task is determined by the independent task properties only and the cost for accessing a resource  $r^k$  is always  $|\text{map}(G(r^k))| \times c^k$ , as shown in Equation (3). Below we prove the optimality of DMPO under  $S_{\dagger}$  and  $S_{\ddagger}$  respectively.

**With traditional MSRP analysis  $S_{\dagger}$ :** we firstly prove that  $R'_z \leq D_z$  after the priority swap. Under  $\Lambda_{\Gamma}^x$ ,  $\tau_z$  incurs interference and indirect spin delay from  $\tau_y$ , which is calculated as the following by Equations (2) and (3).

$$\left\lceil \frac{R_z}{T_y} \right\rceil (C_y + \sum_{r^k \in F(\tau_y)} N_y^k \times |\text{map}(G(r^k))| \times c^k) \quad (14)$$

With  $\Lambda_{\Gamma}^{x-1}$ ,  $\tau_z$  will not incur such interference, where it can only be preempted by  $\Gamma_{top}$ . However,  $\tau_z$  could incur a potentially increased arrival blocking due to the priority swap. According to Equation (6), the resources that can cause  $\tau_z$  to incur arrival blocking under  $\Lambda_{\Gamma}^{x-1}$  is identified as

$$F^A(\tau_z) \triangleq \{r^k | r^k \in F(\Gamma_{btm}) \cup F(\tau_y) \wedge (r^k \text{ is global} \vee r^k \in F(\Gamma_{top}) \cup F(\tau_z))\} \quad (15)$$

However, under  $\Lambda_{\Gamma}^x$ ,  $F^A(\tau_z)$  is

$$F^A(\tau_z) \triangleq \{r^k | r^k \in F(\Gamma_{btm}) \wedge (r^k \text{ is global} \vee r^k \in F(\Gamma_{top}) \cup F(\tau_y) \cup F(\tau_z))\} \quad (16)$$

The above calculations illustrate that there could be more resources that can cause  $\tau_z$  to incur arrival blocking with  $\Lambda_{\Gamma}^{x-1}$  than that of  $\Lambda_{\Gamma}^x$ . First, the number of global resources that can cause arrival blocking to  $\tau_z$  after the swapping could be increased, where such resources are  $r^k \in F(\Gamma_{btm})$  in  $\Lambda_{\Gamma}^x$  and  $r^k \in F(\tau_y) \cup F(\Gamma_{btm})$  under  $\Lambda_{\Gamma}^{x-1}$ , assuming  $r^k$  is a global resource. In addition, local resources that are requested by  $\tau_y$  and tasks with a priority equal to or higher than  $\text{pri}(\tau_z)$  (i.e.,  $G(r^k) = \{\Gamma_{top}, \tau_y\}$ ,  $G(r^k) = \{\tau_y, \tau_z\}$  or  $G(r^k) = \{\Gamma_{top}, \tau_y, \tau_z\}$ ) can now block  $\tau_z$  upon its arrival under  $\Lambda_{\Gamma}^{x-1}$ . Accordingly, the resources that can cause  $\tau_z$  to incur an increased arrival blocking is summarised as follows:

$$\{r^k | r^k \in F(\tau_y) \wedge (r^k \text{ is global} \vee r^k \in F(\Gamma_{top}) \cup F(\tau_z))\} \quad (17)$$

Therefore, in the worst case,  $\tau_z$ 's arrival blocking can increase after the priority swap due to the resources identified above. However, as shown above, the resource (say  $r^k$ ) that causes this increased arrival blocking is one of  $\tau_y$ 's requested resources (i.e.,  $r^k \in F(\tau_y)$ ), regardless whether  $r^k$  is a global or local resource.

Recall the decreased indirect spin delay of  $\tau_z$  after the priority swap in Equation (14). The potential increase of the arrival blocking of  $\tau_z$  after the priority swap is at most equal to the guaranteed decrease of its indirect spin delay, as the arrival blocking can occur only once (i.e.,  $|\text{map}(G(r^k))| \times c^k$ ) whilst  $\tau_y$  could access that resource at least once during each release.

In addition, as  $\tau_z$  will not incur the interference of  $\tau_y$ 's pure computation time (i.e.,  $C_y$ ) after the priority swap, the increase of the arrival blocking of  $\tau_z$  is always less than the decrease of its interference after the priority swap i.e.,  $R'_z < R_z \leq D_z$ .

On the other side, if  $\tau_z$ 's arrival blocking is not increased after the priority swap, it is still schedulable as it incurs less interference with the same amount of direct spin delay and a non-increased arrival blocking. Therefore,  $\tau_z$  remains schedulable under  $\Lambda_{\Gamma}^{x-1}$  with schedulability test  $S_{\dagger}$  regardless of the exact resource usage.

Now we prove that  $R'_y \leq D_y$ . This can be achieved by comparing  $R_z$  and  $R'_y$ . Firstly, we observe that  $\tau_z$  in  $\Lambda_{\Gamma}^x$  and  $\tau_y$  in  $\Lambda_{\Gamma}^{x-1}$  can be blocked upon their arrival by the same set of resources (see Equation (16)), and hence, leads to the same amount of arrival blocking, denote as  $B$  below. To ease the comparison, we ignore the interference from the tasks in  $\Gamma_{top}$  for the time being, and will consider this interference later on. Such an approach is valid because the amount of interference from high priority tasks increases monotonically with the increase of response time, where  $R_1 \geq R_2$  then  $\left\lceil \frac{R_1}{T_x} \right\rceil C_x \geq \left\lceil \frac{R_2}{T_x} \right\rceil C_x$  for  $\tau_x$  and vice versa.

The following presents the calculations of response time of  $\tau_z$  with  $\Lambda_{\Gamma}^x$  and  $\tau_y$  under  $\Lambda_{\Gamma}^{x-1}$  respectively.

$$\begin{aligned} R_z &= \overline{C}_z + B + \left\lceil \frac{R_z}{T_y} \right\rceil \overline{C}_y \\ &= \overline{C}_z + B + N_p \times \overline{C}_y \end{aligned} \quad (18)$$

For  $\tau_z$  in  $\Lambda_{\Gamma}^x$ , as no assumption can be made between  $R_z$  and  $T_y$  so that  $\tau_y$  could preempt  $\tau_z$  more than once, where  $N_p$  denotes the number of such preemptions and  $N_p = \left\lceil \frac{R_z}{T_y} \right\rceil \geq 1$ .

$$\begin{aligned} R'_y &= \overline{C}_y + B + \left\lceil \frac{R'_y}{T_z} \right\rceil \overline{C}_z \\ &= \overline{C}_y + B + \left\lceil \frac{\overline{C}_y + B}{T_z} \right\rceil \overline{C}_z \\ &= \overline{C}_y + B + \overline{C}_z \end{aligned} \quad (19)$$

