Per Processor Spin-Based Protocols for Multiprocessor Real-Time Systems*

Sara Afshar[1], Moris Behnam[2], Reinder J. Bril[3], and Thomas Nolte[4]

1 **Mälardalen University, Västerås, Sweden**
`sara.afshar@mdh.se`
2 **Mälardalen University, Västerås, Sweden**
`moris.behnam@mdh.se`
3 **Mälardalen University, Västerås, Sweden**
`reinder.j.bril@mdh.se`
Technische Universiteit Eindhoven, Eindhoven, The Netherlands
`r.j.bril@tue.nl`
4 **Mälardalen University, Västerås, Sweden**
`thomas.nolte@mdh.se`

Abstract

This paper investigates preemptive spin-based global resource sharing protocols for resource-constrained real-time embedded multi-core systems based on partitioned fixed-priority preemptive scheduling. We present preemptive spin-based protocols that feature (*i*) an increased schedulability ratio of task sets and reduced response jitter of tasks compared to the classical non-preemptive spin-based protocol, (*ii*) similar memory requirements for the administration of waiting tasks as for the non-preemptive protocol whilst only causing (*iii*) a minimal increase of the minimal number of required stacks per core from one to at most two, and (*iv*) strong progress guarantees to tasks. We complement these protocols with a unified worst-case response time analysis that specializes to the classical analysis for the non-preemptive protocol. The paper includes a comparative evaluation of the preemptive protocols and the non-preemptive protocol based on synthetic data.

2012 ACM Subject Classification Computer systems organization~Real-time systems, Software and its engineering~Multiprocessing / multiprogramming / multitasking, Software and its engineering~Real-time schedulability

Keywords and Phrases multiprocessor, resource sharing, spin-lock protocols

Digital Object Identifier 10.4230/LITES-v004-i002-a003

Received 2017-02-06 **Accepted** 2017-10-10 **Published** 2018-01-08

1 Introduction

In this paper, we consider industrial real-time embedded multi-core systems. These systems typically control dedicated hardware and have strict timing requirements, e.g. the system shall not only provide responses to events within well-defined intervals, but also minimizes response-time fluctuations to guarantee specified quality levels of control. Due to their embedded nature, these systems are in many cases resource constrained for cost-efficiency reasons. For industrial real-time multi-core systems, partitioned fixed-priority preemptive scheduling is the defacto standard, i.e. tasks are statically allocated to cores, as exemplified by AUTOSAR [5], which is a standard for the automotive industry. For global resource sharing, i.e. sharing of, e.g., data or memory mapped

* This work is supported by the Swedish Foundation for Strategic Research via the research program PRESS, the Swedish Knowledge Foundation and ARTEMIS Joint Undertaking project EMC2 (grant agreement 621429).

© Sara Afshar, Moris Behnam, Reinder J. Bril, and Thomas Nolte;
licensed under Creative Commons Attribution 3.0 Germany (CC BY 3.0 DE)

I/O between tasks that are executing on different cores, AUTOSAR prescribes spin-based global resource-access protocols, which is the focus of this paper.

Two main requirements of these systems include cost-efficiency (i.e. resource constraints) and quality of control (i.e. jitter constraints). As a result of being resource constrained, resource usage, such as the amount of memory shall be restricted. Given our focus on global resource-access protocols for industrial embedded multi-core systems, we therefore aim at a high schedulability ratio of task sets on individual cores and low memory requirements. In the context of single-core systems, the stack resource policy (SRP) [6], which provides mutual exclusive access to shared resources, allows tasks to share a single stack by preventing interleaved executions of tasks, reducing memory requirements. The multiprocessor stack resource policy (MSRP) [18] generalizes SRP from a single core to a multi-core, whilst maintaining the attractive property of allowing tasks that are executing on the same core to share a single stack. MSRP essentially provides non-nested, non-preemptive spinning (i.e. busy-waiting) and non-preemptive resource access to global resources, and assumes first-in-first-out (FIFO) queueing of tasks that are waiting for those resources. Non-preemptive spinning has as an attractive side-effect that the length of individual global resource queues (and even the sum of the lengths of the global resource queues) is bounded by the number of cores.

On the other hand, embedded systems have jitter constraints. Response time fluctuations, i.e. response jitter, of control tasks may significantly degrade the control performance and, in the worst case, make control systems unstable. Response jitter shall therefore be limited for critical control tasks. The response jitter of a control task is bounded by the difference of the worst-case and best-case response time of that task [25, 11]. Assuming (a lower bound on) the best-case response time to be independent of a global resource sharing protocol, the bound on the response jitter decreases when the worst-case response time decreases. To minimize response jitter, control tasks are typically given the highest priorities in a system.

Unfortunately, non-preemptive spinning can impose a reduction in system schedulability, since a task that is spinning on a global resource blocks tasks with a higher priority on the same core that arrive during its busy-waiting time. *Preemptive spin-based* protocols reduce the blocking time of tasks with a priority higher than the priority at which the waiting task is spinning. Moreover, non-preemptive spinning may significantly increase the worst-case response time of control tasks due to remote blocking of tasks with a lower priority than the control tasks. *preemptive spin-based protocols* can reduce the blocking time of control tasks, thereby reducing their response jitter. In this paper we focus on a set of spin-based protocols that offer low memory usage and confine the length of the global resource queues to the number of cores the same as MSRP. Further in this section, we explore preemptive spin-based protocols, and conclude the section with the contributions.

1.1 Preemptive spin-based protocols

Before presenting the specific protocol considered in this paper, we first briefly describe this field. In particular, we identified three main characteristics of preemptive spin-based protocols, being the *spin-lock priority*, the *ordering* during waiting and the *impact of preemption on ordering*. We subsequently consider memory requirements and progress guarantees in more detail.

1.1.1 Main characteristics

We use *spin-lock priority* to refer to the priority at which a task is spinning while waiting for a global resource. Assuming a fixed spin-lock priority for a spin-based protocol, we identified that there are five possibilities for tasks to use spin-lock priorities. Tasks can use a fixed spin-lock

priority (*i*) per core, (*ii*) per task, (*iii*) per resource, (*iv*) per request and (*v*) hybrid, i.e. combination of any of the previous ones.

In the literature, preemptive spin-based global resource sharing protocols typically assume that a task is spinning at the priority of the task itself, i.e. the task's "own" or "original" priority [3, 13, 22, 27, 29] and hence it can be preempted due to the arrival of any higher priority task. We classify this protocol as of type (*ii*) and refer to it as *OP* (own priority).

The second and third characteristic concerns the ordering and the impact of preemption on ordering. Traditionally, there are two main techniques to determine which task is allowed to access a global resource when multiple tasks have pending requests, i.e. *ordered* (also termed *queued*) or *unordered*. In case of *ordered*, first-in-first-out (FIFO) queueing or queueing based on the priority of tasks are most common. While using priority-ordered resource queues may cause longer delays for a set of tasks (low priority tasks), it can decrease the waiting times of higher priority tasks. In fact, Wieder and Brandenburg [29] have shown that none of the queueing techniques dominates the other.

For *ordered*, three policies are typically considered in the literature for handling tasks that are residing in a global resource queue while being preempted during spinning, being *de-queuing* [13, 22, 3, 29], *skipping* [27] and the *classic* policy upon pre-emption, i.e. a task is neither de-queued nor skipped. The two former policies are typically used in conjunction with the spin-lock priority of type (*ii*), in particular with *OP*, allowing preempting tasks on the same core to access the global resource before the preempted task. De-queuing implies that a task that is preempted while spinning on a global resource is removed from the resource queue. It will again be put in the global resource queue when it is allowed to continue spinning. As a result, it may have to wait for, i.e. may be blocked by, additional remote tasks with later requests to the same global resource. Skipping implies that the task remains in the queue, but is not amenable for selection when the global resource becomes available. As a result, it may have to wait for an additional remote task with a later request to the same global resource that has been granted the resource while it was preempted. Under the classic policy, a task remains in the global resource queue when preempted and it is granted access to the global resource when it is at the head of the queue and the resource becomes available.

In this work, we consider spin-based protocols of type (*i*), where a fixed priority level is used for spinning for all tasks that are allocated to the same core. Moreover, to be consistent with MSRP, similar to MSRP [18], we assume FIFO-ordered queueing and the *classic* policy upon pre-emption.

1.1.2 Memory Requirements

As we described previously [1], the traditional spin-based and suspension-based global resource sharing protocols can conceptually be unified by viewing a suspension-based protocol as a spin-based protocol that uses the lowest priority level on a core, i.e. a priority lower than any "original" priority of tasks on that core. We refer to a suspension-based protocol as *LP* (lowest priority) and to the non-preemptive spin-based protocol as *HP* (highest priority). The *flexible spin-lock model* (FSLM) [1] allows the selection of an arbitrary priority level in the range of *LP* to *HP*, and addressed both specific instantiations of type (*i*), e.g. *LP* and *HP*, and type (*ii*), i.e. *OP*. Next to *LP* and *HP*, it also considered *CP* (ceiling priority), i.e. the highest priority of any task on a core using a global resource. In this paper, we focus on a particular subset of spin-based protocols from FSLM, i.e. those protocols that spin at a fixed priority per core in the range $[CP, HP]$ (with a slight misuse of notation, where we refer to both the protocol and its associated spin-lock priority by means of the same identifier, e.g. *CP*). An attractive point of this subset of protocols is that at most one task at any time on a core can either have a pending request on or access to a global

resource (see Lemma 12 in [1]). Using protocols from this range in combination with FIFO-ordered queueing will confine the length of any global resource queue to m, where m is the number of cores [1]. Using priority levels other than this range from the whole spectrum, such as of LP or OP, may result in a higher number of pending requests for a global resource on a core and thus longer queue sizes. Moreover, this subset of protocols can be used with a minimal increase of the number of required stacks per core. Later, in Section 4, we show that using spin-lock priorities from the range $[CP, HP)$ requires an increase from one, for MSRP (i.e. HP) to at most two stacks. This number is significantly lower than when using LP or OP which require in the worst-case n stacks per core, where n is the number of tasks allocated to that core. We therefore leave the study of the rest of the spectrum of spin-lock priorities as future work.

1.1.3 Progress Guarantees

As described above, the policies de-queueing and skipping are typically used for handling tasks that are residing in a global resource queue while being preempted during spinning. Both policies may, however, significantly increase the remote blocking time experienced by a preempted task. Under the de-queueing policy, every time a task is preempted during spinning, it has to wait for all the tasks that have been enqueued which the task was preempted. Therefore, the task may have to wait in the worst-case for an additional amount of at most $m-1$ remote tasks when using FIFO-ordered queues. Under the skipping policy, in the worst-case, every time a task is waiting for a global resource it may get preempted by a higher priority task just before it can get access to the resource and thus it will loose the access to the next queued task. Therefore, it may be delayed for an additional global resource access on a remote core. To feature the same strong progress guarantee as non-preemptive spin-based protocols, such as MSRP [18], we use the same policy as MSRP, where we keep the task in the resource queue upon preemption and immediately grant the task access to the global resource when it becomes available. In this way, a task has to wait for at most $m-1$ remote tasks when requesting a global resource, thereby preventing extra delays that can be imposed to a task under both de-queueing and skipping.

1.2 Main contributions and outline

This paper investigates preemptive spin-based global resource sharing protocols for resource-constrained real-time embedded systems based on partitioned fixed-priority preemptive scheduling. We focus on protocols with a fixed spin-lock priority per core, where the spin-lock priorities are taken from the range $[CP, HP]$. By design, the protocols feature similar memory requirements for the administration of waiting tasks as for the non-preemptive protocol and strong progress guarantees to tasks.

This paper has five main contributions. Firstly, we prove that these protocols feature a minimal increase of the minimal number of required stacks per core from one to at most two. Secondly, we introduce a special spin-based protocol from the introduced range, denoted by $\widehat{CP}$, where we (*i*) prove that it dominates the classical non-preemptive spin-based protocol and all spin-based protocols that use spin-lock priorities between $\widehat{CP}$ and HP, (*ii*) show by means of examples that CP and $\widehat{CP}$ are incomparable. This means that $\widehat{CP}$ performs always equal to or better than HP, unlike CP. Although $\widehat{CP}$ does not dominate CP, still we show that there are cases in which $\widehat{CP}$ performs better than CP. Thirdly, we provide a unified worst-case response time analysis for these protocols that specializes to the classical analysis for the non-preemptive protocol. Moreover, we show that our new analysis provides tighter blocking bounds for CP than the analysis in [1]. Fourthly, we show that there may exist an intermediate spin-lock priority within the range $[CP, \widehat{CP}]$ that can make a task set schedulable if CP and $\widehat{CP}$ cannot, which can be found via a simple linear search. Finally, we perform a comparative evaluation of HP, CP and

$\widehat{CP}$, based on the schedulability ratio of task sets and the improvement in response times of tasks.

The remainder of this paper is organized as follows. Sections 2 summarizes related work and Section 3 presents the system model. In Section 4, we prove that the minimal number of required stacks per core for the spin-based protocols under consideration increases to at most two. Section 5 proposes a new spin-based protocol $\widehat{CP}$. We subsequently present a generalized worst-case response time analysis for the protocols in Section 6. A theoretical comparison of *HP*, *CP*, and $\widehat{CP}$ is presented in Section 7. In Section 8, a comparative evaluation of *HP*, *CP*, and $\widehat{CP}$ is presented. We conclude the paper in Section 9.

2 Related Work

A non-exhaustive amount of work has been done on spin-based resource sharing protocols. In the following we briefly present the most related synchronization protocols used for multiprocessor systems.

Mellor-Crummey and Scott [23] investigate scalable spin-based protocols to minimize the network transactions that lead to contention, with a focus on non-preemptive spin-based protocols with FIFO-ordering. This work later inspired Craig and Johnson [13, 21] to use a priority-ordered variant. Whereas Craig [21] mainly focuses on non-preemptive spin-based protocols, Johnson[13] also investigates preemptive spin-based protocols with FIFO-ordering, using the de-queueing technique upon preemptions. There are extensions of these works[22, 3] that used a preemptive version with FIFO-ordering. Another work which has used a preemptive spin-based protocol is by Takada and Sakamura [27] which is based on a skipping policy. As described above, we neither use a de-queueing technique nor a skipping technique, because they expose tasks to longer remote blocking delays.

The Multiprocessor Stack Resource Policy (MSRP) was introduced by Gai et al. [18] for partitioned systems based on a non-preemptive spin-based protocol. MSRP is an extension of the Stack Resource Policy (SRP) [6] for multiprocessors and was the first work which carried out a formal blocking analysis for a spin-based protocol. Global resource waiting queues are FIFO-ordered under this protocol.

Devi et al. [16] introduced a non-preemptive spin-based protocol for global scheduling under the EDF policy. Faggioli et al. presented the Multiprocessor Bandwidth Inheritance (M-BWI) protocol [17], an extension of the Bandwidth Inheritance (BWI) protocol, for reservation-based scheduling and preemptive spinning. Under their protocol not only a spinning task can be preempted but also lock holder tasks may be preempted which leads to longer delays for releasing a resource compared to non-preemptive resource accesses that we use. M-BWI can be used in open systems where tasks can dynamically be added or removed. The resource queues used in M-BWI are FIFO-ordered.

The Flexible Multiprocessor Locking Protocol (FMLP) introduced by Block et al. [8] combines both spin-based and suspension-based protocols. Thus, it is categorized of type per-resource spin-based protocols in our classification described in Section 1.1.1. Tasks spin non-preemptive on so-called “short resources” and suspend on so-called “long resources”. FMLP uses FIFO-ordered global resource queues and has been introduced for both partitioned and global scheduling. The partitioned FMLP, was later extended for fixed-priority scheduling in [10].

A recent work by Wieder and Brandenburg [29] has investigated both preemptive and non-preemptive spin-based protocols under four different queue handling policies: FIFO and unordered spin-based protocols, and priority ordered spin-based protocols with FIFO-ordered and unordered tie breaking. They use a de-queueing technique in order to avoid transitive arrival blocking problem which occurs in combination with FIFO-ordered queues and preemptive spin-based

protocols, where tasks spin with their original priority (which we refer to it as *OP*). Transitive arrival blocking occurs when a task waiting in a global resource queue is preempted by a higher priority task on the core that require the same global resource, thus the higher priority task has to wait for that lower priority task. This problem does not occur for the spin-lock priority levels considered in this paper, since spinning is performed at a priority level equal to or higher than the priority of any task using a global resource. Wieder et al. [29] achieve tighter blocking bounds using mixed-integer linear program (ILP) techniques to bound the maximum cumulative blocking imposed to a task. We show later in Section 6.4 how ILP can be used for the set of spin-based protocols considered in this paper. In this paper, we investigate alternative spin-lock priorities that can improve schedulability, reduce memory requirements, and reduce response jitter.

The Multiprocessor resource sharing Protocol (MrsP) is a preemptive spin-based protocol that is proposed by Burns and Wellings [12]. MrsP is an extension of PCP [26] for multiprocessor fixed-priority partitioned scheduling where each global resource on each core is associated with a ceiling that is the highest priority among the tasks that request that resource on that core. Since ceiling of resources are used as the spin-lock priority for tasks on a core, unlike the spin-based protocols considered in this paper, MrsP uses spin-lock priorities of type (*iii*) mentioned in Section 1.1.1. Another key difference of this protocol from the protocols considered in this paper is that the critical sections are preemptive. Moreover, a helping method [28] has been used for MrsP where a spinning task donates its spinning time to a task that has locked the resource but cannot proceed since it has been preempted on its core. Under this method the preempted task migrates to a core where a task is spinning to lock the same resource and access its locked resource there. Global resource queues are FIFO-based under this protocol.

3 System Model

Our system consists of m identical processors executing a set of n sporadic tasks using fixed-priority partitioned scheduling. The set of tasks allocated to a processor P_k is denoted by $\mathcal{T}_{P_k}$. Each task τ_i is presented by $< C_i, D_i, T_i >$ and consists of an infinite sequence of jobs. C_i denotes the worst-case execution time of task τ_i. T_i denotes the minimum inter-arrival time of τ_i and D_i denotes the relative deadline of τ_i. We assume constrained deadlines tasks, i.e. $D_i \leq T_i$. The priority of the task τ_i is denoted by π_i where $\pi_i \geq 1$. U_i denotes the utilization of a task τ_i and is calculated as $U_i = C_i/T_i$. We assume that a task τ_i has a priority higher than task τ_j, i.e., $\pi_i > \pi_j$, if $i > j$, e.g. $\pi_2 > \pi_1$. We assume tasks with unique priorities on each processor.

Tasks in the system may use *local* or *global* resources. Local resources are those that are accessed only by tasks on the same processor, whereas global resources are accessed by tasks on different processors. The section of a task that uses global and local resource is called global and local critical section (*gcs,lcs*), respectively. The sets of local and global resources which are accessed by tasks on a processor P_k are denoted by $\mathcal{R}^{\mathrm{L}}_{P_k}$ and $\mathcal{R}^{\mathrm{G}}_{P_k}$, respectively. Similarly, we denote the set of local and global resources that are accessed by jobs of a task τ_i as $\mathcal{RS}^{\mathrm{L}}_i$ and $\mathcal{RS}^{\mathrm{G}}_i$, respectively. Further, $Cs_{i,q}$ denotes the worst-case execution time among all requests of any job of a task τ_i for a resource R_q. Moreover, $n^{\mathrm{G}}_{i,q}$ denotes the maximum number of possible requests by any job of a task τ_i for a specific global resource R_q. The set of tasks on a processor P_k requesting access to a specific resource R_q is denoted by $\mathcal{T}_{P_k,q}$. Nested resource access is not the focus of this paper. A complete set of notations can be found in Table 1 in Appendix A.

Based on the partitioned fixed-priority scheduling schema, we categorize the delay that can be introduced to any task due to resource sharing in this paper into two general blocking notions: *(i)* priority inversion blocking (pi-blocking) [24, 26] and *(ii)* remote blocking. Pi-blocking happens due to tasks assigned on the same core. When a lower priority job on the same core is scheduled while

a higher priority task is pending but not scheduled, the lower priority task causes a pi-blocking to the higher priority task. A job of a task is pending when it has arrived but not finished. Remote blocking, on the other hand, is the type of blocking that a job of a task experiences due to waiting for obtaining a global resource that is in use by a task on a remote core. The maximum duration of remote blocking experienced by a task is referred to as spin-lock time under a spin-based protocol (see Definitions 10 and 11).

In this paper, we denote the B_i as the total pi-blocking that is imposed to a task τ_i and we exclude the spin-lock time of τ_i from this term.

We assume negligible run-time overhead for the analysis step. We will leave investigation of such overheads for th next step towards implementation of the protocol.

3.1 General Definitions

Below, we present a set of definitions which will be used in the rest of this paper.

▶ **Definition 1.** The highest priority level on a processor P_k is denoted by $\pi^{\max}_{P_k}$ as follows, $\pi^{\max}_{P_k} = \max_{\tau_i \in \mathcal{T}_{P_k}} \{\pi_i\}$. This is the spin-lock priority used for the *HP* spin-based protocol (see Section 3.4.1).

▶ **Definition 2.** Ceiling-based resource-access protocols (such as SRP) assign a ceiling to any local resource $R_l \in \mathcal{R}^{\mathrm{L}}_{P_k}$, where $ceil_{P_k}(R_l) = \max\{\pi_i |\ \tau_i \in \mathcal{T}_{P_k} \wedge R_l \in \mathcal{RS}^{\mathrm{L}}_i\}$. [6]

▶ **Definition 3.** We denote the highest local ceiling of any regular[1] local resource on a processor P_k as $\pi^{\mathrm{L}}_{P_k}$, where $\pi^{\mathrm{L}}_{P_k} = \max\{\pi_i | \tau_i \in \mathcal{T}_{P_k} \wedge \mathcal{RS}^{\mathrm{L}}_i \neq \emptyset\}$, i.e., $\pi^{\mathrm{L}}_{P_k} \in [1, \pi^{\max}_{P_k}]$.

▶ **Definition 4.** We denote the highest local ceiling of any global resource on a processor P_k as π^{G}_{Pk}, where $\pi^{\mathrm{G}}_{P_k} = \max\{\pi_i | \tau_i \in \mathcal{T}_{P_k} \wedge \mathcal{RS}^{\mathrm{G}}_i \neq \emptyset\}$, i.e., $\pi^{\mathrm{G}}_{P_k} \in [1, \pi^{\max}_{P_k}]$. This is the spin-lock priority used for the *CP* spin-based protocol (see Section 3.4.2).

▶ **Definition 5.** We denote the highest local ceiling of any resource on a processor (either local or global) P_k as $\pi^{\mathrm{LG}}_{P_k}$, where $\pi^{\mathrm{LG}}_{P_k} = \max(\pi^{\mathrm{L}}_{P_k}, \pi^{\mathrm{G}}_{P_k})$, i.e., $\pi^{\mathrm{LG}}_{P_k} \in [1, \pi^{\max}_{P_k}]$. This is the spin-lock priority used for the $\widehat{CP}$ spin-based protocol (see Section 5).

▶ **Definition 6.** When a task spins on a processor to acquire a global resource, its priority might change during spinning depending on the spin-based protocol that is used. The spin-lock priority of a spin-based protocol σ is denoted by $\pi^{\mathrm{spin}_\sigma}_{P_k}$ which denotes an arbitrary spin-lock priority level that is used for every task when it spins on a processor P_k. We simply use $\pi^{\mathrm{spin}}_{P_k}$ if we do not refer to a specific spin-based protocol. In this paper, we consider $\pi^{\mathrm{spin}}_{P_k} \in [\pi^{\mathrm{G}}_{P_k}, \pi^{\max}_{P_k}]$.

▶ **Definition 7.** We denote the spin-lock priority used by a task τ_i as π^{spin}_i. According to our system model, $\pi^{\mathrm{spin}}_i = \pi^{\mathrm{spin}}_{P_k}$ where $\{\forall \tau_i \in \mathcal{T}_{P_k} | \mathcal{RS}^{\mathrm{G}}_i \neq \varnothing, k = 1, ..., m\}$

▶ **Definition 8.** We refer to pi-blocking that is imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ by lower priority tasks on the same core that request local resources as *local blocking due to local resources* (*LBL*) and global resources as *local blocking due to global resources* (*LBG*).

▶ **Definition 9.** The LBG delay that is imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ is divided into two different sections: *(a) spin-delay* blocking of an LBG which is due to spinning of a lower priority task that can preempt τ_i and *(b)* global *resource access* blocking of an LBG which is due to non-preemptive access of a lower priority task to a global resource. Note that a LBG delay does not necessarily need to contain the spin-delay part, e.g., when the lower priority task access the resource immediately.

[1] We define a special (local) spin resource $R^{\mathrm{spin}}_{P_k}$ in Rule 21.

▶ **Definition 10.** The maximum time that any task on a processor P_k has to spin to acquire a global resource $R_q \in \mathcal{R}^{\mathrm{G}}_{P_k}$ is referred to as spin-lock time to acquire R_q and is denoted by $spin_{P_k,q}$, which is the maximum imposed remote blocking to acquire R_q.

▶ **Definition 11.** The maximum time that a task τ_i has to spin to acquire all its global resources is referred to as spin-lock time of task τ_i and is denoted by $spin_i$, which is the maximum imposed remote blocking to τ_i for acquiring all its global resources .

Under spin-based protocols usually the execution times of tasks are inflated by the spin-lock time [18, 20, 1] as presented by the following definition.

▶ **Definition 12.** The inflated execution time of a task τ_i is denoted by $\acute{C}_i$ and is calculated as $\acute{C}_i = C_i + spin_i$.

▶ **Note 13.** *By definition (see Definition 4),* $\forall \pi_i > \pi^{\mathrm{G}}_{P_k} |\ \tau_i \in \mathcal{T}_{P_k} \Longrightarrow \acute{C}_i = C_i$ *since* $spin_i = 0$.

▶ **Definition 14.** Davis et al. [15] defined: algorithm A dominates algorithm B, if all of the task sets that are schedulable according to algorithm B are also schedulable according to algorithm A, and task sets exist that are schedulable according to A, but not according to B. Moreover, algorithms A and B are incomparable, if there exist task sets that are schedulable according to algorithm A, but not according to algorithm B and vice versa. Since resource sharing protocols are part of scheduling algorithms, these definitions also apply for spin-based protocols in this paper. Based on this conclusion, if a task set is schedulable by both algorithms and the worst-case response times of tasks under spin-based protocol 1 is always smaller than or equal to under spin-based protocol 2 and there is at least one task that has a strictly smaller worst-case response time under protocol 1 compared to protocol 2, then by reducing the deadline of this task we create a new task set for which it is schedulable under protocol 1 but not 2 anymore which infers the dominance of protocol 1 over 2. Inferred similarly, spin-based protocols 1 and 2 are incomparable if a task set is schedulable under both protocols and there is a task that has a strictly smaller worst-case response time under one compared to the other and vice versa.

3.2 Resource Sharing Rules

This section presents the resource sharing rules based on FSLM [1] for any spin-based protocol with $\pi^{\mathrm{spin}}_{P_k} \in [\pi^{\mathrm{G}}_{P_k}, \pi^{\max}_{P_k}]$. The key idea is that a task τ_i waiting for a global resource, will busy wait, i.e. spin, whenever the resource is not available using a specific priority level in the aforementioned range. However, the priority level on which the task spins is fixed for a core; see Definition 6.

▶ **Rule 15.** *Local resources are handled by means of the SRP uniprocessor synchronization protocol [6].*

▶ **Rule 16.** *For each global resource, a FIFO-ordered queue is used to enqueue the tasks waiting for the related resource.*

The key idea behind using FIFO-ordered queues for global resources is to use a similar setup as the existing protocols (*HP* and *CP*), so that the comparison is feasible.

▶ **Rule 17.** *Whenever a task τ_i on a processor P_k requests a global resource that is in use by another processor, it places its request in the associated resource queue and spins. The task will spin with a priority level $\pi^{\mathrm{spin}}_{P_k}$ (Definition 6).*

▶ **Rule 18.** *When a task is granted access to its requested global resource on a processor P_k, its priority is boosted in an atomic operation to $\pi^{\max}_{P_k} + 1$, i.e., it access the resource immediately and executes non-preemptively on the core.*

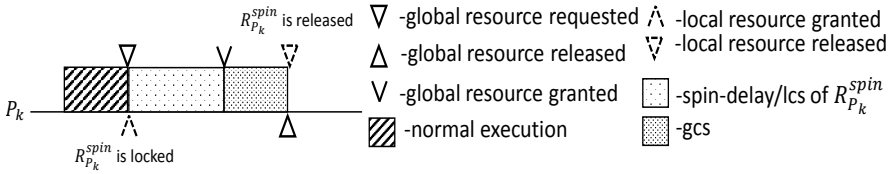

Figure 1 Spinning on P_k is viewed as locking a special local resource $R^{\text{spin}}_{P_k}$.

▶ **Rule 19.** *The priority of the task is changed to its original priority as soon as it finishes the global critical section where it becomes preemptable again.*

▶ **Rule 20.** *When the global resource becomes available (i.e. it is released), the task at the head of the global resource queue (if any) is granted the resource.*

The analysis of blocking bounds and all claims regarding the considered spin-based protocols in this paper are based on our system model and presented resource sharing rules.

3.3 View on spinning and global resource access

Under a spin-based protocol a task spins whenever it is blocked on a global resource. In classical spin-based protocols such as MSRP [18, 20] (which we refer to it as *HP*) as soon as a task is blocked on a global resource, its priority is increased to the highest on its assigned core. The task maintains this priority until it releases the resource. In this paper, we use a different approach which is increasing the priority of the task in two steps. In the first step, i.e., as soon as the task on a processor P_k request a global resource, its priority is increased to $\pi^{\text{spin}}_{P_k}$. In the second step, i.e., as soon as the task is granted access to the global resource, its priority is boosted such that it becomes non-preemptive (see Rule 18). The idea behind increasing the priority of the blocked task in two steps is to allow the high priority tasks on the core, which may even not use any (global) resource, to proceed when a lower priority task is waiting.

Conceptually, we can view spinning as accessing a "virtual" local resource (similar to a local *pseudo resource* [18]). Under a local resource sharing protocol when a task acquires a local resource its priority is raised to the ceiling of the resource, which is higher than or equal to the task's own priority. Since we only consider spin-lock priorities that are higher than or equal to the priority of any task using a global resource on the core based on our system model (Definition 6), we can treat spinning in the same way as acquiring a regular local resource by assigning the ceiling of such local resource equal to the priority during spinning. The benefit of such a view is that a local resource sharing protocol can take care of changing the priority of the task for spinning which removes the need for operating system to take such an action. Moreover, such a view simplifies validating the analysis. Having this in mind, we refer to such a virtual resource as *spin resource* and denote it for a processor P_k as $R^{\text{spin}}_{P_k}$. Rule 21 is the outcome of such a view.