For  $\tau_y$  under  $\Lambda_{\Gamma}^x$ , the recursive calculation of  $R'_y$  starts from  $R'_y = \overline{C}_y + B$ . From the calculation in Equation (18), we observe that  $\overline{C}_y + B < R_z \leq D_z \leq T_z$  ( $\tau_z$  is schedulable under  $\Lambda_{\Gamma}^{x-1}$ ). Therefore,  $R'_y$  is updated as  $\overline{C}_y + B + \overline{C}_z$ , with  $\left\lceil \frac{\overline{C}_y + B}{T_z} \right\rceil = 1$ . With further recursive calculations of  $R'_y$ ,  $R'_y$ 's value is not changed because  $R'_y = \overline{C}_y + B + \overline{C}_z \leq R_z \leq D_z \leq T_z$  so that  $\left\lceil \frac{R'_y}{T_z} \right\rceil$  is fixed to 1.

Thus, it is now clear that  $R'_y \leq R_z$ , assuming no interference from tasks in  $\Gamma_{top}$ . With such interface considered,  $\left\lceil \frac{R'_y}{T_h} \right\rceil \overline{C}_h$  is at most equal to  $\left\lceil \frac{R_z}{T_h} \right\rceil \overline{C}_h$  for each  $\tau_h$  in  $\Gamma_{top}$ , given  $R'_y \leq R_z$ . This further supports the conclusion that  $R'_y \leq R_z$ . Therefore, we prove that  $\tau_y$  remains schedulable after the priority swap as  $R'_y \leq R_z \leq D_z < D_y$  with any resource usage applied.

**With traditional MrsP analysis  $S_{\ddagger}$ :** With the traditional MrsP test, the only difference from  $S_{\dagger}$  is that, the resources that can cause tasks to incur arrival blocking are determined by the ceiling priority of shared resources, as given through Equation (7). Accordingly, the set of resources that can cause arrival blocking to  $\tau_y$  and  $\tau_z$  under both priority orderings with MrsP adopted are identified below.

With the priority ordering  $\Lambda_{\Gamma}^x$ , resources that can cause arrival blocking of  $\tau_y$  is:

$$F_{\ddagger}^A(\tau_y) \triangleq \{r^k | r^k \in F(\Gamma_{btm}) \cup F(\tau_z) \wedge r^k \in F(\Gamma_{top}) \cup F(\tau_y)\} \quad (20)$$

and the resources that can cause arrival blocking of  $\tau_z$  is:

$$F_{\ddagger}^A(\tau_z) \triangleq \{r^k | r^k \in F(\Gamma_{btm}) \wedge r^k \in F(\Gamma_{top}) \cup F(\tau_y) \cup F(\tau_z)\} \quad (21)$$

Under the priority ordering  $\Lambda_{\Gamma}^{x-1}$ ,  $F_{\ddagger}^A(\tau_y)$  becomes

$$F_{\ddagger}^A(\tau_y) \triangleq \{r^k | r^k \in F(\Gamma_{btm}) \wedge r^k \in F(\Gamma_{top}) \cup F(\tau_z) \cup F(\tau_y)\} \quad (22)$$

and  $F_{\ddagger}^A(\tau_z)$  is calculated as:

$$F_{\ddagger}^A(\tau_z) \triangleq \{r^k | r^k \in F(\Gamma_{btm}) \cup F(\tau_y) \wedge r^k \in F(\Gamma_{top}) \cup F(\tau_z)\} \quad (23)$$

Similar with the  $S_{\ddagger}$  case,  $\tau_z$  could incur an increased arrival blocking due to a resource that is requested by  $\tau_y$  (i.e., in  $F(\tau_y)$ ) after the priority swap. As proved before under  $S_{\ddagger}$ , the potential increase of  $\tau_z$ 's arrival blocking is always less than the guaranteed decrease of the interference it suffers. Thus,  $\tau_z$  remains schedulable in  $\Lambda_{\Gamma}^{x-1}$  i.e.,  $R'_z < R_z \leq D_z$ , based on  $S_{\ddagger}$ .

In addition, the set of resources that can cause  $\tau_z$  in  $\Lambda_{\Gamma}^x$  and  $\tau_y$  in  $\Lambda_{\Gamma}^{x-1}$  to incur arrival blocking are identical, as shown in Equations (21) and (22). That is, the calculations of  $R_z$  and  $R'_y$  under  $S_{\ddagger}$  are identical with the  $S_{\ddagger}$  case, see Equations (18) and (19). Therefore, it also leads to the conclusion that  $R'_y \leq R_z \leq D_z < D_y$  after the priority swap. Accordingly, the DMPO algorithm remains optimal under  $S_{\ddagger}$  with the considered system and task model.  $\square$

With the above proof, we provide evidence that two adjacent tasks that are schedulable under priority  $\Lambda_{\Gamma}^x$  remain schedulable after swapping their priorities (i.e., in  $\Lambda_{\Gamma}^{x-1}$ ), according to the traditional schedulability tests. By swapping all the adjacent tasks with the incorrect priority order in each processor in Figure 1 according to DMPO, a schedulable system with the DMPO algorithm can be obtained.

Summarise the above, we conclude that DMPO remains optimal in multiprocessor systems with shared resources under the traditional schedulability tests of MSRP or MrsP, in which the response time is decided only by independent task parameters (e.g.,  $C_i$  and  $T_i$ ) and the accessing time of a shared resource is fixed by the number of processor that contain tasks that request that resource.

## VI. PRIORITY ASSIGNMENTS UNDER HOLISTIC SCHEDULABILITY TESTS

As described in Section IV-B, the holistic schedulability test is less pessimistic than the traditional analysis by bounding the exact number of executable tasks within a given duration. However, whilst obtaining tighter schedulability results, *response time dependencies* are also introduced, where the response time of a given task potentially depends on the response time of all other tasks in the system. Under such a schedulability test, optimality of the DMPO and applicability of the existing OPA-like search-based priority assignments could be jeopardised. In this section, we investigate the optimality and applicability of DMPO, OPA and RPA, with the holistic schedulability tests  $S_{\ddagger}^{\diamond}$  and  $S_{\ddagger}^{\circ}$  given in Table I.

### A. Optimality of DMPO

We prove that the DMPO algorithm is not optimal with holistic schedulability tests by deriving two counter examples, as described in the following theorem and proof.