▶ **Rule 21.** *For each processor P_k a special (local)* spin resource $R^{\text{spin}}_{P_k}$ *is dedicated. Every task τ_i on P_k that wants to request a global resource R_q, first locks $R^{\text{spin}}_{P_k}$ where, $ceil_{P_k}(R^{\text{spin}}_{P_k}) = \pi^{\text{spin}}_{P_k}$. The global resource access of R_q is nested within the local spin resource access of $R^{\text{spin}}_{P_k}$. The spin resource is released after the global resource is released.*

Figure 1 illustrates nesting of the global critical section within the local critical section of the special local resource $R^{\text{spin}}_{P_k}$. As can be seen, for a task the access time to the local spin resource consists of two parts: (*1*) the time that the task non-preemptively access a global resource and (*2*) the time that the task is spinning to acquire the global resource. Based on this and according to Rule 21 and the fact that the maximum time that a task may spin to acquire a global resource R_q is $spin_{P_k,q}$ (see Definition 10), hence we can reformulate Definition 9, as follows.

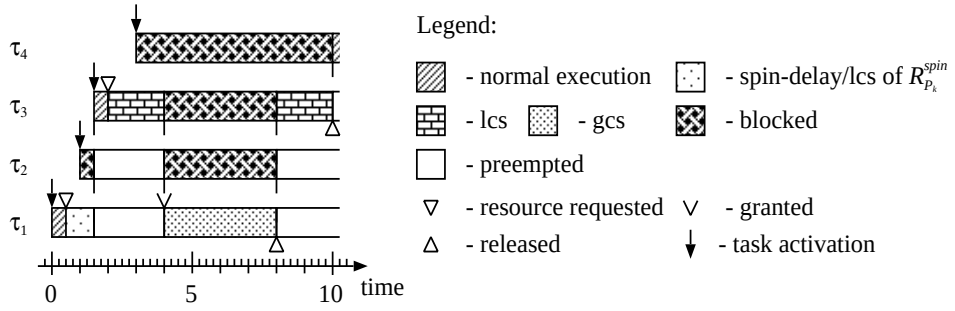

Figure 2 Task τ_4 experiences both a spin-delay and a global resource access blocking delay of an LBG.

▶ **Definition 22.** Any LBG that is imposed to a task on a processor P_k is due to non-preemptive access to a global resource of a job of a lower priority task that is nested within an access to the special spin resource $R^{\text{spin}}_{P_k}$. The maximum duration of such blocking is equal to $spin_{P_k,q} + \max_{\substack{\forall q,j:\tau_j\in\mathcal{T}_{P_k} \\ \wedge R_q\in\mathcal{RS}^{\text{G}}_j\wedge\pi_j<\pi_i}} Cs_{j,q}$.

In Figure 2 it can be seen that a task τ_4 experiences two types of blocking delay from lower priority tasks, due to both local resource access as well as global resource access. In the time interval $[3,4)\cup[8,10)$ τ_4 experiences LBL from τ_3 that has arrived earlier and has requested a local resource with a ceiling higher than or equal to τ_4's priority. In the time interval $[4,8)$ it experiences LBG from τ_1 that has arrived earlier and has requested a global resource. τ_1 has issued its request for a global resource at time 0.5, however has got blocked on the resource since the resource has been in use on a different processor. Thus when τ_1 is granted access to the global resource at time 4 it preempts τ_3 and execute its gcs non-preemptively (see Rule 18). By viewing spinning as access to a special local resource $R^{\text{spin}}_{P_k}$ on processor P_k, the time duration in which τ_1 is spinning can be viewed as an lcs duration which τ_1 access $R^{\text{spin}}_{P_k}$ with a ceiling equal to π_2.

3.4 Recap of Existing Analysis and Lemmas

In this section we briefly present the blocking analysis of the two existing spin-based protocols each of which uses a fixed spin-lock priority from the introduced spin-lock range in the system model, i.e., $[\pi^{\text{G}}_{Pk}, \pi^{\max}_{P_k}]$.

3.4.1 *HP* Spin-Based Protocol

Under *HP*, $\pi^{\text{spin}_{HP}}_{P_k} = \pi^{\max}_{P_k}$ (recall Definitions 1 and 6) which makes a task non-preemptive while spinning. This protocol has been introduced by Gai et al. [18]. Below we present the blocking delays that occur under this protocol.

LBL (Definition 8) imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ due to normal local resources is denoted as B^{L}_i and is upper bounded as follows:

$$B^{\text{L}}_i = \max_{\substack{\forall j,l:\pi_j<\pi_i\wedge\ \tau_i,\tau_j\in\tau_{P_k} \\ \wedge\ R_l\in\mathcal{RS}^{\text{L}}_j\wedge\ \pi_i\le ceil_{P_k}(R_l)}} \{Cs_{j,l}\}. \tag{1}$$

LBG (Definition 8) imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ is denoted as B^{G}_i and is upper bounded as follows:

$$B^{\text{G}}_i = \max_{\substack{\forall j,q:\pi_j<\pi_i\wedge\tau_i,\tau_j\in\mathcal{T}_{P_k} \\ \wedge R_q\in\mathcal{RS}^{\text{G}}_j}} \{Cs_{j,q} + spin_{P_k,q}\}. \tag{2}$$

The total pi-blocking imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ is denoted by B_i and is upper bounded as follows:

$$B_i = \max\{B_i^{\mathrm{L}}, B_i^{\mathrm{G}}\}. \tag{3}$$

$spin_{P_k,q}$ (Definition 10) and $spin_i$ (Definition 11) are upper bounded as follows [20]:

$$spin_{P_k,q} = \sum_{\forall P_r \neq P_k} \max_{\forall \tau_j \in \mathcal{T}_{P_r,q}} Cs_{j,q}. \tag{4}$$

$$spin_i = \sum_{\forall q: R_q \in \mathcal{RS}_i^{\mathrm{G}} \wedge \tau_i \in \mathcal{T}_{P_k}} n_{i,q}^{\mathrm{G}} \times spin_{P_k,q}. \tag{5}$$

For simplicity, under *HP* the execution times are inflated with the spin-lock time of the task. The inflated execution time of a task τ_i, $\acute{C}_i$ is calculated according to Definition 12 where $spin_i$ incorporated in it is calculated by (5).

3.4.2 *CP* Spin-Based Protocol

Under *CP* $\pi_{P_k}^{\mathrm{spin}_{CP}} = \pi_{P_k}^{\mathrm{G}}$ (recall Definitions 4 and 6) which makes a task to be non-preemptive while spinning for any task that uses a global resource on the core. This protocol has been studied previously [1]. Below we present the blocking delays that occur under this protocol.

LBL (Definition 8) imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ for *CP* is upper bounded similar as in *HP*, according to (1).

LBG (Definition 8) imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ is upper bounded as follows:

$$B_i^{\mathrm{G}} = \max_{\substack{\forall j,q: \pi_j < \pi_i \wedge \tau_i, \tau_j \in \mathcal{T}_{P_k} \\ \wedge R_q \in \mathcal{RS}_j^{\mathrm{G}}}} \{Cs_{j,q} + spin_{P_k,q} | (\pi_i \leq \pi_{P_k}^{\mathrm{G}})\}. \tag{6}$$

The total pi-blocking imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ B_i is upper bounded as follows:

$$B_i = \begin{cases} B_i^{\mathrm{L}} + B_i^{\mathrm{G}} & \text{if } \pi_i > \pi_{P_k}^{\mathrm{G}} + 1 \\ \max\{B_i^{\mathrm{L}}, B_i^{\mathrm{G}}\} & \text{if } \pi_i \leq \pi_{P_k}^{\mathrm{G}} + 1 \end{cases}. \tag{7}$$

▶ **Note 23.** *$spin_{P_k,q}$ and $spin_i$ are calculated as in (4) and (5), respectively and the inflated execution time of a task τ_i, i.e., $\acute{C}_i$ is calculated according to Definition 12.*

▶ **Note 24.** *If $\pi_{P_k}^{\mathrm{G}} = \pi_{P_k}^{\max}$ then CP is equal to HP, hence (7) and (6) specialize from (2) and (3), respectively.*

3.4.3 Recap of Useful Lemmas

Here we repeat some lemmas presented previously [1] that will be used in this paper.

▶ **Lemma 25.** *A job of a task $\tau_i \in \mathcal{T}_{P_k}$ experiences at most one LBL (recall Definition 8) from any lower priority task when SRP is used for local resource sharing (Property of SRP [6]).*

▶ **Lemma 26.** *A job of a lower priority task τ_j cannot issue any resource request after any job of a higher priority task τ_i on the same core arrives, where $\pi_i \leq \pi_i^{\mathrm{spin}}$ (Lemma 2 in [1]).*

▶ **Lemma 27.** *A job of a lower priority task τ_j can cause pi-blocking to any job of a higher priority task τ_i at most once, where $\pi_i \leq \pi_i^{\mathrm{spin}}$ (Lemma 3 in [1]).*

▶ **Note 28.** *According to Definitions 6 and 7 based on our system model,* $\{\forall\tau_i \in \mathcal{T}_{P_k} | \mathcal{RS}_i^{\mathrm{G}} \neq \varnothing \Longrightarrow \pi_i^{\mathrm{spin}} = \pi_{P_k}^{\mathrm{spin}} \geq \pi_{P_k}^{\mathrm{G}}\}$*. Since by definition, i.e., Definition 4,* $\pi_{P_k}^{\mathrm{G}}$ *is the highest priority of any task requesting a global resource therefore, for any such task* τ_i *that uses a global resource* $\pi_i \leq \pi_{P_k}^{\mathrm{G}}$*. Thus,* $\pi_i \leq \pi_i^{\mathrm{spin}}$*. As a result Lemmas 26 and 27 are valid based on our system model as well.*

4 Number of Stacks

In this section we present the maximum number of required stacks for tasks that use a spin-lock priority $\pi_{P_k}^{\mathrm{spin}}$ where $\pi_{P_k}^{\mathrm{spin}} \in [\pi_{P_k}^{\mathrm{G}}, \pi_{P_k}^{\max}]$. It has been shown by Gai et al. [18] that *HP* spin-based protocol (MSRP) allows using a single stack for all tasks on a core. Here, we show that a spin-based protocol that uses any spin-lock priority in the range $\pi_{P_k}^{\mathrm{spin}} \in [\pi_{Pk}^{\mathrm{G}}, \pi_{P_k}^{\max})$ allows using of only two stacks for scheduling all tasks on a core. For this purpose, we show that (*i*) all tasks with a priority at most $\pi_{P_k}^{\mathrm{spin}}$ can share one stack, and (*ii*) all tasks with a priority higher than $\pi_{P_k}^{\mathrm{spin}}$ and smaller than or equal to $\pi_{P_k}^{\max}$ can share another stack. It has been shown [19] that if task executions are non-interleaved then it is possible to use a single stack for scheduling all tasks. Therefore, it is enough to show that executions of tasks in group (*i*) and similarly in group (*ii*) are non-interleaved. Non-interleaved execution means that if a job of a task τ_i preempts a job of a task τ_j, the job of τ_j cannot execute before the job of τ_i is finished. Thus, we present such a property by Lemmas 30 and 31 for the execution of tasks in the two above mentioned groups. First we recapitulate the outcome of using SRP in Lemma 29 which will be used in those two lemmas.

▶ **Lemma 29.** *The local resource sharing protocol SRP ensures that once a job is started, it cannot be blocked due to a local resource until completion; it can only be preempted by higher priority jobs. [6]*

▶ **Lemma 30.** *For any task* $\tau_i \in \mathcal{T}_{P_k}$ *where* $\pi_i \leq \pi_{P_k}^{\mathrm{spin}}$ *a job of a lower priority task which is preempted by a job of* τ_i *cannot execute until the job of* τ_i *is finished.*

Proof. Proof by contradiction. Let us assume a job of τ_i preempts a job of a lower priority task τ_j with priority π_j at time t_1, and before τ_i is finished at time t_3 τ_j preempts τ_i at time t_2. This implies that at time t_2, the priority of τ_j must have been raised to or above τ_i's priority. Three situations may happen for τ_j so that its priority is raised: (*1*) τ_j accesses a local resource R_l where $ceil_{P_k}(R_l) > \pi_i$, (*2*) τ_j is granted access to the special local spin resource $\pi_{P_k}^{\mathrm{spin}}$ due to a request for a global resource R_g (see Rule 21) where $\pi_{P_k}^{\mathrm{spin}} > \pi_i$ and (*3*) τ_j accesses a global resource R_g and becomes non-preemptive (see Rule 18). In Cases (*1*) and (*2*), τ_i is blocked by τ_j at time t_2 which cannot happen according to Lemma 29, having in mind that SRP treats the local spin resource similar to any other normal local resource. In Case (*3*) the priority of τ_j has been raised to $\pi_{P_k}^{\max} + 1$ at time t_2. According to Rule 21 τ_j must first have locked the spin resource, let us assume at time t_0. τ_j cannot issue any request after arrival of τ_i at time t_a (Lemma 2 in [1]; see Lemma 26), thus $t_0 < t_a$. Further, it is obvious that $t_1 \geq t_a$ therefore it is inferred that $t_0 < t_1$. This implies that the priority of τ_j is raised to $\pi_{P_k}^{\mathrm{spin}}$ when it locks $R_{P_k}^{\mathrm{spin}}$ at time t_0 until t_2 where it gets access to R_g. Therefore, τ_i could only preempt τ_j at time t_1 if its priority has been raised higher than $\pi_{P_k}^{\mathrm{spin}}$. The only situation that τ_i's priority is raised higher than $\pi_{P_k}^{\mathrm{spin}}$ is when it is granted access to a global resource (see Rule 18). However, according to Rule 21, in order to access a global resource τ_i must have locked the local spin resource $R_{P_k}^{\mathrm{spin}}$ at time t_1 as well. This is not possible since τ_j is already holding $R_{P_k}^{\mathrm{spin}}$ at time t_1. We therefore conclude that τ_j could not have preempted τ_i at time t_2 which means that it cannot execute until τ_i is finished. ◀

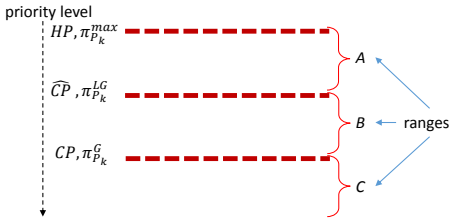

Figure 3 Priority ranges when $\pi_{P_k}^{\mathrm{G}} < \pi_{P_k}^{\mathrm{L}}$

▶ **Lemma 31.** *For any two tasks $\tau_i, \tau_j \in \mathcal{T}_{P_k}$ where $\pi_{P_k}^{\mathrm{spin}} < \pi_i, \pi_j \leq \pi_{P_k}^{\mathrm{max}}$ a job of τ_j which is preempted by a job of τ_i cannot execute until the job of τ_i is finished.*

Proof. According to Definition 6 $\pi_{P_k}^{\mathrm{spin}} \geq \pi_{P_k}^{\mathrm{G}}$ thus, by definition (see Definition 4), any task with a priority higher than $\pi_{P_k}^{\mathrm{spin}}$ does not use any global resource. therefore, τ_i and τ_j may only share local resources. Hence, this lemma can be inferred based on Lemma 29. ◀

▶ **Theorem 32.** *It is enough to use only two stacks for scheduling tasks on a processor P_k when selecting spin-based protocols with a spin-lock priority in the range $[\pi_{Pk}^{\mathrm{G}}, \pi_{P_k}^{\mathrm{max}})$.*

Proof. It is shown by Lemmas 30 and 31 that tasks of group (*i*) as well as tasks of group (*ii*) have non-interleaved executions, respectively. This implies that tasks of each group can use a single stack. However, tasks of group (*i*) cannot share the same stack with group (*ii*). This is due to the fact that if a task τ_j from group (*i*) is blocked on a global resource, i.e., it has locked the spin resource with ceiling $\pi_{P_k}^{\mathrm{spin}}$ but has not yet been granted access to the global resource, it can be preempted by a task τ_i from group (*ii*). If τ_j is granted access to the global resource before τ_i is finished, it raises its priority to $\pi_{P_k}^{\mathrm{max}} + 1$ and therefore will preempt τ_i. This means that the execution of τ_i and τ_j will not be interleaved and cannot share the same stack. We therefore conclude that all tasks on P_k can be scheduled using two stacks. ◀

5 A Special Spin-Based Protocol $\widehat{CP}$

In this section we introduce a special spin-based protocol from the range [*CP*, *HP*] which we denote by $\widehat{CP}$. This protocol uses the lowest priority for spinning such that no other task at the same core using either a local or global resource can preempt during spinning, i.e., $\pi_{P_k}^{\mathrm{spin}_{\widehat{CP}}} = \pi_{P_k}^{\mathrm{LG}}$ (recall Definitions 5 and 6). We show in Section 5.1 that $\widehat{CP}$ dominates *HP* and all spin-based protocols that use spin-lock priorities in between. In Section 5.2, we show by means of an example that *CP* and $\widehat{CP}$ are incomparable.