**Theorem 2.** *DMPO is not optimal in fully partitioned multiprocessors constrained-deadline systems with shared resources managed by MSRP or MrsP under  $S_{\ddagger}^{\diamond}$  (MSRP) or  $S_{\ddagger}^{\circ}$  (MrsP).*

*Proof.* This proof is conducted by counter examples. Figure 2 shows a three-processor system with two shared resources  $r^1$  and  $r^2$ . Table II gives the task property and resource usage in processor  $P_1$ . A priority swap is performed between  $\tau_2$  and  $\tau_3$ . Under priority ordering  $W^x$ ,  $pri(\tau_3) > pri(\tau_2) > pri(\tau_1)$  with  $D_3 > D_2$ , i.e., the task with a longer deadline is assigned with a higher priority. With DMPO,  $pri(\tau_2) > pri(\tau_3) > pri(\tau_1)$  so that priorities are assigned in the reverse order of deadlines. In addition, we assume there exists sufficient requests to  $r^1$  and  $r^2$  from both  $P_0$  and  $P_2$  so that the cost of accessing  $r^1$  (or  $r^2$ ) from  $P_1$  is always  $3c^1$  (or  $3c^2$ ).

Under MSRP, both  $r^1$  and  $r^2$  can cause arrival blocking to  $\tau_2$  and  $\tau_3$  in both priority orderings due to the non-preemptive resource accessing approach so that  $F^A(\tau_2) = F^A(\tau_3) = \{r^1, r^2\}$  in  $\Lambda_{\Gamma}^x$  and DMPO. The response time calculation of  $\tau_3$  and  $\tau_2$  under priority ordering  $\Lambda_{\Gamma}^x$  is shown below.

$$\begin{aligned} R_3 &= C_3 + E_3 + B_3 \\ &= C_3 + 3c^1 + \max\{3c^1, 3c^2\} \\ &= 1 + 3 \times 1 + 3 \times 2 \\ &= 10 \\ R_2 &= C_2 + E_2 + B_2 + \left\lceil \frac{R_2}{T_3} \right\rceil C_3 \\ &= C_2 + 3c^2 + \max\{3c^1, 3c^2\} + \left\lceil \frac{R_2}{T_3} \right\rceil C_3 + \left\lceil \frac{R_2 + R_3}{T_3} \right\rceil 3c^1 \\ &= 1 + 3 \times 2 + 3 \times 2 + \left\lceil \frac{R_2}{27} \right\rceil \times 1 + \left\lceil \frac{R_2 + 10}{27} \right\rceil \times 3 \\ &= 17 \end{aligned}$$

From the above calculations, we can see that both tasks are able to meet their deadlines before the priority swap. However, after the swap,  $R'_3$  has missed its deadline, as computed below.

$$\begin{aligned} R'_2 &= C_2 + E_2 + B_2 \\ &= C_2 + 3c^2 + \max\{3c^1, 3c^2\} \\ &= 1 + 3 \times 2 + 3 \times 2 \\ &= 13 \\ R'_3 &= C_3 + E_3 + B_3 + \left\lceil \frac{R'_3}{T_2} \right\rceil C_2 \\ &= C_3 + 3c^1 + \max\{3c^1, 3c^2\} + \left\lceil \frac{R'_3}{T_2} \right\rceil C_2 + \left\lceil \frac{R'_3 + 13}{17} \right\rceil 3c^2 \\ &= 1 + 3 \times 1 + 3 \times 2 + \left\lceil \frac{R'_3}{17} \right\rceil \times 1 + \left\lceil \frac{R'_3 + 13}{17} \right\rceil \times (3 \times 2) \\ &= 30 \end{aligned}$$

With MrsP's holistic test (i.e.,  $S_{\ddagger}^{\diamond}$ ), the situation is similar but with different response time values. Under this protocol (which uses priority ceiling),  $F_{\ddagger}^A(\tau_3) = \{r^1\}$  in  $\Lambda_{\Gamma}^x$  and  $F_{\ddagger}^A(\tau_2) = \{r^2\}$  after the priority swap. However, it does not change the fact that both tasks are schedulable before the swap ( $R_3 = 7$ ,  $R_2 = 17$ ) but  $\tau_3$  misses its deadline after having its priority reduced ( $R'_2 = 13$  and  $R'_3 = 30$ ).

Based on the above counter-example, we demonstrate that DMPO is not optimal under the holistic test due to response time dependencies in Equation (10), which are introduced to reduce the pessimism for inflating task execution time.

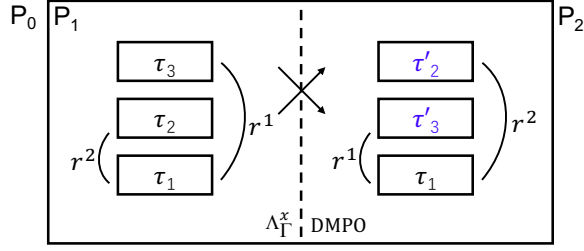


Fig. 2: A priority swap: Example one

TABLE II: Task properties and resource usage in  $P_1$  of the system in Figure 2

Task ( $\tau_x$ )	$C_x$	$T_x$	$D_x$
$\tau_2$	1	17	17
$\tau_3$	1	27	27
Resource ( $r^k$ )	$c^k$	$G(r^k)$	$N_x^k$
$r^1$	1	$\{\tau_1, \tau_3\}$	$N_1^1 = 1, N_3^1 = 1$
$r^2$	2	$\{\tau_1, \tau_2\}$	$N_1^2 = 1, N_2^2 = 1$

However, the above calculations are performed with the assumption that the cost of each access to  $r^k$  is  $|\text{map}(G(r^k))|$ . Below we provide another example demonstrating that the optimality of DMPO can also be undermined due to the response time dependency from remote tasks (in Equation (11)), which is applied to minimise the pessimism from using a constant upper bound when computing the cost of accessing shared resources. Figure 3 presents a dual-processor system with three tasks and a resource ( $r^1$ ) shared between two processors. Table III gives the task parameter and resource usage of this system. A priority swap is performed between  $\tau_1$  and  $\tau_2$  on  $P_0$ , as given in Figure 3. In this example,  $\text{pri}(\tau_1) > \text{pri}(\tau_2)$  before the priority swap whilst  $\text{pri}(\tau_2) > \text{pri}(\tau_1)$  after the swap.