When $\pi_{P_k}^{\mathrm{L}} \leq \pi_{P_k}^{\mathrm{G}}$, then according to Definition 5 $\pi_{P_k}^{\mathrm{LG}} = \pi_{P_k}^{\mathrm{G}}$, having in mind that $\pi_{P_k}^{\mathrm{G}}$ is the spin-lock priority of *CP* [1]. Therefore, $\widehat{CP}$ only differs from *CP* when $\pi_{P_k}^{\mathrm{L}} > \pi_{P_k}^{\mathrm{G}}$. By this observation we introduce three priority ranges based on these two key priority levels, as illustrated in Figure 3 which later are elaborated in Sections 7 and 8. We specify these ranges as follows: (*A*) $\forall \pi \mid \pi > \pi_{P_k}^{\mathrm{LG}}$, (*B*) $\forall \pi \mid \pi_{P_k}^{\mathrm{G}} < \pi \leq \pi_{P_k}^{\mathrm{LG}}$ and (*C*) $\forall \pi \mid \pi \leq \pi_{P_k}^{\mathrm{G}}$ where π denotes an arbitrary priority level on a processor P_k.

5.1 Dominance of $\widehat{CP}$ over *HP* and In-Between Spin-Based Protocols

We show by means of Lemma 33 that, following our proposed spin-lock model, $\widehat{CP}$ dominates all spin-based protocols that use a spin-lock priority higher than π_{Pk}^{LG}.

▶ **Lemma 33.** *$\widehat{CP}$ dominates any spin-based protocol that uses a spin-priority level higher than what is used by $\widehat{CP}$, including HP.*

Proof. To prove this we assume an arbitrary spin-based protocol σ with a spin-lock priority $\pi_{P_k}^{\text{spin}_\sigma}$ higher than that of $\widehat{CP}$, i.e., $\pi_{P_k}^{\text{spin}_\sigma} > \pi_{P_k}^{\text{LG}}$ on P_k. Let us assume that a lower priority task τ_j incurs an LBG to a task τ_i. According to Definition 22, the LBG delay incurred to τ_i is divided into two parts: (a) delay due to a non-preemptive access to the global resource by τ_j and (b) delay due to the remaining access to the special local spin resource $R_{P_k}^{\text{spin}}$ with ceiling $\pi_{P_k}^{\text{spin}}$ (Definition 21). Since the access to a global resource is non-preemptive (Rule 18) hence the incurred LBG delay regarding case (a) is incurred to τ_i under any spin-based protocol, thus under both $\widehat{CP}$ and σ. However, the remaining access to the spin resource by τ_j will only block τ_i, i.e., τ_i will incur LBG delay regarding case (b) if and only if $\pi_i \leq \pi_{P_k}^{\text{spin}}$ where $\pi_{P_k}^{\text{spin}} = \pi_{P_k}^{\text{LG}}$ under $\widehat{CP}$ and $\pi_{P_k}^{\text{spin}} = \pi_{P_k}^{\text{spin}_\sigma}$ under σ spin-based protocol. Let us assume three different possible priority ranges for a task τ_i on P_k being: (i) $\pi_i \leq \pi_{P_k}^{\text{LG}}$, (ii) $\pi_{P_k}^{\text{LG}} < \pi_i \leq \pi_{P_k}^{\text{spin}_\sigma}$ and (iii) $\pi_i > \pi_{P_k}^{\text{spin}_\sigma}$. τ_i will experience the delay of type (b) using both spin-based protocols σ and $\widehat{CP}$ under the condition of case (i) since $\pi_i \leq \pi_{P_k}^{\text{LG}} < \pi_{P_k}^{\text{spin}_\sigma}$ and will not experience it using both spin-based protocols under the condition of case (iii) since $\pi_i > \pi_{P_k}^{\text{spin}_\sigma} > \pi_{P_k}^{\text{LG}}$. Thus, for a task under conditions (i) and (iii), there is no difference in using either of the spin-based protocols. Looking at the condition of case (ii), however, τ_i experiences the delay of type (b) using protocol σ but does not experience the delay using $\widehat{CP}$. This implies that the response time of τ_i is smaller when using $\widehat{CP}$ compared to when using σ. Hence, when the task set is schedulable under $\widehat{CP}$, we can make the task set unschedulable under σ by reducing the deadline of τ_i (see Definition 14). As a result, $\widehat{CP}$ dominates σ. Since σ can be any spin-based protocol where $\pi_{P_k}^{\text{LG}} < \pi_{P_k}^{\text{spin}_\sigma}$, it can be concluded that $\widehat{CP}$ dominates any spin-based protocol with a spin-lock priority higher than that of $\widehat{CP}$, i.e., also *HP* since $\pi_{P_k}^{\text{LG}} < \pi_{P_k}^{\max}$. This finishes the proof. ◀

From Lemma 33, we draw the following conclusion.

▶ **Corollary 34.** *If $\pi_{P_k}^{\text{G}} = \pi_{P_k}^{\text{LG}} < \pi_{P_k}^{\max}$, then CP dominates HP.*

5.2 $\widehat{CP}$ and CP incomparability

Next, we show by means of an example that CP and $\widehat{CP}$ are incomparable (Definition 14).

▶ **Example 35.** In the example depicted in Figure 4 which consists of two scenarios a task set is scheduled on two processors P_1 and P_2 where $\mathcal{T}_{P_1} = \tau_1, ..., \tau_6$ and $\mathcal{T}_{P_2} = \tau_7$. $T_1 = T_7 = 100$, $T_2 = 100.2$, $T_3 = T_4 = 101$ and $T_5 = T_6 = 106$, moreover, $D_1 = 9$, and the deadline for the rest of the tasks is 20. For each task τ_i, the given task specifications are specified in the format (C_i, $Cs_{i,l}$, $Cs_{i,g}$). For the following tasks these values are the same under both scenarios. τ_1: (4, 0, 3), τ_2: (1, 0, 1), τ_4: (3, 0, 0), τ_5: (1, 1, 0) and τ_6: (1, 0, 0). Note that value zero implies that the task does not use that specific resource. Under scenario *(1)*, τ_3: (2, 1, 0) and τ_7: (7, 0, 5). In scenario *(1)* τ_4 misses its deadline under $\widehat{CP}$. In scenario *(2)*, τ_3: (4, 4, 0) and τ_7: (4, 0, 1), and τ_4 misses its deadline under CP in this scenario.Given the tasks resource requesting specification under both scenarios $\pi_{P_k}^{\text{spin}_{CP}} = 2$ and $\pi_{P_k}^{\text{spin}_{\widehat{CP}}} = 5$. Since in each scenario τ_4 misses its deadline using either CP or $\widehat{CP}$, thus according to Definition 14 CP or $\widehat{CP}$ are incomparable.

6 Generalized Analysis

In this section we derive a general blocking analysis for selection of any arbitrary fixed spin-lock priority from FSLM where $\pi_{P_k}^{\text{spin}} \in [\pi_{Pk}^{\text{G}}, \pi_{P_k}^{\max}]$. To provide the maximum blocking delay to a task,

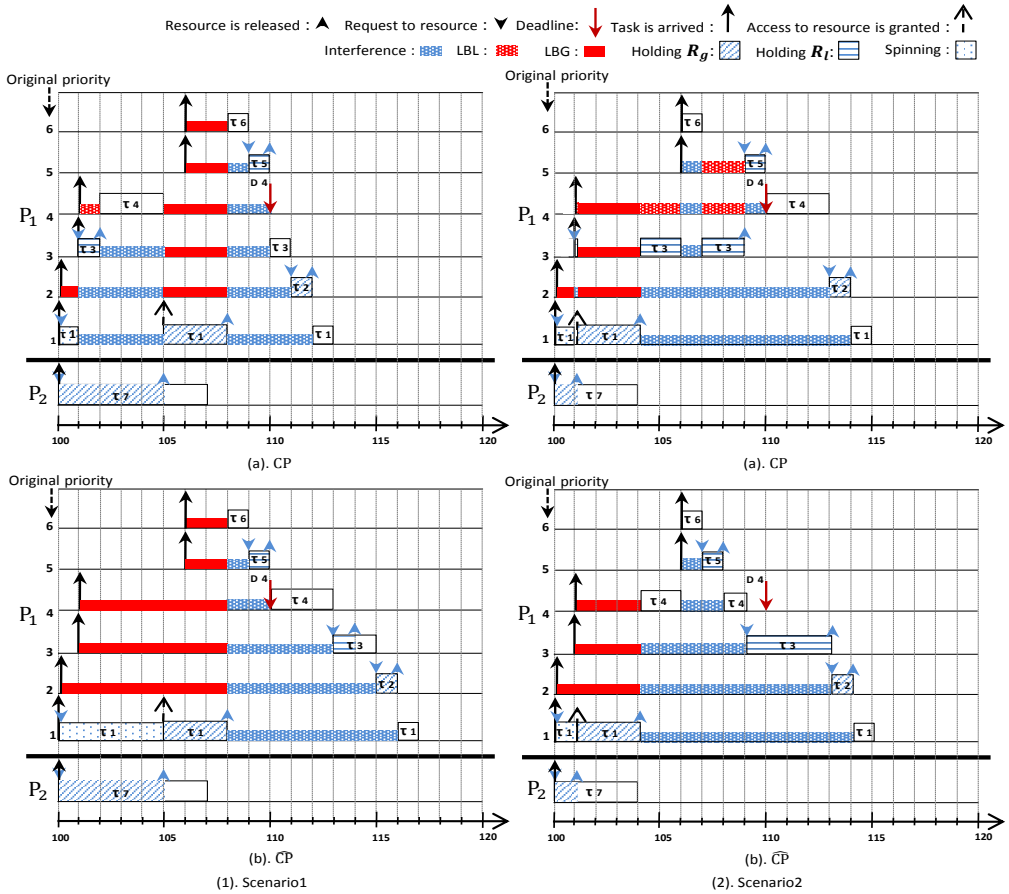

Figure 4 Incomparability of CP and $\widehat{CP}$. $\pi_{P_k}^{\text{spin}_{CP}} = 2$ and $\pi_{P_k}^{\text{spin}_{\widehat{CP}}} = 5$ in both scenarios. Scenario *(1)*: τ_4 misses its deadline under $\widehat{CP}$ but not under CP. Scenario *(2)*: τ_4 misses its deadline under CP but not under $\widehat{CP}$.

we need the maximum number of occurrence of each type of blocking i.e., LBL and LBG, to a task , and the incorporation of it with the maximum length of such blocking. To do so, we first, in Section 6.1, present the maximum possible number of blocking as well as the identified type (i.e., LBL/LBG) that a task may experience. Next, in Section 6.2 we calculate the maximum amount of such blocking using the identified number and type of blocking provided in Section 6.1.

6.1 Number and Type of Blocking

In this section we show by means of Lemmas 37 and 39, and Corollaries 36 and 38 the maximum number and type of blocking a task τ_i experiences under three determinant cases: *(i)* $\pi_{P_k}^{\text{L}} < \pi_i$, *(ii)* $\pi_i \leq \pi_{P_k}^{\text{spin}}$ and *(iii)* $\pi_{P_k}^{\text{spin}} < \pi_i \leq \pi_{P_k}^{\text{L}}$. It is enough to investigate the amount of blocking under the three aforementioned cases since other cases which appears under the assumption $\pi_{P_k}^{\text{L}} \leq \pi_{P_k}^{\text{spin}}$ falls under one of the above mentioned categories. As an example under such assumption, the cases $\pi_{P_k}^{\text{L}} < \pi_i \leq \pi_{P_k}^{\text{spin}}$ and $\pi_i \leq \pi_{P_k}^{\text{L}}$ both falls under Case *(ii)*.

Any LBG delay imposed to a task on a core P_k is due to a global resource access of a lower priority task which according to Definition 22 is nested within the access to the local spin resource $R_{P_k}^{\text{spin}}$ on P_k. According to SRP any job of a task can be blocked for at most one (outermost) local

critical section of any lower-priority task (Lemma 25). $R^{\text{spin}}_{P_k}$ is treated by SRP similar as any other regular local resource on P_k, thus, Lemma 25 can be extended to the following corollary.

▶ **Corollary 36.** *A job of any task $\tau_i \in \mathcal{T}_{P_k}$ can experience at most one LBG from any lower priority task.*

Next, we present the type and the maximum number of blocking that a task experiences under Cases *(i)* and *(ii)*.

▶ **Lemma 37.** *A job of a task $\tau_i \in \mathcal{T}_{P_k}$ experiences at most one either LBG or LBL due to normal local resources from any lower priority task under Cases* (i) $\pi^{\text{L}}_{P_k} < \pi_i$ *or* (ii) $\pi_i \leq \pi^{\text{spin}}_{P_k}$.

Proof. Under Case *(i)*, by definition, τ_i cannot experience any LBL delay due to a normal local resource. However, it still can experience blocking due to the local spin resource. Since according to Definition 22 access to the special local resource contains an access to a global resource. Thus τ_i, in the worst-cases, experiences LBG which according to Corollary 36 is at most one from any lower priority task. Therefore, the lemma is valid for this case. According to Lemma 25, τ_i experiences at most one LBL from lower priority task due to requesting local resources. Since under Case *(ii)*, $\pi_i \leq \pi^{\text{spin}}_{P_k}$ and remembering from Rule 21 that $\pi^{\text{spin}}_{P_k}$ is the ceiling of the special local spin resource on P_k, thus such an LBL to τ_i can be due to acquiring the local spin resource by a lower priority task. In other words, the imposed LBL to τ_i can be either due to a normal local resource or the spin resource on P_k. Moreover, since the access to the spin resource contains an access to a global resource, if τ_i experiences an LBL due to the local spin resource then it experiences an LBG. This concludes that τ_i experiences at most one either LBL due to a normal local resource or an LBG. This finishes the proof. ◀

To further realize the scenario of Lemma 37 let us assume that in the example in Figure 2 there exists another task τ_0 with priority lower than that of τ_2 besides τ_1 which also arrives earlier than τ_2 and uses the same local resource as τ_3. It is easy to observe that either τ_1 could issue its request for the "special" local resource $R^{\text{spin}}_{P_k}$ which has a ceiling equal to 2 and delay τ_2 upon its arrival or τ_0 could for its normal local resource with ceiling equal to 3.

Next, we present the type and the maximum number of blocking that a task experiences under Case *(iii)*. According to Corollary 36, a task τ_i experiences at most one LBG from lower priority tasks. According to Definition 22 an LBG to a task is due to non-preemptive global resource access of a lower priority task which is always nested within an access to the special spin resource $R^{\text{spin}}_{P_k}$. Unlike in Lemma 37, a task τ_i with a priority $\pi^{\text{spin}}_{P_k} < \pi_i$, cannot experience LBL due to the special spin resource $R^{\text{spin}}_{P_k}$ where $ceil_{P_k}(R^{\text{spin}}_{P_k}) = \pi^{\text{spin}}_{P_k}$ (Rule 21). However, since the access to global resource is non-preemptive (i.e., the priority of the task is raised higher than any task when the resource is granted, see Rule 18), thus, in the worst-case, τ_i experiences the resource access delay of an LBG from a lower priority task. Further, τ_i can experience at most one LBL a from a normal local resource when $\pi_i \leq \pi^{\text{L}}_{P_k}$ where according to Lemma 25 it can be at most one from any lower priority task. Therefore, the following corollary is drawn from Lemma 25 and Corollary 36.

▶ **Corollary 38.** *A job of a task $\tau_i \in \mathcal{T}_{P_k}$ experiences at most one LBL from lower priority tasks due to a normal local resource and one resource access delay of an LBG from any lower priority task under Case* (iii) $\pi^{\text{spin}}_{P_k} < \pi_i \leq \pi^{\text{L}}_{P_k}$.

The scenario of Corollary 38 can be remembered from Figure 2 where τ_4 experiences LBL from τ_3 and resource access blocking of an LBG from τ_1.

Next, we identify the set of lower priority tasks from Lemma 38 that causes LBL delay to a task τ_i under *(iii)*.

▶ **Lemma 39.** *If a job of a task $\tau_i \in \mathcal{T}_{P_k}$ experiences both an LBL and a resource access delay of an LBG then the LBL is caused by job of a lower priority task τ_l where $\pi_l > \pi^{\text{spin}}_{P_k}$.*

Proof. Let us assume that τ_i experiences LBG by the lower priority task τ_m due to a request for the global resource R_q which is issued at time t_m, and it experiences LBL by the lower priority task τ_l due to request for the local resource R_ℓ which is issued at time t_l.

According to Lemma 26 and having in mind Note 28, these requests are issued before τ_i's arrival at time t_i, thus, $t_l < t_i$ and $t_m < t_i$. However, one of the possible three following cases can be valid for t_l and t_m: *(i)* $t_l = t_m$, *(ii)* $t_l < t_m$, or *(iii)* $t_m < t_l$. According to Rule 21, τ_m first locks the special local resource $R^{\text{spin}}_{P_k}$ before locking R_q. Moreover, according to SRP, access to a local resource happens at the time of request. Thus, both τ_l and τ_m access R_ℓ and the special local resource $R^{\text{spin}}_{P_k}$ at times t_l and t_m, respectively. Further, according to Lemma 25, one LBL can happen to τ_i at any time which rules out the case *(i)*, i.e., $t_l \neq t_m$. Moreover, τ_l causing LBL to τ_i implies that it raises its priority to $ceil(R_\ell)$ at time t_l where $ceil(R_\ell) \geq \pi_i$. Therefore, no task with priority lower than that of τ_i can run in the interval $[t_l, t_i]$. This rules out case *(ii)*. Therefore, case *(iii)* is valid. τ_m first locks $R^{\text{spin}}_{P_k}$ at a time t_0 with $t_0 \leq t_m$ and raises its priority to the ceiling of this resource, i.e., $\pi^{\text{spin}}_{P_k}$ (Rule 21). Therefore, in order to τ_l be able to run at time $t_l > t_m$, it must be that $\pi_l > \pi^{\text{spin}}_{P_k}$. This finishes the proof. ◀

The scenario of Lemma 39 can be remembered from Figure 2 where τ_4 experiences both an LBL and LBG delay from τ_3 and τ_1, respectively where $\pi_3 > \pi^{\text{spin}}_{P_k} = 1$.

6.2 Amount of Blocking

In this section first we present maximum deuration of LBL and LBG delay that a task experiences by Corollary 40 and Lemmas 41 and 42. Finally, Theorem 43 concludes the total worst-case blocking delay calculation experienced by a task.

Maximum LBL duration experienced by a task is presented by the following corollary that is driven from Lemma 25.

▶ **Corollary 40.** *The maximum LBL blocking duration experienced by a task $\tau_i \in \mathcal{T}_{P_k}$ is formulated as follows.*

$$B^{\text{L}}_i = \max_{\forall j : \pi_j < \pi_i \wedge \tau_i, \tau_j \in \tau_{P_k}} B^{\text{L}}_{i,j}, \tag{8}$$

where $B^{\text{L}}_{i,j}$ is denoted as the maximum LBL delay duration imposed to a task $\tau_i \in \mathcal{T}_{P_k}$ by a local lower priority task τ_j and is calculated according to SRP specification [6] as below:

$$B^{\text{L}}_{i,j} = \max_{\substack{\forall l : R_l \in \mathcal{RS}^{\text{L}}_j \\ \wedge \pi_j < \pi_i \leq ceil_{P_k}(R_l)}} \{Cs_{j,l}\}. \tag{9}$$

In the following, we present the maximum LBG duration experienced by a task from a lower priority task.

▶ **Lemma 41.** *The maximum LBG blocking duration experienced by a task $\tau_i \in \mathcal{T}_{P_k}$ from any job of a local lower priority task τ_j using a spin-based protocol where $\pi^{\text{spin}}_{P_k} \geq \pi^{\text{G}}_{P_k}$ is denoted as $B^{\text{G}}_{i,j}(\pi^{\text{spin}}_{P_k})$ and is calculated as follows.*

$$B^{\text{G}}_{i,j}(\pi^{\text{spin}}_{P_k}) = \max_{\substack{\forall q : R_q \in \mathcal{RS}^{\text{G}}_j \\ \wedge \pi_j < \pi_i}} \left(Cs_{j,q} + \begin{cases} spin_{P_k,q} & \text{if } \pi_i \leq \pi^{\text{spin}}_{P_k} \\ 0 & \text{otherwise} \end{cases} \right), \tag{10}$$

where $spin_{P_k,q} = \sum_{\forall P_r \neq P_k} \max_{\forall \tau_j \in \mathcal{T}_{P_r,q}} \{Cs_{j,q}\}$ as in (4).

Proof. Since LBG is a type of pi-blocking (see Definition 8), thus, according to Lemma 27 (and having in mind Note 28), any job of a lower priority task τ_j can cause LBG to any job of a higher priority task τ_i at most once. Further according to Definition 22 the maximum duration of such a delay is $spin_{P_k,q} + \max_{\substack{\forall q,j:\tau_j \in \mathcal{T}_{P_k} \\ \wedge R_q \in \mathcal{RS}_j^{\mathrm{G}} \wedge \pi_j < \pi_i}} Cs_{j,q}$. However, the access to the special local resource $R_{P_k}^{\mathrm{spin}}$ by τ_j where $ceil_{P_k}(R_{P_k}^{\mathrm{spin}}) = \pi_{P_k}^{\mathrm{spin}}$ can preempt τ_i only if $\pi_i \leq \pi_{P_k}^{\mathrm{spin}}$. Based on this (10) is derived. ◀

Next, we present the maximum LBG duration experienced by a task from all lower priority tasks.

▶ **Lemma 42.** *The maximum LBG blocking duration experienced by a task τ_i from local lower priority tasks using a spin-based protocol where $\pi_{P_k}^{\mathrm{spin}} \geq \pi_{P_k}^{\mathrm{G}}$ is denoted as $B_i^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}})$ and is calculated as follows.*

$$B_i^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}}) = \max_{\forall j:\pi_j < \pi_i \wedge \tau_i, \tau_j \in \mathcal{T}_{P_k}} \{B_{i,j}^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}})\}, \tag{11}$$

where $B_{i,j}^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}})$ is calculated according to (10).

Proof. Follows immediately from Corollary 36 and Lemma 41. ◀

▶ **Theorem 43.** *The worst-case total pi-blocking experienced by a task $\tau_i \in \mathcal{T}_{P_k}$ when $\pi_{P_k}^{\mathrm{G}} \leq \pi_{P_k}^{\mathrm{spin}} \leq \pi_{P_k}^{\max}$ is denoted by $B_i(\pi_{P_k}^{\mathrm{spin}})$ and is calculated as follows.*

$$B_i(\pi_{P_k}^{\mathrm{spin}}) = \begin{cases} \begin{cases} a & \text{if } \pi_{P_k}^{\mathrm{L}} < \pi_i \\ b & \text{if } \pi_i \leq \pi_{P_k}^{\mathrm{L}} \end{cases} & \text{if } \pi_{P_k}^{\mathrm{L}} \leq \pi_{P_k}^{\mathrm{G}} \\ \begin{cases} c & \text{if } \pi_{P_k}^{\mathrm{L}} < \pi_i \\ d & \text{if } \pi_{P_k}^{\mathrm{spin}} < \pi_i \leq \pi_{P_k}^{\mathrm{L}} \\ e & \text{if } \pi_i \leq \pi_{P_k}^{\mathrm{spin}} \end{cases} & \text{if } \pi_{P_k}^{\mathrm{G}} < \pi_{P_k}^{\mathrm{L}} \end{cases} \tag{12}$$

where,

$$a = c = B_i^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}}), \tag{13}$$

$$b = e = \max(B_i^{\mathrm{L}}, B_i^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}})), \tag{14}$$

$$d = \max\left(\max_{\substack{\forall j:\pi_j < \pi_i \\ \wedge \pi_{P_k}^{\mathrm{spin}} < \pi_j \\ \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B_{i,j}^{\mathrm{L}}\} + B_i^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}}), \max_{\substack{\forall j:\pi_j < \pi_i \\ \wedge \pi_j \leq \pi_{P_k}^{\mathrm{spin}} \\ \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B_{i,j}^{\mathrm{L}}\} \right), \tag{15}$$

and $B_{i,j}^{\mathrm{L}}$, B_i^{L} and $B_i^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}})$ are calculated according to (9), (8) and (11).

Proof. We prove the calculation of the terms a, b, c, d and e under clauses *(a)*, *(b)*, *(c)*, *(d)* and *(e)*, respectively.

Proof of Clauses *(a)* and *(c)*: when $\pi_{P_k}^{\mathrm{L}} < \pi_i$ a task τ_i cannot experience any LBL due to normal local resources, thus, $B_i^{\mathrm{L}} = 0$. However, according to Corollary 36, τ_i can experience at most one LBG delay which its maximum duration according to Lemma 42 is (13).

Proof of Clauses *(b)* and *(e)*: according to the conditions of Clause *(b)* $\forall \tau_i | \pi_i \leq \pi^{\mathrm{L}}_{P_k}$, it is valid that $\pi_i \leq \pi^{\mathrm{G}}_{P_k}$ since $\pi^{\mathrm{L}}_{P_k} \leq \pi^{\mathrm{G}}_{P_k}$ under this clause. Further, since $\pi^{\mathrm{G}}_{P_k} \leq \pi^{\mathrm{spin}}_{P_k}$ (Definition 6) thus, $\pi_i \leq \pi^{\mathrm{spin}}_{P_k}$ under the conditions of Clause *(b)*, which is the same as the condition of Clause *(e)*. According to Lemma 37, τ_i experiences at most one either LBL or LBG from lower priority tasks when $\pi_i \leq \pi^{\mathrm{spin}}_{P_k}$. Thus, both terms *(b)* and *(e)* are inferred based on Corollary 40 and Lemma 42 which introduce the maximum amount of such blocking.

Proof of Clause *(d)*: we assume that the term d is constructed by three elements λ_1, γ and λ_2 such that $\lambda_1 = \max_{\substack{\forall j:\pi^{\mathrm{spin}}_{P_k} < \pi_j \\ \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B^{\mathrm{L}}_{i,j}\}$, $\gamma = B^{\mathrm{G}}_i(\pi^{\mathrm{spin}}_{P_k})$ and $\lambda_2 = \max_{\substack{\forall j:\pi_j \leq \pi^{\mathrm{spin}}_{P_k} \\ \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B^{\mathrm{L}}_{i,j}\}$. We construct three cases *(i)*, *(ii)* and *(iii)* for each of which we present a worst-case blocking delay that is imposed to τ_i under this clause.

Case *(i)*: since under the condition of Clause *(d)* $\pi^{\mathrm{spin}}_{P_k} < \pi_i \leq \pi^{\mathrm{L}}_{P_k}$, thus, according to Corollary 38 and Lemma 39, τ_i experience in the worst-case both a resource access delay of an LBG and an LBL from a lower priority task τ_j where $\pi^{\mathrm{spin}}_{P_k} < \pi_j$. By considering Lemma 42 and (9), this leads to τ_i experience at most a blocking equal to $\lambda_1 + \gamma$.

Case *(ii)*: on the other hand, it is true that according to Lemma 25 a task can experience at most one LBL from any lower priority task. By looking at (8), it is easy to see that the maximum LBL imposed to τ_i can be rewritten as $\max(\lambda_1, \lambda_2)$.

Case *(iii)*: furthermore, according to Corollary 36, it is also true that a task can experience at most one LBG from any lower priority task which based on Lemma 42 the maximum of such delay is the term γ.

All the three aforementioned cases are valid. However, the only way to find the maximum imposed delay to a task is to find the one that gives rise to the maximum blocking imposed to τ_i since they are overlapping cases. According to Lemma 25 the blocking delay derived under case *(i)* and case *(ii)* cannot both be imposed to τ_i. Similarly, according to Corollary 36 the blocking delay derived under case *(i)* and case *(iii)* cannot both be imposed to τ_i as well. Further, occurrence of blocking delay of case *(ii)* and *(iii)* is case *(i)*. Therefore, to find the maximum delay imposed to τ_i we present β that gives the maximum delay imposed under each of the three cases where, $\beta = \max(\lambda_1 + \gamma, \gamma, \max(\lambda_1, \lambda_2)) = \max(\lambda_1 + \gamma, \max(\lambda_1, \lambda_2))$. Let us assume two scenarios: *(1)* $\lambda_1 < \lambda_2$ and *(2)* $\lambda_2 \leq \lambda_1$. Under scenario *(1)*, $\beta = \max(\lambda_1 + \gamma, \lambda_2)$ which is the same as term d in (12). On the other hand, under scenario *(2)* $\beta = \lambda_1 + \gamma$. However, since under this scenario it is derived that $\lambda_2 \leq \lambda_1 + \gamma$, thus, $\max(\lambda_1 + \gamma, \lambda_2)$, i.e., term d, gives similar result as β here. As a result both scenarios *(1)* and *(2)* can be presented with the term d. This finishes the proof. ◀

It is easy to observe that the total pi-blocking to a task $\tau_i \in \mathcal{T}_{P_k}$, i.e., $B_i(\pi^{\mathrm{spin}}_{P_k})$ in (16) presented by Corollary 44 gives similar terms as in (12) under those conditions.

▶ **Corollary 44.** *$B_i(\pi^{\mathrm{spin}}_{P_k})$ for a task $\tau_i \in \mathcal{T}_{P_k}$ is presented as follows.*

$$B_i(\pi^{\mathrm{spin}}_{P_k}) = \max \left(\max_{\substack{\forall j:\pi_j < \pi_i \\ \wedge \pi^{\mathrm{spin}}_{P_k} < \pi_j \\ \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B^{\mathrm{L}}_{i,j}\} + B^{\mathrm{G}}_i(\pi^{\mathrm{spin}}_{P_k}), \max_{\substack{\forall j:\pi_j < \pi_i \\ \wedge \pi_j \leq \pi^{\mathrm{spin}}_{P_k} \\ \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B^{\mathrm{L}}_{i,j}\} \right), \tag{16}$$

▶ **Note 45.** *Similar to HP and CP, $spin_{P_k,q}$, $spin_i$ are calculated as in (4) and (5), respectively and the inflated execution time of a task τ_i, i.e., $\acute{C}_i$ is calculated according to Definition 12.*

6.3 Tighter Bounds under *CP*

In this section, we show by means of Lemma 46 that the new analysis for calculating the blocking terms given in (16) provides tighter bounds for *CP* compared to the analysis using (7), i.e. as given in [1].