Under  $\Lambda_{\Gamma}^x$ , no task will incur arrival blocking (i.e.,  $\hat{e} = 0$  for all tasks) with either MSRP or MrsP applied, as only  $\tau_1$  and  $\tau_3$  request  $r^1$ . Thus, the response time of all three tasks under both  $S_{\dagger}^{\circ}$  and  $S_{\ddagger}^{\circ}$  is identical, and all tasks are able to

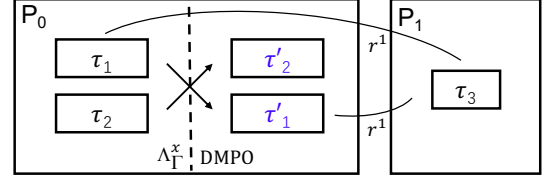


Fig. 3: A priority swap: Example two

TABLE III: Task properties and resource usage of the system in Figure 3

Task ( $\tau_x$ )	$P_x$	$C_x$	$T_x$	$D_x$
$\tau_1$	0	2	28	28
$\tau_2$	0	5	20	20
$\tau_3$	1	3	35	20
Resource ( $r^k$ )	$c^k$	$G(r^k)$	$N_x^k$	
$r^1$	4	$\{\tau_1, \tau_3\}$	$N_1^1 = 1, N_3^1 = 3$	

meet their deadlines, as shown below.

$$\begin{aligned}
R_1 &= C_1 + E_1 \\
&= C_1 + N_1^1 c^1 + \min\{N_1^1 c^1, \lceil \frac{R_1 + R_3}{T_3} \rceil N_3^1 c^1\} \\
&= 2 + 4 + \min\{4, \lceil \frac{R_1 + 18}{35} \rceil \times 12\} \\
&= 10 \\
R_2 &= C_2 + E_2 + \lceil \frac{R_2}{T_1} \rceil C_1 \\
&= C_2 + \min\{\lceil \frac{R_2 + R_1}{T_1} \rceil N_1^1 c^1, \lceil \frac{R_2 + R_3}{T_3} \rceil N_3^1 c^1\} \\
&\quad + \lceil \frac{R_2 + R_1}{T_1} \rceil N_1^1 c^1 + \lceil \frac{R_2}{T_1} \rceil C_1 \\
&= 5 + \min\{\lceil \frac{R_2 + 10}{28} \rceil 4, \lceil \frac{R_2 + 18}{35} \rceil 12\} + \lceil \frac{R_2 + 10}{28} \rceil 4 \\
&\quad + \lceil \frac{R_2}{28} \rceil \times 2 \\
&= 15 \\
R_3 &= C_3 + E_3 \\
&= C_3 + N_3^1 c^1 + \min\{N_3^1 c^1, \lceil \frac{R_3 + R_1}{T_1} \rceil N_1^1 c^1\} \\
&= 2 + 12 + \min\{3 \times 4, \lceil \frac{R_3 + 10}{28} \rceil \times 1 \times 4\} \\
&= 18
\end{aligned}$$

However, after the priority swap (i.e., with the DMPO applied),  $\tau_3$  misses its deadline due to a response time increase of  $\tau_1$ , as shown below with  $S_{\dagger}^{\circ}$ .

$$\begin{aligned}
R'_2 &= C_2 + B_1 \\
&= 5 + |\{P_0, P_1\}| \times 4 \\
&= 13 \\
R'_1 &= C_1 + E_1 + \left\lceil \frac{R'_1}{T_2} \right\rceil C_2 \\
&= 2 + 1 \times 4 + \min\{1 \times 4, \left\lceil \frac{R'_1 + 22}{35} \right\rceil \times 12\} + \left\lceil \frac{R'_1}{28} \right\rceil \times 5 \\
&= 15 \\
R'_3 &= C_3 + E_3 \\
&= 2 + 12 + \min\{12, \left\lceil \frac{R'_3 + 15}{28} \right\rceil \times 4\} \\
&= 22
\end{aligned}$$

Under  $S_{\ddagger}^{\circ}$ , the only difference is that  $\tau_2$  now does not incur arrival blocking due to the resource ceiling facility, and hence,  $R'_2 = 5$ . However, this does not affect the values of  $R'_1$  and  $R'_3$ , where  $\tau_3$  still misses its deadline. Note, in this example,  $\left\lceil \frac{R_2 + R_1}{T_1} \right\rceil = \left\lceil \frac{R_2}{T_1} \right\rceil$  so that no extra jobs of  $\tau_1$  are executed during  $R_2$  due to the jitter introduced in Equation (10). Thus, the non-optimality of DMPO in this example is caused only by the response time dependency from remote processors.  $\square$

### B. Applicability of OPA-like Priority Assignments

As for the OPA-like search-based algorithms reviewed in Section II (e.g., OPA and RPA), their optimality hold as long as the given schedulability test is applicable. In [14], three application conditions are formalised for both OPA and RPA (including their extensions):

- 1: “The schedulability of a task  $\tau_x$  may, according to test  $S$ , depend on any independent properties of tasks with priorities higher than  $\text{pri}(\tau_x)$ , but not on any properties of tasks that depend on their relative priority ordering.”
- 2: “The schedulability of a task  $\tau_x$  may, according to test  $S$ , depend on any independent properties of tasks with priorities lower than  $\text{pri}(\tau_x)$ , but not on any properties of those tasks that depend on their relative priority ordering.”
- 3: “When the priorities of any two tasks of adjacent priority are swapped, the task being assigned the higher priority cannot become unschedulable according to test  $S$ , if it was previously schedulable at the lower priority.”

Conditions 1 and 2 are violated by holistic schedulability tests, where the response time of a task depends on response time of potentially all other tasks in the system and such dependency can become a circular chain. For instance,  $\tau_1$  and  $\tau_2$  are allocated to different processors and share the same resource. Thus, the calculation of response time of either task requires the response time of the other. The OPA-like algorithms attempt to get fixed response time with the assumption that all the unexamined tasks (i.e., tasks that have not been assigned with a priority) have higher priorities, but without assuming any exact priority ordering for those tasks. However, with the holistic tests (and so for the ILP-based tests), the response time of tasks in a system must be calculated

iteratively and alternatively until response times of all tasks is fixed with explicit task priorities (assuming the system is schedulable). Therefore, for OPA and RPA, it is not possible to obtain the exact response time of a given task without knowing the response time of the local higher priority tasks and the remote tasks that share the same resources, under the holistic schedulability tests.

Nonetheless, by replacing the jitter parameter (i.e.,  $R_h$  and  $R_j$  in Equations (10) and (11) respectively) to an independent task property (e.g., deadlines), these search-based priority assignment algorithms can be applied as the schedulability tests now satisfy all application conditions described above. We denote such an approach as OPA-D and RPA-D hereafter. However, the concern with such an approach is that, by replacing  $R_x$  with  $D_x$ , considerable pessimism can be imposed on the schedulability results as  $R_x \leq D_x$  in a schedulable system, which could lead to unschedulable results for systems that are actually feasible. Furthermore, as compromises must be made to the schedulability test, the optimality of these priority ordering algorithms could also be undermined. In Section VIII, evaluations are conducted to investigate the performance of OPA-D and RPA-D and to provide evidence that these compromised priority ordering algorithms under tighter holistic schedulability tests are not optimal.