▶ **Lemma 46.** *Maximum blocking imposed to any task τ_i under CP spin-based protocol, gives tighter bounds using (16) compared to the blocking analysis given using (7), i.e. as given in [1]in [1].*

Proof. We define the maximum blocking under *CP* that is presented in Section 3.4.2 by (7) as B_i^{old} and the maximum blocking under *CP* that is calculated by (16) as B_i^{new}.

By looking at (7), B_i^{old} is calculated differently, as presented by the following clauses that depend on the priority π_i of τ_i and the spin-priority $\pi_{P_k}^{\text{G}}$:

(i) if $\pi_{P_k}^{\text{G}} + 1 < \pi_i$ then $B_i^{\text{old}} = \acute{a} + \acute{b}$,
(ii) if $\pi_i = \pi_{P_k}^{\text{G}} + 1$ then $B_i^{\text{old}} = \max(\acute{a}, \acute{b})$,
(iii) if $\pi_i \leq \pi_{P_k}^{\text{G}}$ then $B_i^{\text{old}} = \max(\acute{a}, \tilde{b})$,

where $\acute{a}$ is the same as (8), i.e., $\acute{a} = \max_{\substack{\forall l: R_l \in \mathcal{RS}_j^{\text{L}} \\ \wedge \pi_j < \pi_i \leq ceil_{P_k}(R_l)}} \{Cs_{j,l}\}$, $\acute{b} = \max_{\substack{\forall j,q: \pi_j < \pi_i \\ \wedge \tau_i, \tau_j \in \mathcal{T}_{P_k} \wedge R_q \in \mathcal{RS}_j^{\text{G}}}} Cs_{j,q}$ and $\tilde{b} = \max_{\substack{\forall j,q: \pi_j < \pi_i \\ \wedge \tau_i, \tau_j \in \mathcal{T}_{P_k} \wedge R_q \in \mathcal{RS}_j^{\text{G}}}} \{Cs_{j,q} + spin_{P_k,q}\}$.

We investigate each clause separately. From (16) let us assume $B_i^{\text{new}} = \max(a + b, c)$ where $a = \max_{\substack{\forall j: \pi_j < \pi_i \\ \wedge \pi_{P_k}^{\text{G}} < \pi_j \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B_{i,j}^{\text{L}}\}$, $b = \max_{\forall j: \pi_j < \pi_i \wedge \tau_i, \tau_j \in \mathcal{T}_{P_k}} \{B_{i,j}^{\text{G}}(\pi_{P_k}^{\text{spin}})\}$ calculated from (10) and $c = \max_{\substack{\forall j: \pi_j < \pi_i \\ \wedge \pi_j \leq \pi_{P_k}^{\text{G}} \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B_{i,j}^{\text{L}}\}$. It can be observed under the condition of clause *(i)* that the set where $B_{i,j}^{\text{L}}$ is specified in a, i.e., $\forall j | \pi_j < \pi_i \wedge \pi_{P_k}^{\text{G}} < \pi_j$ is smaller than the set to specify $B_{i,j}^{\text{L}}$ in $\acute{a}$ where it is $\forall j | \pi_j < \pi_i$. This implies that $a \leq \acute{a}$. It also can be observed that $b = \acute{b}$ since the condition $\pi_i \leq \pi_{P_k}^{\text{spin}} = \pi_{P_k}^{\text{G}}$ is not valid in (10) under this clause. Moreover, it can be seen that the set to specify $B_{i,j}^{\text{L}}$ is smaller for c than $\acute{a}$, which leads to $c \leq \acute{a}$. As a result, $B_i^{\text{new}} \leq B_i^{\text{old}}$ under clause *(i)*.

It can be observed that under the condition of clause *(ii)* $a = 0$, $b = \acute{b}$ and $c = \acute{a}$, thus $B_i^{\text{new}} = \max(0 + \acute{b}, \acute{a})$ which means $B_i^{\text{new}} = B_i^{\text{old}}$ under clause *(ii)*. Furthermore, it can be observed that under the condition of clause *(iii)* $a = 0$, $b = \tilde{b}$ since the condition $\pi_i \leq \pi_{P_k}^{\text{spin}} = \pi_{P_k}^{\text{G}}$ is valid in (10) under this clause and $c = \acute{a}$, thus $B_i^{\text{new}} = \max(0 + \tilde{b}, \acute{a})$ which leads to $B_i^{\text{new}} = B_i^{\text{old}}$ under clause *(iii)*. This finishes the proof. ◀

6.4 Use of ILP

In this section we discuss the benefit of using optimization approaches such as mixed-Integer linear program (ILP) for bounding the maximum cumulative blocking imposed to tasks similar to [29] by Wieder and Brandenburg. Wieder and Brandenburg [29] showed that any blocking analysis that is based on inflation of the worst-case execution times of tasks with remote blocking can be pessimistic by a factor of $\Omega(\phi \cdot n)$ where $\phi \approx \lceil \frac{WR_i}{T_h} \rceil$ and τ_h is a higher priority task that spins and delays a lower priority task τ_i (Theorem 1 [29]).

Based on such a result, the tasks on a core with a priority within the range $(\pi_{P_k}^{\text{G}}, \pi_{P_k}^{\max}]$ do not suffer from such pessimism since, by definition, tasks in this range do not use any global resource (see Note 13). Therefore, using ILP could not tighten the blocking analysis for this range of tasks nor could benefit our comparative evaluation later in Section 8. Based on the results derived from Corollaries 49, 50 and 51 we show that it is enough to consider the tasks with a priority within

the above mentioned range when comparing the effectiveness of different spin-based protocols with a spin-lock priority from the same range.

However, ILP can tighten the blocking imposed to tasks on a core with a priority within the range $[1, \pi^{\mathrm{G}}_{P_k}]$, where 1 is the lowest possible priority level that a task can have on a core, since for this range of tasks the worst-case execution times are inflated with the remote blocking parameter in case those tasks use global resources. For tasks on a core with a priority within the range $[1, \pi^{\mathrm{G}}_{P_k}]$ selecting any spin-based protocol that uses a spin-lock priority within the range $[\pi^{\mathrm{G}}_{P_k}, \pi^{\max}_{P_k} - 1]$ will give the same blocking bound as when using *HP* (see Lemma 47 in Section 7.1). Therefore, for the tasks with a priority in the range $[1, \pi^{\mathrm{G}}_{P_k}]$ the same ILP constraints that has been presented for FIFO-ordered non-preemptive spin-based protocols [29] could be used to tighten the blocking bounds. Therefore, for the simplicity of the experiments we use the traditional analysis based on inflation of worst-case execution times of tasks when the schedulability of a core is checked for the set of tasks with a priority lower than this range.

Moreover, the holistic analysis of spin locks [9] encompasses pessimism due to inflating tasks' execution times [29] which has been overcome by the ILP-based analysis presented in [29]. However, since ILP cannot tighten the analysis for the set of tasks considered in our comparative evaluation, therefore, holistic analysis cannot as well. Thus, we do not consider this analysis approach here as well.

7 Properties of Spin-Based Protocols

In this section we specify the set of tasks on a processor for which the selection of any two spin-based protocols from the triple (*HP*, *CP*, $\widehat{CP}$), yield the same worst-case blocking bounds for a task τ_i. This facilitates the evaluation of the results, later, in Section 8. First, we present Lemmas 47 and 48 under which we show, respectively, that for selection of any two spin-lock priorities from the range $[\pi^{\mathrm{G}}_{P_k}, \pi^{\max}_{P_k}]$ where one is smaller than the other, for a task τ_i with a priority either *(a)* lower than the priority of both or *(b)* higher than the priority of both if τ_i does not use any local resource, using either spin-lock priorities will lead to the same blocking bounds for τ_i. This will help us to specify the set of tasks for which using either *CP* or *HP*, using either $\widehat{CP}$ or *HP* and using either either *CP* or $\widehat{CP}$ will lead to the same blocking bounds that we present by Corollaries 49, 50 and 51, respectively.

▶ **Lemma 47.** *Assume two different spin-based protocols σ_1 and σ_2 on P_k, with spin-lock priorities $\pi^{\mathrm{spin}_{\sigma_1}}_{P_k}$ and $\pi^{\mathrm{spin}_{\sigma_2}}_{P_k}$, respectively, where $\pi^{\mathrm{spin}_{\sigma_1}}_{P_k}, \pi^{\mathrm{spin}_{\sigma_2}}_{P_k} \in [\pi^{\mathrm{G}}_{P_k}, \pi^{\max}_{P_k}]$ and $\pi^{\mathrm{spin}_{\sigma_1}}_{P_k} \leq \pi^{\mathrm{spin}_{\sigma_2}}_{P_k}$. For any task $\tau_i \in \mathcal{T}_{P_k}$ where $\pi_i \leq \pi^{\mathrm{spin}_{\sigma_1}}_{P_k}$, using either σ_1 or σ_2 will yield the same value for the worst-case blocking B_i.*

Proof. We assume B_i which is calculated by (16) is equal to $max(A + B, C)$ where A, B and C are $A = \max_{\substack{\forall j:\pi_j<\pi_i \\ \wedge\pi^{\mathrm{spin}}_{P_k}<\pi_j\wedge\tau_j\in\mathcal{T}_{P_k}}} \{B^{\mathrm{L}}_{i,j}\}$, $B = B^{\mathrm{G}}_i(\pi^{\mathrm{spin}}_{P_k})$ and $C = \max_{\substack{\forall j:\pi_j<\pi_i \\ \wedge\pi_j\leq\pi^{\mathrm{spin}}_{P_k}\wedge\tau_j\in\mathcal{T}_{P_k}}} \{B^{\mathrm{L}}_{i,j}\}$. It is easy to see that $A = 0$ for any task τ_i that $\pi_i \leq \pi^{\mathrm{spin}_{\sigma_1}}_{P_k}$. Moreover, B is the same when using either σ_1 or σ_2 due to the fact that the condition $\pi_i \leq \pi^{\mathrm{spin}}_{P_k}$ in (10), which is the set from which B is derived, is satisfied using both σ_1 or σ_2. Further, C is also the same when using either σ_1 or σ_2 since the sets $\forall j : \pi_j \leq \pi^{\mathrm{spin}_{\sigma_1}}_{P_k}$ and $\forall j : \pi_j \leq \pi^{\mathrm{spin}_{\sigma_2}}_{P_k}$, which are the sets from which C is derived when σ_1 and σ_2 are used, respectively, are the same and leads to $\forall j : \pi_j < \pi_i$. ◀

▶ **Lemma 48.** *Assume two different spin-based protocols σ_1 and σ_2 on P_k, with spin priority levels $\pi^{\mathrm{spin}_{\sigma_1}}_{P_k}$ and $\pi^{\mathrm{spin}_{\sigma_2}}_{P_k}$ (remember Definition 6), respectively, where $\pi^{\mathrm{spin}_{\sigma_1}}_{P_k}, \pi^{\mathrm{spin}_{\sigma_2}}_{P_k} \in [\pi^{\mathrm{G}}_{P_k}, \pi^{\max}_{P_k}]$ and*

$\pi_{P_k}^{\text{spin}_{\sigma_1}} \leq \pi_{P_k}^{\text{spin}_{\sigma_2}}$, *and* $\pi_{P_k}^{\text{LG}} \leq \pi_{P_k}^{\text{spin}_{\sigma_2}}$. *For any task* $\tau_i \in \mathcal{T}_{P_k}$ *where* $\pi_{P_k}^{\text{spin}_{\sigma_2}} < \pi_i$, *then using either* σ_1 *or* σ_2 *will lead to the same worst-case blocking results.*

Proof. By assuming B_i calculated by (16) equal to $max(A+B, C)$ similar as in proof of Lemma 47, it is easy to see that $A = C = 0$ since by $\pi_{P_k}^{\text{LG}} < \pi_i$, by definition τ_i does not use any local resource which leads $B_{i,j}^{\text{L}} = 0$ in (16). Further, the condition of $\pi_i \leq \pi_{P_k}^{\text{spin}}$ in (10), which is the set from which B is derived, is not satisfied under both σ_1 and σ_2, thus B_i is the same under both σ_1 and σ_2 knowing that by definition, $\forall \tau_j | \pi_{P_k}^{\text{G}} < \pi_j < \pi_i \Rightarrow \mathcal{RS}_i^{\text{G}} = 0$. ◀

7.1 *CP* versus *HP*

In the following, we specify the set of tasks on a processor P_k that have the same blocking bounds for *CP* and *HP*.

From Lemma 47 the following corollary can be drawn.

▶ **Corollary 49.** *For any task* $\tau_i \in \mathcal{T}_{P_k}$ *where* $\pi_i \leq \pi_{P_k}^{\text{G}}$, *(i.e.,* τ_i*'s priority is in range C in Figure 3) then using either CP or HP will lead to the same blocking bounds.*

7.2 $\widehat{CP}$ versus *HP*

In the following, we specify the set of tasks on a processor P_k that have the same blocking bounds for *HP* and $\widehat{CP}$.

From Lemma 47, we draw the following conclusion.

▶ **Corollary 50.** *For any task* $\tau_i \in \mathcal{T}_{P_k}$ *where* $\pi_i \leq \pi_{P_k}^{\text{LG}}$, *(i.e.,* τ_i*'s priority is in ranges B or C in Figure 3) then using either* $\widehat{CP}$ *or HP will lead to the same blocking bounds.*

7.3 $\widehat{CP}$ versus *CP*

In the following, we specify the set of tasks on a processor P_k that have the same blocking bounds for *CP* and $\widehat{CP}$. This simplifies the comparison of the two protocols later in Section 8.

From Lemmas 47 and 48, we draw the following conclusion.

▶ **Corollary 51.** *If* $\pi_{P_k}^{\text{G}} \leq \pi_{P_k}^{\text{LG}}$, *for any task* $\tau_i \in \mathcal{T}_{P_k}$ *where* $\pi_i \leq \pi_{P_k}^{\text{G}}$, *or* $\pi_{P_k}^{\text{LG}} < \pi_i$ *(i.e.,* τ_i*'s priority is in range C or A in Figure 3) then using either CP or* $\widehat{CP}$ *will lead to the same blocking bounds.*

We already have shown in Section 5.2 by means of an illustrative example that *CP* and $\widehat{CP}$ are incomparable. The same result is also achievable based on the worst-case response time using (16) for calculating the blocking term.

▶ **Example 52.** The blocking term and worst-case response time of task τ_4 in Example35 for scenario *(1)* and *(2)* are denoted by $B_4^{sc_1-CP}$, $WR_4^{sc_1-CP}$ and by $B_4^{sc_2-CP}$, $WR_4^{sc_2-CP}$ for *CP* and by $B_4^{sc_1-\widehat{CP}}$, $WR_4^{sc_1-\widehat{CP}}$ and by $B_4^{sc_2-\widehat{CP}}$, $WR_4^{sc_2-\widehat{CP}}$ for $\widehat{CP}$, respectively. Their values for τ_4 under scenario *(1)* are as follows: $B_4^{sc_1-CP} = 4$, thus $WR_4^{sc_1-CP} = 9$ and $B_4^{sc_1-\widehat{CP}} = 8$, thus $WR_4^{sc_1-\widehat{CP}} = 13$, and under scenario *(2)* are as follows: $B_4^{sc_2-CP} = 7$, thus $WR_4^{sc_2-CP} = 12$ and $B_4^{sc_2-\widehat{CP}} = 4$, thus $WR_4^{sc_2-\widehat{CP}} = 9$. Therefore, τ_4 misses its deadline using $\widehat{CP}$ in scenario *(1)* and using *CP* in scenario *(2)*.