## VII. SPO: A SLACK-BASED PRIORITY ORDERING ALGORITHM

As proved in Section VI, DMPO is not optimal under the holistic schedulability tests of MSRP or MrsP due to response time dependencies. In addition, OPA-like algorithms are not applicable without compromises, which could undermine the accuracy of the schedulability tests, which in turn undermines their optimality (see evidence in Section VIII). Thus, with holistic tests, there could be cases where a system that is actually schedulable with a certain priority ordering, but which cannot be found by either the static (i.e., DMPO) or search-based (i.e., OPA-like) priority assignment algorithms considered in this paper. In addition, due to the high degree of response time dependency in the holistic tests, optimal priority assignments may not be achievable as the response time of a given task cannot be fixed at a given priority level without knowing the priority and response time of other tasks.

In this section, a new priority ordering algorithm is developed, namely Slack-based Priority Ordering (SPO). SPO shares a similar philosophy to OPA-like algorithms, which examine each priority level (starting from the lowest priority) and assigns the priority to a task from amongst all the unexamined tasks (i.e., tasks that are not assigned a priority). However, unlike the OPA-like algorithms, a response time approximation approach is introduced in SPO to minimise the pessimism incurred due to its use in holistic schedulability tests.

In essence, SPO takes into consideration the response time dependencies and aims at minimising the pessimism in the

<sup>4</sup>If tasks have the same  $\lambda$ , the task with longest deadline is assigned with the priority.

<sup>5</sup>In this paper, the extension parameter  $\eta$  is set to 5.

---

**Algorithm 1: The SPO Algorithm**

---

```
1 for  $m=0, \dots, M$  do
2   for each priority level  $Pri$ , lowest first do
3     for  $\tau_x \in$  unexamined tasks on  $P_m$  do
4       Assuming that all unassigned tasks in  $P_m$ 
         have higher priorities, calculate  $\lambda_x$  for each
         unexamined task  $\tau_x$  by Algorithm 2;
       end
5     Assign priority  $Pri$  to the task with the largest
        $\lambda^+$ ;
     end
6   Get response times of all tasks in  $P_m$  via test  $S$ , and
     set  $R = D$  for all unexamined tasks and tasks with
      $R > D$  on  $P_m$ ;
   end
7 Get response time of all tasks in the system via  $S$ ;
8 if system is schedulable then
9   return true;
   else
10  return false;
   end
```

---

---

**Algorithm 2: Computing  $\lambda_x$  for  $\tau_x$** 

---

```
1 Set  $R = D$  for each unexamined remote task;
2 Calculate response times of all tasks in  $P(\tau_x)$  iteratively
   and alternately by test  $S$ . The calculation ends when  $R$ 
   is fixed for each task in  $P_m$ , or  $R \geq \eta \cdot D$  for all other
   tasks that missed their deadlines;5
3 return  $D_x - R_x$ ;
```

---

response time calculation with such dependencies. Compared with OPA-D and RPA-D, which simply assume  $R = D$  for all other tasks when computing  $R_i$  of a given task  $\tau_i$ , SPO gradually replaces this pessimistic upper bound for tasks achieving a fixed point of response time during the iterative process, which will be explained below in detail. Therefore, SPO generally yields more accurate response times than OPA-D and RPA-D. In addition, when computing  $R_i$ , SPO calculates the response time of all tasks on  $\tau_i$ 's processor in a holistic fashion (detailed description below). Such an approach completely avoids the pessimism caused by the response time dependency between  $\tau_i$  and its local higher-priority tasks. By contrast, in OPA-D and RPA-D, this pessimism exists during the entire priority assignment process.

Algorithm 1 gives the pseudo code of the SPO algorithm, where  $S$  denotes the underlying schedulable test. As shown in the algorithm, SPO assumes that tasks are pre-allocated to each processor before applying priority ordering. For a system with  $M$  processors, the algorithm starts from the first processor  $P_0$  and yields a priority ordering for tasks in each processor. Optimising the order for iterating the processors will be conducted in future work, to further improve the performance of the proposed algorithm. For a given processor  $P_m$ , the algorithm calculates the free capacity  $\lambda_x$  (where  $\lambda_x = D_x - R_x$  for  $\tau_x$ ) of all unexamined tasks on  $P_m$

for each priority level  $Pri$  by Algorithm 2. Then, the task with the maximum free capacity  $\lambda_x$  is assigned with the priority (line 4). After examining each task in  $P_m$ , the response time of tasks in this processor is calculated in line 6. Then, the algorithm repeats the above procedures for tasks in each processor. Finally, with each task assigned with a priority, SPO verifies whether the given priority ordering can lead to a schedulable system by schedulability test  $S$ .

Unlike OPA-D and RPA-D described in Section VI-B, the SPO algorithm minimises the pessimism due to the assumption that *the response times of all other tasks are equal to their deadlines*. First, to estimate the response time of a given task, SPO calculates its response time in the context of its hosting processor. That is, the response time of a given task is calculated by iterating and alternating the response time calculation of all tasks in that processor. Specifically, in each iteration, the response time of every task is computed once, starting from the highest-priority one. Taking a partition of three tasks  $\tau_1, \tau_2, \tau_3$  with  $pri(\tau_1) > pri(\tau_2) > pri(\tau_3)$  as an example, the computation of  $R_2$ , for instance, is performed by iterating the response time calculations of all three tasks, to cope with the discussed response time dependency. In each iteration, the response time of  $\tau_1, \tau_2$ , and  $\tau_3$  is computed once in this order to update  $R_2$ . The iterative process is terminated when for all tasks, a fixed point is reached or  $R \geq \eta \cdot D$ . In this way, the response time dependencies from local higher priority tasks (see Equation (10)) are considered (see line 6 in Algorithm 1 and line 2 in Algorithm 2), and hence, the pessimism caused by ignoring this dependency (e.g., the OPA-D approach) is eliminated.

As for the response time dependency from remote tasks (Equation (11)), this algorithm holds the assumption of  $R = D$  only for unexamined remote tasks (i.e., tasks that are not yet assigned a priority). For instance, when assigning priorities to tasks in  $P_5$ , the response times of tasks in  $P_0$  to  $P_4$  (calculated by line 6 in previous rounds) are used to calculate the response time of tasks in  $P_5$ , instead of their deadlines. By doing so, more accurate (less pessimistic) response times of tasks in  $P_5$  could be obtained in general as  $R \leq D$  for all examined tasks.