7.4 Key Trade-Off Factors

In this section, we elaborate the trade-off factors between CP and $\widehat{CP}$. According to Corollary 51 only if a task's priority level is within range B, then using $\widehat{CP}$ and CP may lead to different schedulability results. Note that, $\widehat{CP}$ exists only when $\pi^{\mathrm{G}}_{P_k} < \pi^{\mathrm{L}}_{P_k}$ (see Section 5). It can be observed that the maximum blocking imposed to a task τ_i with a priority within range B calculated by (16) results in term e in (12) when $\widehat{CP}$ is used and term d when CP is used. Thus, to compare the maximum blocking delay under CP and $\widehat{CP}$, we should compare (14) and (15). By looking at these two terms, it can be observed that using CP may cause an extra LBL term besides the LBG term compared to using $\widehat{CP}$. On the other hand, according to (41) the LBG term can be smaller under CP compared to $\widehat{CP}$ since $spin_{P_k,q}$ term is zero for a task τ_i in range B if CP is used. In other words, such a task has to wait for its lower priority task's spin-lock time under $\widehat{CP}$ but not under CP.

Followed by the discussion above, a task τ_i may experience one extra LBL if CP is used, whereas it may experience longer LBG if $\widehat{CP}$ is used since it has to wait for the spin-lock time of a lower priority task. These two parameters determine the trade-off factors of the two protocols. One conclusion from this discussion is that if the extra LBL is not imposed under CP, then CP outperforms $\widehat{CP}$ since under $\widehat{CP}$ a task may be delayed longer due to the spinning of lower priority tasks.

7.5 Intermediate Spin-Based Protocol

In Sections 5.2 and 7.3, we showed that CP and $\widehat{CP}$ are incomparable by means of both a trace example and analysis in Examples 35 and 52, respectively. In this section, we show for the same example using a third scenario (Scenario 3) through the analysis results that if CP and $\widehat{CP}$ cannot make a task set schedulable on a core, there may exist an intermediate spin-lock priority within the range $(CP, \widehat{CP})$ that can make the task set schedulable. Under a third scenario *(3)* we denote the blocking term and worst-case response time of a task τ_i under CP by $B_i^{sc_3-CP}$ and $WR_i^{sc_3-CP}$, under $\widehat{CP}$ by $B_i^{sc_3-\widehat{CP}}$ and $WR_i^{sc_3-\widehat{CP}}$ and under a third protocol which we call $\widetilde{CP}$ by $B_i^{sc_3-\widetilde{CP}}$ and $WR_i^{sc_3-\widetilde{CP}}$, respectively. Under scenario *(3)* we again use dedicated task specifications $(C_i, Cs_{i,l}, Cs_{i,g})$ for task τ_3 and τ_7, i.e., τ_3: (2, 2, 0) and τ_7: (7, 0, 5). $\pi_{P_k}^{\mathrm{spin}_{CP}} = 2$, $\pi_{P_k}^{\mathrm{spin}_{\widehat{CP}}} = 5$ and $\pi_{P_k}^{\mathrm{spin}_{\widetilde{CP}}} = 3$. The blocking terms and respective worst-case response times of τ_4 under this scenario are as follows: $B_4^{sc_3-CP} = 5$, thus $WR_4^{sc_1-CP} = 10$, $B_4^{sc_3-\widehat{CP}} = 8$, thus $WR_4^{sc_1-\widehat{CP}} = 13$, and $B_4^{sc_3-\widetilde{CP}} = 3$, thus $WR_4^{sc_1-\widetilde{CP}} = 9$. Therefore, since τ_4 misses its deadline under both CP and $\widehat{CP}$ but not under $\widetilde{CP}$, thus the task set is schedulable under $\widetilde{CP}$ only. To determine such spin-lock priority, if any, the priority levels between CP and $\widehat{CP}$ need to be explored linearly, applying worst-case response time calculations using (16). Please note that finding $\widehat{CP}$ limits the search. In general, spin-lock priorities in the range $(CP, \widehat{CP})$ do not necessarily dominate $HP.$, whereas $\widehat{CP}$ does. Moreover, whenever a task set is schedulable by CP and $\widehat{CP}$, this does not imply that the set is also schedulable by all, some or even any spin-lock priority in the range $(CP, \widehat{CP})$. We have performed an experiment which shows that out of 8000 task sets $\widetilde{CP}$ could only schedule 2 more task sets compared to both CP and $\widehat{CP}$.

8 Evaluation

In this section we present the experimental results of comparing HP, CP and $\widehat{CP}$. According to Corollaries 49, 50 and 51, it is enough to compare the worst-case response time of tasks of range B in Figure 3 when comparing CP and $\widehat{CP}$, range A when comparing $\widehat{CP}$ and HP and range

$A \cup B$ when comparing *CP* and *HP*. In our experiments, we therefore only consider tasks in the related range, effectively ignoring tasks that have identical results under the compared protocols. As we discussed in Section 6.4, using ILP cannot tighten the blocking bounds for this range of tasks. Therefore, for simplicity, we use the traditional worst-case response time analysis [4] for all tasks on a core. We run two types of experiments. In our first type of experiments we calculate the improvement in worst-case response time of tasks of one protocol compared to the other. We use $RTI(a, b)$ to denote the *response time improvement* under protocol a compared to protocol b. Since the response jitter of a task is bounded by the difference of the worst-case and best-case response time of that task [25, 11] and the best-case response time being independent of a global resource sharing protocol, the bound on the response jitter decreases when the worst-case response time decreases. Therefore, response time improvement of tasks are directly correlated with response jitters. We denote $RTI_i(a, b)$ for a task τ_i as $\frac{(WR_i^b - WR_i^a)}{max(WR_i^a, WR_i^b)} \times 100$, where WR_i^a and WR_i^b denote the worst-case response time of task τ_i under protocols a and b, respectively. For a randomly generated task set, we show the percentage of tasks as a function of $RTI(a, b)$. We have performed the experiments for how different system parameters as the number of processors, task set utilization, number of tasks per core and local and global critical section lengths can affect the $RTI(a, b)$. Due to space constraints, we only illustrate *RTI* for changing the numbers of cores as well as local critical section lengths here. Further results can be found in [2]. In the second type of experiments, we compare the system schedulability under *HP*, *CP* and $\widehat{CP}$. We perform the response-time analysis [4] to check the schedulability of task sets according to [18]. For this experiment we have generated 200,000 task sets for 4-processors. We denote $PS_{(C)}$ as the percentage of the schedulable systems under condition C, e.g., $PS_{(HP \wedge CP \wedge \neg \widehat{CP})}$ denotes the percentage of systems that are schedulable under both *HP* and *CP* but not under $\widehat{CP}$. This percentage is calculated based on the number of systems that are schedulable under any of the three aforementioned spin-based protocols.

8.1 Experimental Setup

In each experiment we randomly generate task sets for each processor. The number m of processors is selected from the set {4, 8, 12, 16}. For each experiment 1000 schedulable task sets under the considered protocols are generated. The task set size is the same for each processor and is selected from the set {20, 40, 60}. Tasks are randomly selected to be dedicated to ranges A, B and C in Figure 3 such that at least one task is dedicated to priority ranges A and B each in order to implement $\widehat{CP}$ and *CP* protocols for the sake of comparison. The task set utilization is also the same for each processor and is selected from the set {0.4, 0.6, 0.8}. The UUnifast algorithm [7] is used to generate the utilization of each task. The period of each task is randomly generated from the range [10, 150] ms with a granularity of 10 ms. The worst-case execution time of a task is calculated by $C_i = U_i \times T_i$. Deadlines of tasks are selected randomly according to a uniform distribution in the range $[C_i + \alpha \times (T_i - C_i), T_i]$ with $\alpha = 0.5$ as the default [14]. The maximum number of accesses to local and global resources for each task is 4. The local and global critical section lengths (lcs and gcs) are generated according to $Cs_q = \beta \times C_i$, where β is selected from the set $\{0.1, 0.2, 0.3\}$. The number of local resources per processor as well as number of global resources per task set is set to 3.

In our basic system configuration which is used for both types of experiments in Sections 8.2 and 8.3, the number of processors is set to $m = 4$, the task set utilization per core is 0.6, the number of tasks on each processor is 20, and $\beta = 0.2$.

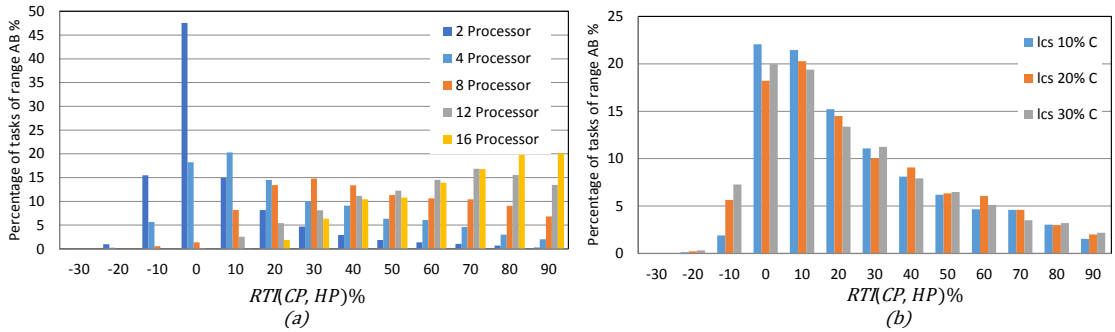

Figure 5 RTI for CP versus HP for (a) different number of processors m, and (b) different values of lcs.

8.2 Results for Response Time Improvements

For these experiments bar charts will be used to visualize the results, and the presented graphs show the distribution of the tasks for the calculated RTI. The X-axis in the graphs represents $RTI(a, b)$ and the Y-axis shows the percentage of the examined tasks that have that improvement. Note that, values in the X-axis present a non-continuous range. A bar in a graph that presents $RTI(a, b)$ with x_i as X value and y_i as Y value shows that $y_i\%$ of tasks have an improvement in the range $(x_{i-1}\%, x_i\%]$ in their response times under protocol b compared to protocol a. Note that a positive RTI value for a graph representing $RTI(a, b)$, shows that response times under protocol b are larger compared to protocol a. Similarly a negative RTI value shows that response times are smaller under protocol b compared to protocol a. The results in Figures 6, 5 and 7 show the variation in distribution of tasks for the calculated RTI values for different numbers of processors. More experimental results are available in [2] from which similar conclusions are derived as from the graphs presented here.

8.2.1 Evaluation Results of CP versus HP

Figure 5 shows that CP improves response time of tasks up to 90% compared to HP. In more detail, it can be observed from Figure 5.(a) that increasing the number of cores leads to more tasks having larger response time improvement under CP compared to HP. For $m = 2$ around 1% of tasks have up to 20% improvement. The same trend was also obtained by increasing the global critical section lengths for which the results can be found in [2]. This results are confirmed by revisiting (16) where increasing the number of cores and global critical sections is positively correlated with $spin_{P_k,q}$ included in $B_i^{\mathrm{G}}(\pi_{P_k}^{\mathrm{spin}})$ which is zero for the compared tasks under CP and not under HP. Moreover, Figure 6.(b) shows that by increasing the local critical section lengths, CP's performance decreases compared to HP with regard to response time improvement. This is due to the fact that by increasing the local critical sections $B_{i,j}^{\mathrm{L}}$ increases. The same trend was also obtained by increasing the number of tasks per core [2]. The reason is that by increasing the number of tasks on a core, the number of tasks in range B may increase as well, which in the worst-case leads to in an increase in the first $B_{i,j}^{\mathrm{L}}$ term in RHS of (16). This term is zero under HP. Increasing the task set utilization did not show a significant improvement. The reason is that by increasing the task set utilization the execution time of tasks are increased leading in an increase in both local and global critical section lengths which seems to nullify the effect of each other. The interesting observation is to have both positive and negative RTI values in the graphs which shows response time improvement under both CP and HP. This confirms the incomparability

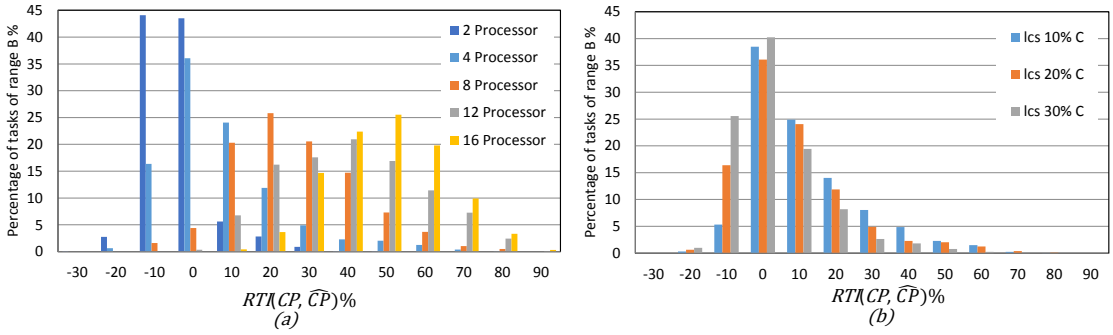

Figure 6 *RTI* for *CP* versus $\widehat{CP}$ for (*a*) different number of processors m, and (*b*) different values of *lcs*.

claim of *CP* and *HP* which we previously have shown by examples [1]. In conclusion, the sum of the percentages for the positive values is larger than the sum of the percentages for the negative values from graphs. Hence, overall with the given system configuration in these experiments, *CP* introduces less delays to tasks compared to *HP*.

8.2.2 Evaluation Results of *CP* versus $\widehat{CP}$

Figure 6 shows that, in general, *CP* improves response time of tasks compared to $\widehat{CP}$ which can reach up to 80%. However, it can be observed that when the number of cores are small $\widehat{CP}$ could improve response time of tasks up to 20%. In more detail, it can be observed in Figure 6.(a) that for $m = 2$, roughly 45% of tasks have up to 10% response time improvement under $\widehat{CP}$ compared to *CP* and around 3% have up to 20% improvement. Similar trend is achieved here as well by increasing the number of cores, global critical and local critical sections, task set size and utilization similar to when comparing *CP* and *HP*. The reason is that for all tasks of range *B* the spin-lock priority under $\widehat{CP}$ is higher than their priority similar to spin-lock priority under *HP*, when comparing *CP* versus $\widehat{CP}$ compared to when comparing *CP* and *HP*. The interesting observation here is that both positive and negative *RTI* values exist which confirms the incomparability of *CP* and $\widehat{CP}$ previously shown by the example in Section 5.2. In conclusion, overall with the given system configuration in these experiments as well, *CP* introduces less delays to tasks compared to $\widehat{CP}$.

8.2.3 Evaluation Results of $\widehat{CP}$ versus *HP*

Figure 7 shows that $\widehat{CP}$ improves response time of tasks up to 90% compared to *HP*. Figure 7.(a) illustrates that by increasing the number of cores $\widehat{CP}$ outperforms *HP* more, in terms of response time improvement. The reason is that for tasks of range *A*, $spin_{P_k,q} = 0$ under $\widehat{CP}$ and not under *HP*. The interesting observation here is that there are no negative *RTI* values in any of the related graphs meaning that response times cannot be improved under the *HP* compared to $\widehat{CP}$ which confirms dominance of $\widehat{CP}$ compared to *HP* proven by Lemma 33.

8.3 Schedulability Results

In the second type of experiments, the schedulability under *HP*, *CP* and $\widehat{CP}$ is investigated. The results show that in general from the schedulable systems, a higher percentage are schedulable under *CP* compared to the other two protocols, i.e., $PS_{(CP)} = 99.6\%$, $PS_{(\widehat{CP})} = 76.2\%$ and

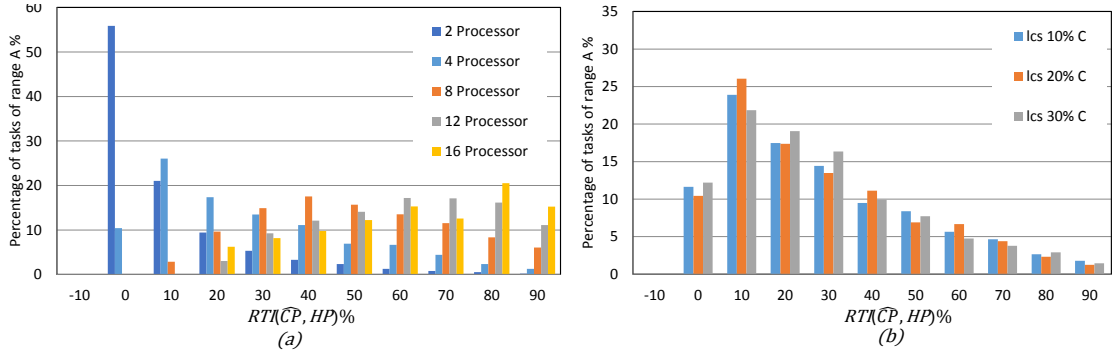

Figure 7 RTI for $\widehat{CP}$ versus HP for (a) different number of processors m, and (b) different values of lcs.

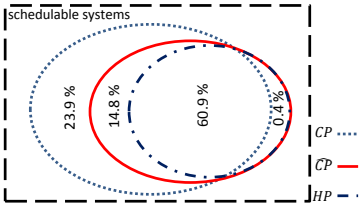

Figure 8 Schedulability percentage under HP, CP and $\widehat{CP}$.

$PS_{(HP)} = 61.4\%$. The results show that most of the schedulable systems were schedulable under all three protocols, i.e., $PS_{(CP \wedge \widehat{CP} \wedge HP)} = 60.9\%$. Moreover, this results also confirms that $\widehat{CP}$ dominates HP, i.e., all systems that were schedulable under HP were also schedulable under $\widehat{CP}$, however a percentage of tasks were only schedulable under $\widehat{CP}$ and not under HP which is presented by $PS_{(\widehat{CP}) \wedge \neg HP} = 14.8\%$. These schedulability results have also been illustrated in Figure 8. Note that the values in the graph, illustrate the schedulability of area in which the value is located.

9 Conclusion and Future Work

In this paper, we investigated spin-based protocols for resource and jitter constrained embedded multi-core platforms with the aim to improve the cost-efficiency and quality of control as well as schedulability performance. We have focused on fixed-priority partitioned scheduling which is the industry's preferred scheduling approach. For such systems, non-preemptive spin-based protocols have shown a good performance in improving the systems costs by offering use of one shared stack for running all tasks residing on a core. However, they have shown a poor efficiency in preserving the control and schedulability quality of those systems. To address these aspects, we have investigated preemptive spin-based protocols which give a better promise to pertain all these factors. Further, we showed that the selection of priority upon which a task spins is significantly important since it affects the blocking duration and hence the response time and response jitter of tasks. We have presented spin-lock priorities for preemptive-spin-based protocols for which the response time of tasks and hence the corresponding response jitters are decreased compared to when using the non-preemptive spin-based protocol.

We focused on spin-based protocols where a fixed spin-lock priority is used for spinning of any task on the core in combination with FIFO-ordering policy where under a classical technique tasks are kept in the queue upon preemption. From this type we focused on a special range that

offer attractive properties where spinning occurs at a priority at least the highest ceiling of any global resource which is the spin-lock priority of an existing spin-based protocol *CP*. Selecting from this range keeps the remote blocking bound as well as the resource queue size confined to a factor that is the number of cores in the system, similar to non-preemptive spin-based protocols. It also guarantees that the number of some resources such as stack used per core is not increased considerably compared to the non-preemptive spin-based protocols. In this paper we introduced a special spin-based protocol from this range, called $\widehat{CP}$ where we showed that it dominates the traditional non-preemptive spin-based protocol *HP*, and all spin-based protocols that use a spin-lock priority between those used by these two. Further, we showed that $\widehat{CP}$ and *CP* are incomparable, thus we have provided the blocking analysis for the considered range of spin-locks in order to enable the comparative evaluation of these incomparable protocols. The new analysis turns out to give tighter blocking bounds than those previously presented for the *CP* protocol.

Finally, we showed that if a task set is unschedulable under both *CP* and $\widehat{CP}$ on a processor, there may exist a spin-based protocol that uses a spin-lock priority in between of those used by *CP* and $\widehat{CP}$ which can make the task set schedulable. The complexity of finding such spin-based protocol, if any, is linear and can be a small value. The experimental results showed that, in general, *CP* can provide better schedulability results compared to *HP* and $\widehat{CP}$. Moreover, the results showed that although $\widehat{CP}$ and *CP* are incomparable, under specific system configurations tasks can obtain up to 70% improvement in their response times under *CP* compared to $\widehat{CP}$. Similarly, tasks can gain up to 90% improvements in their response times under *CP* and $\widehat{CP}$ compared to *HP*. It can be viewed from the evaluation results that in general, more tasks can have shorter response times under *CP* than under *HP* and $\widehat{CP}$.

Towards optimizing the spin-based protocols for tasks, we would like to look at the following steps: optimizing the spin-lock priority *(i)* per processor, *(ii)* per task, *(iii)* per resource and (iv) per resource access. In this paper we have focused on step *(i)* for a specific range of spin-based protocols. We leave the later steps as future work.

References

1 Sara Afshar, Moris Behnam, Reinder J. Bril, and Thomas Nolte. Flexible spin-lock model for resource sharing in multiprocessor real-time systems. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems, SIES 2014, Pisa, Italy, June 18-20, 2014*, pages 41–51. IEEE, 2014. doi:10.1109/SIES.2014.6871185.

2 Sara Afshar, Moris Behnam, Reinder J. Bril, and Thomas Nolte. On per processor spin-lock priority for partitioned multiprocessor real-time systems. Technical report, Mälardalen Real-Time Research Centre, Mälardalen University, 2014. URL: http://www.es.mdh.se/publications/3766-.

3 James H. Anderson, Rohit Jain, and Kevin Jeffay. Efficient object sharing in quantum-based real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998*, pages 346–355. IEEE Computer Society, 1998. doi:10.1109/REAL.1998.739768.

4 Neil C. Audsley, Alan Burns, Mike M. Richardson, Ken Tindell, and Andy J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993. doi:10.1049/sej.1993.0034.

5 AUTOSAR release 4.1, specification of operating system, 2013. URL: http://www.autosar.org.

6 Theodore P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991. doi:10.1007/BF00365393.

7 Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005. doi:10.1007/s11241-005-0507-9.

8 Aaron Block, Hennadiy Leontyev, Björn B. Brandenburg, and James H. Anderson. A flexible real-time locking protocol for multiprocessors. In *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), 21-24 August 2007, Daegu, Korea*, pages 47–56. IEEE Computer Society, 2007. doi:10.1109/RTCSA.2007.8.

9 Björn B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-time Operating Systems*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 2011. AAI3502550.

10 Björn B. Brandenburg and James H. Anderson. An implementation of the pcp, srp, d-pcp, m-pcp, and FMLP real-time synchronization protocols in litmus$^{\mathrm{rt}}$. In *The Fourteenth IEEE Internationl Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2008, Kaohisung, Taiwan, 25-27 August 2008, Proceedings*,

pages 185–194. IEEE Computer Society, 2008. doi:10.1109/RTCSA.2008.13.

11 Reinder J. Bril, Elisabeth F. M. Steffens, and Wim F. J. Verhaegh. Best-case response times and jitter analysis of real-time tasks. *J. Scheduling*, 7(2):133–147, 2004. doi:10.1023/B:JOSH.0000014069.63823.e7.

12 Alan Burns and Andy J. Wellings. A schedulability compatible multiprocessor resource sharing protocol - mrsp. In *25th Euromicro Conference on Real-Time Systems, ECRTS 2013, Paris, France, July 9-12, 2013*, pages 282–291. IEEE Computer Society, 2013. doi:10.1109/ECRTS.2013.37.

13 Travis S. Craig. Queuing spin lock algorithms to support timing predictability. In *Proceedings of the Real-Time Systems Symposium. Raleigh-Durham, NC, December 1993*, pages 148–157. IEEE Computer Society, 1993. doi:10.1109/REAL.1993.393505.

14 Robert I. Davis and Marko Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*, pages 39–50. IEEE Computer Society, 2012. doi:10.1109/RTSS.2012.57.

15 Robert I. Davis and Alan Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, 2011. doi:10.1145/1978802.1978814.

16 UmaMaheswari C. Devi, Hennadiy Leontyev, and James H. Anderson. Efficient synchronization under global EDF scheduling on multiprocessors. In *18th Euromicro Conference on Real-Time Systems, ECRTS'06, 5-7 July 2006, Dresden, Germany, Proceedings*, pages 75–84. IEEE Computer Society, 2006. doi:10.1109/ECRTS.2006.10.

17 Dario Faggioli, Giuseppe Lipari, and Tommaso Cucinotta. The multiprocessor bandwidth inheritance protocol. In *22nd Euromicro Conference on Real-Time Systems, ECRTS 2010, Brussels, Belgium, July 6-9, 2010*, pages 90–99. IEEE Computer Society, 2010. doi:10.1109/ECRTS.2010.19.

18 Paolo Gai, Giuseppe Lipari, and Marco Di Natale. 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 (RTSS 2001), London, UK, 2-6 December 2001*, pages 73–83. IEEE Computer Society, 2001. doi:10.1109/REAL.2001.990598.

19 Paolo Gai, Giuseppe Lipari, and Marco Di Natale. Stack size minimization for embedded real-time systems-on-a-chip. *Design Autom. for Emb. Sys.*, 7(1-2):53–87, 2002. doi:10.1023/A:1019795414875.

20 Paolo Gai, Marco Di Natale, Giuseppe Lipari, Alberto Ferrari, Claudio Gabellini, and Paolo Marceca. A comparison of MPCP and MSRP when sharing resources in the janus multiple-processor on a chip platform. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2003), May 27-30, 2003, Toronto, Canada*, page 189. IEEE Computer Society, 2003. doi:10.1109/RTTAS.2003.1203051.

21 Theodore Johnson and Krishna Harathi. A prioritized multiprocessor spin lock. *IEEE Trans. Parallel Distrib. Syst.*, 8(9):926–933, 1997. doi:10.1109/71.615438.

22 Leonidas I. Kontothanassis, Robert W. Wisniewski, and Michael L. Scott. Scheduler-conscious synchronization. *ACM Trans. Comput. Syst.*, 15(1):3–40, 1997. doi:10.1145/244764.244765.

23 John M. Mellor-Crummey and Michael L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. Comput. Syst.*, 9(1):21–65, 1991. doi:10.1145/103727.103729.

24 Ragunathan Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

25 Ola Redell and Martin Sanfridson. Exact best-case response time analysis of fixed priority scheduled tasks. In *14th Euromicro Conference on Real-Time Systems (ECRTS 2002), 19-21 June 2002, Vienna, Austria, Proceedings*, pages 165–172. IEEE Computer Society, 2002. doi:10.1109/EMRTS.2002.1019196.

26 Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Computers*, 39(9):1175–1185, 1990. doi:10.1109/12.57058.

27 H. Takada and K. Sakamura. Predictable spin lock algorithms with preemption. In *11th IEEE Workshop on Real-Time Operating Systems and Software (RTOSS'94)*, pages 2–6, 1994. doi:10.1109/RTOSS.1994.292571.

28 Hideyuki Takada and Ken Sakamura. A novel approach to multiprogrammed multiprocessor synchronization for real-time kernel. In *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97), December 3-5, 1997, San Francisco, CA, USA*, pages 134–143. IEEE Computer Society, 1997. doi:10.1109/REAL.1997.641276.

29 Alexander Wieder and Björn B. Brandenburg. On spin locks in AUTOSAR: blocking analysis of fifo, unordered, and priority-ordered spin locks. In *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013, Vancouver, BC, Canada, December 3-6, 2013*, pages 45–56. IEEE Computer Society, 2013. doi:10.1109/RTSS.2013.13.

A Table of Notations

Table 1 Table of notations

Notations	Description
P_k	processor k
τ_i	task i
C_i	worst-case execution time of τ_i
$\acute{C}_i$	inflated execution time of τ_i
T_i	minimum inter-arrival time of τ_i
D_i	relative deadline of τ_i
π_i	priority of τ_i
$\mathcal{T}_{P_k}$	set of tasks allocated to processor P_k
R_q	resource q
$\mathcal{R}^{\mathrm{L}}_{P_k}$	set of local resources accessed by tasks on P_k
$\mathcal{R}^{\mathrm{G}}_{P_k}$	set of global resources accessed by tasks on P_k
$\mathcal{RS}^{\mathrm{L}}_i$	set of local resources accessed by jobs of τ_i
$\mathcal{RS}^{\mathrm{G}}_i$	set of global resources accessed by jobs of τ_i
$Cs_{i,q}$	worst-case execution time in all τ_i's requests on R_q
$n^{\mathrm{G}}_{i,q}$	maximum number of requests of a job of τ_i for a global resource R_q
$\mathcal{T}_{P_k,q}$	set of tasks on P_k that request R_q
WR_i	worst-case response time of τ_i
BR_i	best-case response time of τ_i
RJ_i	response jitter of τ_i
$\pi^{\max}_{P_k}$	highest priority level on P_k
$ceil_{P_k}(R_l)$	ceiling of R_l on P_k
$\pi^{\mathrm{L}}_{P_k}$	highest local ceiling of any local resource on P_k
π^{G}_{Pk}	highest local ceiling of any global resource on P_k
$\pi^{\mathrm{LG}}_{P_k}$	highest local ceiling of any (local or global) resource on P_k
$\pi^{\mathrm{spin}}_{P_k}$	an arbitrary fixed spin-lock priority for any task on P_k
$\pi^{\mathrm{spin}_\sigma}_{P_k}$	spin-lock priority of spin-based protocol σ of P_k
LBL	local blocking due to local resources
LBG	local blocking due to global resources
$spin_{P_k,q}$	maximum remote blocking (i.e. spin-lock time) for any task on P_k to acquire R_q
$spin_i$	maximum total remote blocking (i.e. spin-lock time) for τ_i to acquire all its resources
$R^{\mathrm{spin}}_{P_k}$	the "virtual" local spin resource on P_k
B^{L}_i	LBL imposed to a task τ_i under an arbitrary spin-based protocol
B^{G}_i	LBG imposed to a task τ_i under an arbitrary spin-based protocol
B_i	total pi-blocking imposed to a task τ_i under an arbitrary spin-based protocol
$B_i(\pi^{\mathrm{spin}}_{P_k})$	total blocking to a task $\tau_i \in \mathcal{T}_{P_k}$ under a spin-based protocol with spin-lock priority $\pi^{\mathrm{spin}}_{P_k}$