To this end, the time complexity of SPO can be determined. Although the algorithm contains a three-level nested loop, in total, at most  $n(n+1)/2$  calls (similar with OPA and RPA) to line 4 is required to assign priorities to a taskset with  $n$  tasks. As for line 4, instead of one invocation to  $S$  with OPA and RPA applied, SPO issues up to  $n$  invocations to test  $S$  to get a response time approximation for each task. Therefore, the time complexity of the SPO algorithm is  $O(n^3)$ ; but note the non-polynomial time complexity of the actual schedulability test  $S$ .

In addition, unlike OPA and RPA, SPO allows the situation where all the tasks miss their deadlines for a given priority level. As the response times of unexamined remote tasks are assumed to be their deadlines, there can be cases where the response times of tasks are higher than their deadlines under a given priority level, but some of the tasks are actually schedulable. In this situation, the SPO algorithm aims to assign the priority to the task that is closet to be schedulable among these deadline-missing tasks. To achieve this, an extension

parameter  $\eta$  is introduced to extend the iterative response time calculations, where the calculation ends when the response time has elapsed to  $\eta \cdot D$  for all deadline-missing tasks except  $\tau_x$  (the currently studied task), as shown in Algorithm 2.

The implication is that for a given priority level, there could be several tasks with response times that are slightly higher than their deadlines. However, with further iterative calculations, variances between the  $\lambda$  value of these tasks could be revealed or magnified, where the task with the highest  $\lambda$  value is closest to be schedulable at this priority level in general, compared to other tasks. Therefore, by using this approach, tasks are more likely to be assigned appropriate priorities so that the possibility for obtaining a feasible priority ordering is increased. Such an approach has also been applied in [26] in a different context and with its effectiveness proved, where the response time calculation of tasks is extended for deadline-missing tasks to determine the system configuration that is closest to be schedulable.

### VIII. EXPERIMENTAL RESULTS

The above sections have investigated the optimality and applicability of existing major priority ordering approaches. To this end, we have concluded that DMPO is not optimal under the holistic tests described in Section IV-B whilst both OPA and RPA suffer from extra pessimism in order to be applicable with such tests (i.e., the OPA-D and RPA-D approaches). Then, a new search-based priority ordering algorithm (SPO) is developed that minimises the pessimism arising from adopting the holistic schedulability tests. In this section, experiments are conducted with these priority ordering algorithms to 1) investigate the impact of priority assignment on the schedulability of multiprocessor systems with shared resources, and to 2) compare the performance (in terms of resulting system schedulability) between these priority assignment algorithms<sup>6</sup>.

The experimental setup is similar to that of the ILP-based analysis work in [32], which covers a large scale of system configurations. We consider platforms with up to  $M = 16$  processors, and up to  $n = 10M$  tasks with a total utilisation  $U = 0.1n$ . Tasks are pre-allocated to processors based on the Worst-Fit heuristic. Task periods are randomly chosen between  $[1ms, 1000ms]$  in a log-uniform distribution. In this evaluation, we assume that the deadline of the tasks are equal to their periods ( $D = T$ ). The utilisation of each task is computed based on the UUniFast-Discard algorithm in [4] and subsequently, the total computation time (including the time executing with shared resources) for each task can be obtained, denoted as  $C'_x$ . In addition, tasks in each system share either  $M/2$ ,  $M$  or  $2M$  resources. A wide range of critical section length  $L = [1\mu s, 500\mu s]$  is supported. A real value parameter  $\kappa$  is introduced to specify the number of tasks on each processor that can access resources (i.e.,  $\lfloor \kappa \cdot n \rfloor$ ), where  $\kappa \in [0.0, 1.0]$ . A task will issue requests to a number of randomly chosen resources, but limited to the range

<sup>6</sup>Due to the large volume of results (1440 system configurations in total), in this paper we focus on major trends and selectively present four experiments that exhibit the discussed effects. The complete results for all system configurations are publicly accessible online at <https://github.com/omitted12345/PriorityAssignments> with experimental implementations available.

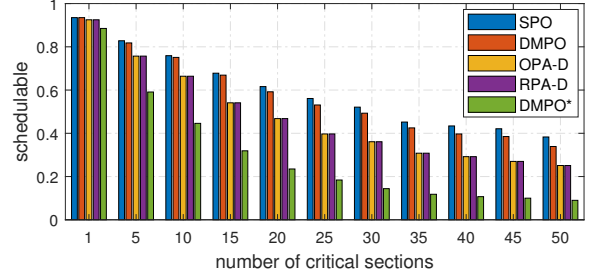


Fig. 4: Schedulability of MSRP systems for  $M = 16$ ,  $n = 64$ ,  $\kappa = 0.4$ ,  $L = [1\mu s, 15\mu s]$  and  $M$  shared resources

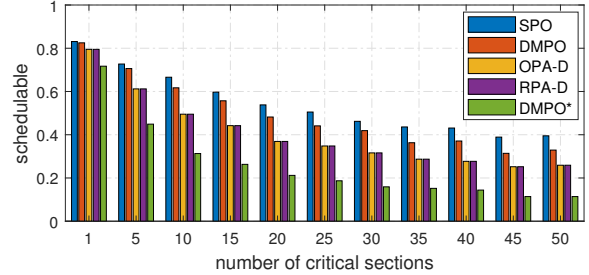


Fig. 5: Schedulability of MrsP systems for  $M = 16$ ,  $n = 64$ ,  $\kappa = 0.25$ ,  $L = [50\mu s, 100\mu s]$  and  $M$  shared resources

$[1, M]$ . The number of requests is randomly decided between  $[1, A]$ , where  $A = \{1, 5, 10, 20\}$  unless specified otherwise. Let  $C_x^{\mathbb{R}}$  be the total resource computation time of  $\tau_x$  (i.e.,  $C_x^{\mathbb{R}} = \sum_{r^k \in F(\tau_x)} N_x^k \cdot c^k$ ), we enforce that  $C'_x - C_x^{\mathbb{R}} \geq 0$  and set  $C_i = C'_x - C_x^{\mathbb{R}}$ .

Figure 4 and Figure 5 (for MSRP and MrsP respectively) shows the percentage of schedulable systems among 1000 randomly generated input systems by each priority ordering algorithm. The experiments are conducted by varying  $A$  (resource request frequency). The holistic schedulability test is applied in SPO, DMPO, OPA-D and RPA-D and the traditional test is used in DMPO\* (see labels in the figures). As observed in both graphs, although the DMPO is optimal with the traditional schedulability test, this priority assignment strategy (i.e., DMPO\*) does not perform well (by comparison) due to the pessimism of the analysis itself (recall Section IV). Indeed this combination yields the worst performance among all examined algorithms. In addition, among priority assignments with the holistic tests, OPA-D and RPA-D exhibit the worst performances, which illustrates that the pessimism introduced for the use of the holistic tests significantly affects the performance of these search-based algorithms. In addition, the RPA algorithm yields the same schedulability as OPA. This is to be expected as RPA aims at providing robust priority ordering for systems that are deemed to be schedulable. For such systems, a schedulable priority ordering can also be found by OPA. This also provides evidence that OPA and RPA are not optimal with holistic tests.

However, although SPO inherits the philosophy of OPA and RPA, in both figures it demonstrates the best performance in general, where it outperforms OPA-like algorithms in all cases.

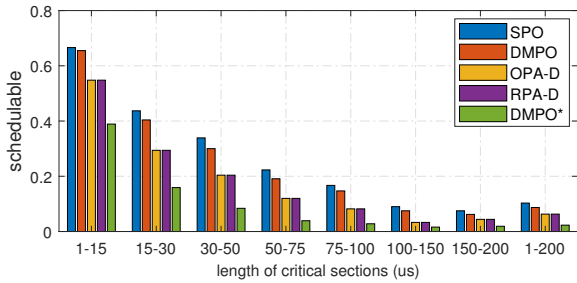


Fig. 6: Schedulability of MSRP systems for  $M = 16$ ,  $n = 48$ ,  $\kappa = 0.45$ ,  $A = 30$  and  $M$  shared resources

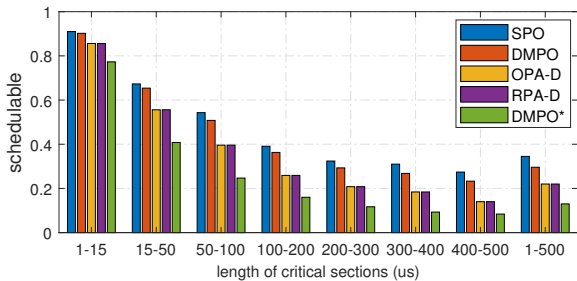


Fig. 7: Schedulability of MrsP systems for  $M = 16$ ,  $n = 48$ ,  $\kappa = 0.4$ ,  $A = 10$  and  $M$  shared resources

In addition, SPO outperforms DMPO in most cases, and their performance differential is amplified when blocking becomes the bottleneck factor of system schedulability (see  $A \geq 20$  in both figures). The above observations indicate that SPO effectively reduces the pessimism incurred when adopting the holistic tests.

Similar observations towards DMPO\*, OPA and RPA are also obtained in Figures 6 and 7, which vary the length of critical sections (i.e.,  $L$ ) under MSRP and MrsP, respectively. One interesting observation is that under this experiment, SPO demonstrates different performances between these protocols. Under MSRP (see Figure 6), the performance of SPO is similar to that of DMPO with  $L \leq 100\mu s$  and can hardly schedule any system. However, with MrsP applied, SPO outperforms other priority assignments and can lead to strong system schedulability with  $L > 100\mu s$ . Such an observation is due to the non-preemptive nature of MSRP, which is not favourable to long critical sections [34] because of the significant arrival blocking it introduces. Thus, for long critical sections, MSRP can hardly schedule any system regardless of the priority assignment applied.

As for MrsP, which adopts a priority ceiling facility and is more favourable to long critical sections [34], SPO demonstrates equal or better performances in most cases. Furthermore, for the last group of results in Figures 6 and 7, mixed length of critical sections is applied (with a length varying from 1 to 200  $\mu s$ ) to provide a realistic scenario where both short and long resources exist in the system. From these results, SPO again outperforms the existing priority assignments, in terms of the resulting system schedulability, when either MSRP or MrsP is applied.

TABLE IV: Percentage of schedulable MSRP systems for  $M = 16$ ,  $n = 64$ ,  $\kappa = 0.4$ ,  $L = [1, 15]\mu s$  and  $M$  resources

$A$	SPO & !DMPO	!SPO & DMPO	SPO & !OPA-D	!SPO & OPA-D	DMPO & !OPA-D	!DMPO & OPA-D
20	2.92	0.68	14.22	0	13.09	1.11
25	3.37	0.45	15.34	0	13.53	1.11
30	3.37	0.81	15.25	0.0	13.73	1.04
35	3.89	0.73	15.01	0.01	12.97	1.13
40	3.99	0.75	14.69	0.01	12.62	1.18
45	3.76	0.8	14.2	0.0	12.21	0.97
50	3.98	0.75	14.08	0.0	12.08	1.23

TABLE V: Percentage of schedulable MrsP systems for  $M = 16$ ,  $n = 48$ ,  $\kappa = 0.4$ ,  $A = 10$  and  $M$  resources

$L$ in $\mu s$	SPO & !DMPO	!SPO & DMPO	SPO & !OPA-D	!SPO & OPA-D	DMPO & !OPA-D	!DMPO & OPA-D
[15, 50]	2.17	0.47	12.07	0.00	11.38	1.00
[50, 100]	3.33	0.51	14.14	0.00	12.63	1.25
[100, 200]	3.76	0.63	13.51	0.00	11.96	1.17
[200, 300]	3.90	0.44	13.12	0.01	10.95	1.13
[300, 400]	4.20	0.43	12.96	0.00	10.29	1.17
[400, 500]	4.16	0.38	12.33	0.01	9.80	1.37
[1, 500]	4.52	0.39	13.04	0.00	10.65	1.15

The above experiments demonstrate the overall performance of the examined priority assignments. Tables IV and V show the percentage of systems where the priority assignment  $A$  can schedule but algorithm  $B$  cannot in 10,000 systems, for MSRP and MrsP respectively.  $A \& !B$  indicates the systems that are schedulable under  $A$  (e.g., SPO) but are unfeasible with  $B$  (e.g., !DMPO). The priority ordering algorithms that are examined in this experiment include the DMPO, RPA-D and SPO algorithms. The outputs of RPA-D is identical with that of OPA-D, and thus, is omitted.

Similar to the results obtained in the figures, both DMPO and SPO algorithms perform better than that of the OPA-D and RPA-D algorithms. In addition, SPO can schedule many more systems than the other algorithms; up to 15% of the input systems are schedulable under SPO but are unfeasible with OPA-D and RPA-D. However, as given in this table, no priority algorithm dominates others. For instance, DMPO is able to schedule 81 and 63 systems that SPO cannot with  $A = 30$  and  $L = [100\mu s, 200\mu s]$  under MSRP and MrsP respectively. Therefore, we conclude that the SPO algorithm yields better performance than others in general with holistic schedulability tests for either MSRP or MrsP. However, none of these priority assignment algorithms is optimal with the system model, resource sharing protocols and tighter holistic tests considered in this paper.

These observations imply that an appropriate approach to use in practice is a two-stage one: first apply the simpler DMPO, if this fails to deliver a schedulable system apply SPO.

## IX. CONCLUSION

In this paper, we investigate the optimality and applicability of DMPO, OPA and RPA in multiprocessor systems with shared resources managed by either MSRP or MrsP. We prove that DMPO remains optimal under the traditional schedulability tests of these protocols, but its optimality is

undermined due to the response time dependencies in the tighter holistic schedulability tests. In addition, a discussion is provided explaining the issues involved in applying the OPA-like search-based algorithm to the holistic tests, and the compromises required for its adoption. Subsequently, a search-based priority assignment method (SPO) that minimises the pessimism from the adoption on the holistic tests is developed, with polynomial time complexity. Finally, the impact of priority assignment on schedulability of multiprocessor systems with shared resources is investigated. We demonstrate that SPO has a better performance in general than other examined priority assignments, however, there exists no optimal priority ordering algorithm for the holistic schedulability tests for either MSRP or MrsP.

Further research towards the optimality and applicability of priority assignment may include (i) generalisation on resource accessing rules (e.g., non-preemptive, priority ceiling, and base priority) and accessing orders (e.g., FIFO and priority-ordered) instead of treating certain specific protocols; (ii) extension to support fine-grained nested locking and consideration of runtime overhead (e.g., preemptions, migrations and cache-related costs). In addition, the impact of priority assignment on the schedulability of more complex system models (e.g., arbitrary deadlines) is not known and requires investigation.

#### REFERENCES

- [1] N. C. Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. University of York, Department of Computer Science, 1991.
- [2] N. C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.
- [3] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.
- [4] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.
- [5] K. Bletsas and N. Audsley. Optimal priority assignment in the presence of blocking. *Information processing letters*, 99(3):83–86, 2006.
- [6] B. B. Brandenburg. *Scheduling and locking in multiprocessor real-time operating systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2011. <https://cs.unc.edu/~anderson/diss/bbbdiss.pdf>.
- [7] B. B. Brandenburg. Multiprocessor real-time locking protocols: A systematic review. *arXiv preprint arXiv:1909.09600*, 2019.
- [8] A. Burns, M. Gutierrez, M. A. Rivas, and M. G. Harbour. A deadline-floor inheritance protocol for edf scheduled embedded real-time systems with resource sharing. *IEEE Transactions on Computers*, 64(5):1241–1253, 2014.
- [9] A. Burns and A. J. Wellings. A schedulability compatible multiprocessor resource sharing protocol—mrsp. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 282–291. IEEE, 2013.
- [10] W. Chang, S. Chakraborty, et al. Resource-aware automotive control systems design: A cyber-physical systems approach. *Foundations and Trends® in Electronic Design Automation*, 10(4):249–369, 2016.
- [11] W. Chang, R. Wei, S. Zhao, A. J. Wellings, J. Woodcock, and A. Burns. Development automation of real-time java: Model-driven transformation and synthesis. *ACM Transactions in Embedded Computing Systems*, 2020.
- [12] W. Chang, S. Zhao, R. Wei, A. Wellings, and A. Burns. From java to real-time java: a model-driven methodology with automated toolchain. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 123–134, 2019.
- [13] Y. Chu and A. Burns. Flexible hard real-time scheduling for deliberative AI systems. *Real-Time Systems*, 40(3):241–263, 2008.
- [14] R. I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 3–14. IEEE, 2007.
- [15] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *Acm Computing Surveys*, 43(4):1–44, 2011.
- [16] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *Journal of systems architecture*, 65:64–82, 2016.
- [17] S. N. Dinh, J. Li, K. Agrawal, C. Gill, and C. Lu. Blocking analysis for spin locks in real-time parallel tasks. *IEEE Transactions on Parallel and Distributed Systems*, 2017.
- [18] S. Fürst, J. Mössinger, S. Bunzel, T. Weber, F. Kirschke-Biller, P. Heitkämper, G. Kinkelin, K. Nishikawa, and K. Lange. AUTOSAR—a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, volume 62, 2009.
- [19] P. Gai, G. Lipari, and M. Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 73–83. IEEE, 2001.
- [20] P. Gai, G. Lipari, and M. Di Natale. Stack size minimization for embedded real-time systems-on-a-chip. *Design Automation for Embedded Systems*, 7(1-2):53–87, 2002.
- [21] J. Garrido, S. Zhao, A. Burns, and A. Wellings. Supporting nested resources in MrsP. In *Ada-Europe International Conference on Reliable Software Technologies*, pages 73–86. Springer, 2017.
- [22] L. George, N. Rivierre, and M. Spuri. *Preemptive and non-preemptive real-time uniprocessor scheduling*. PhD thesis, Inria, 1996. <https://hal.inria.fr/inria-00073732>.
- [23] Y. Jiang, H. Song, R. Wang, M. Gu, J. Sun, and L. Sha. Data-centered runtime verification of wireless medical cyber-physical system. *IEEE transactions on industrial informatics*, 13(4):1900–1909, 2016.
- [24] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209. IEEE, 1990.
- [25] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance evaluation*, 2(4):237–250, 1982.
- [26] A. Racu and L. S. Indrusiak. Using genetic algorithms to map hard real-time on noc-based systems. In *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on*, pages 1–8. IEEE, 2012.
- [27] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Real-Time Systems Symposium, 1988., Proceedings.,* pages 259–269. IEEE, 1988.
- [28] L. Sha, R. Rajkumar, and J. P. Lehoczky. *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*. IEEE Computer Society, 1990.
- [29] J. Shi, K.-H. Chen, S. Zhao, W.-H. Huang, J.-J. Chen, and A. Wellings. Implementation and evaluation of multiprocessor resource synchronization protocol (mrsp) on litmusrt. In *13th Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, 2017.
- [30] K. W. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.
- [31] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS, 2007*.
- [32] A. Wieder and B. B. Brandenburg. On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 45–56. IEEE, 2013.
- [33] S. Zhao. *A FIFO Spin-based Resource Control Framework for Symmetric Multiprocessing*. PhD thesis, University of York, 2018.
- [34] S. Zhao, J. Garrido, A. Burns, and A. Wellings. New schedulability analysis for MrsP. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2017 IEEE 23rd International Conference on*, pages 1–10. IEEE, 2017.
- [35] S. Zhao, J. Garrido, R. Wei, A. Burns, A. Wellings, and A. Juan. A complete run-time overhead-aware schedulability analysis for MrsP under nested resources. *Journal of Systems and Software*, 159:110449, 2020.
- [36] S. Zhao and A. Wellings. Investigating the correctness and efficiency of MrsP in fully partitioned systems. In *The 10th York Doctoral Symposium on Computer Science and Electronics*. The University of York, 2017.
- [37] A. Zuhily and A. Burns. Optimal (D-J)-monotonic priority assignment. *Information Processing Letters*, 103(6):247–250, 2007.