Per Processor Spin-Lock Priority for Partitioned Multiprocessor Real-Time Systems

Sara Afshar[1], Moris Behnam[1], Reinder J. Bril[1,2], Thomas Nolte[1]
[1]Mälardalen University, Västerås, Sweden
[2]Technische Universiteit Eindhoven, Eindhoven, The Netherlands
Email: {sara.afshar, moris.behnam, reinder.j.bril, thomas.nolte}@mdh.se, r.j.bril@tue.nl

ABSTRACT

Two traditional approaches exist for a task that is blocked on a global resource; a task either performs a non-preemptive busy wait, i.e., spins, or suspends and releases the processor. Previously, we have shown that both approaches can be viewed as spinning either at the highest priority (*HP*) or at the lowest priority on the processor (*LP*), respectively. Based on this view, previously we have generalized a task's blocking behavioral model, as spinning at any arbitrary priority level. In this paper, we focus on a particular class of spin-lock protocols from the introduced flexible spin-lock model where spinning is performed at a priority equal to or higher than the highest local ceiling of the global resources accessed on a processor referred to as *CP* spin-lock approach. In this paper, we assume that all tasks of a specific processor are spinning on the same priority level. Given this class and assumption, we show that there exists a spin-lock protocol in this range that dominates the classic spin-lock protocol which tasks spin on highest priority level (HP). However we show that this new approach is incomparable with the *CP* spin-lock approach. Moreover, we show that there may exist an intermediate spin-lock approach between the priority used by *CP* spin-lock approach and the new introduced spin-lock approach that can make a task set schedulable when those two cannot. We provide an extensive evaluation results comparing the *HP*, *CP* and the new proposed approach.

1. INTRODUCTION

The interest from the industry to replace uni-processors with multi-core[1] platforms raises a demand for investigating efficient techniques for such platforms.

In real-time embedded systems, tasks may share resources with each other that may lead to variations in response time of the tasks. For industrial setups that have critical tasks in terms of timing constraints, such as control tasks, not only meeting the deadlines is important but also how long it takes for tasks to finish and the imposed jitter to the tasks can become an important issue. Long response times and/or induced jitter to control tasks may introduce delays in control loops that can decline the control performance and in the worst case destabilize the system. Therefore, it is of great significance to take advantage of techniques that can minimize such induced jitters. Considering resource sharing as a source for such induced jitters in such systems, efficient resource handling techniques can benefit tasks in this context.

In multi-core platforms, upon a request to a global resource, the resource may be in use by a task on another processor. Currently, two approaches exist when such a request for a global resource cannot be satisfied, a task either spins or suspends. Under the spin-based approach, the task performs a non-preemptive busy wait (also called spin) on the highest priority level on the core, which we refer to as *HP* (highest priority level) spin-lock approach. Under the suspension-based approach, a task is suspended.

[1]In this text we will use the terms core and processor interchangeably.

In previous work [1], we showed that this latter approach can also be viewed as spinning at the lowest priority level on the core. We refer to this approach as *LP* (lowest priority level) spin-lock approach, or simply *LP*. Based on such a view, we generalized a task's blocking behavior while spinning, by selecting any arbitrary priority level in the range of *LP* to HP. We referred to this model as *flexible spin-lock model.* In [1], we also introduced two specific spin-lock approaches where the task that is blocked on a (global) resource (i.e. has requested the resource but is not granted since the resource is not available at the moment of request) uses two priority levels that are within the range of HP and *LP* spinning priority levels. These two spin-lock approaches are called *CP* (ceiling priority level) and *OP* (own priority level) spin-lock approaches. The *CP* approach uses the highest local ceiling of global resources on a core for spinning. The *OP* approach uses the task's own priority level for spinning. One of our conclusions in [1] was that *CP* and HP are incomparable, i.e, if some task sets are schedulable under *CP* but not HP and vice versa, therefore neither approach dominates the other.

In this paper, we focus on a particular class of spin-lock protocols from the introduced flexible spin-lock model [1], where spinning is performed at a priority level equal to or higher than that used by *CP*. A main advantage of this class of spin-lock protocols is that at most one task on a processor can either have a pending global resource request or access a global resource (see Lemma 12 in [1]). As a consequence, a task that spinning on a global resource will have to wait for at most $m-1$ accesses of tasks on remote cores, where m is the number of cores. Similarly, the length of any global resource queue is at most $m-1$. Using priority levels other than this range from the whole spectrum may result in a higher number of pending requests for a global resource on a processor. We therefore leave the study of the rest of the spectrum for optimal spin-lock priorities as future work. In this paper, we focus on partitioned scheduling systems considering FIFO-based queues for global resources.

Further, in this work, we assume that all tasks on a specific processor use the same spin-lock priority. Given this class of spin-lock protocols and this assumption, we introduce a new spin-lock approach which we call $\widehat{CP}$ and uses the highest ceiling of any resource (either local or global) for spinning. We show that $\widehat{CP}$ outperforms the classic HP spin-lock approach. Moreover, we show that the CP and $\widehat{CP}$ approaches are incomparable. We show that there may exist a spin-lock approach with spinning priority level between the priority level used by CP and $\widehat{CP}$ that can make a task set schedulable which is unschedulable under both CP and $\widehat{CP}$.

Contributions. In this paper we introduce a new spin-lock approach ($\widehat{CP}$) that dominates the classic HP spin-lock approach. Moreover, we provide a tighter analysis for the CP approach. Further, we provide unified blocking terms for the range of spin-lock priorities [CP, HP]. We show by means of illustrative examples that CP and $\widehat{CP}$ are incomparable. Further, we show that there may exist an intermediate spin-lock approach within the range of spin-lock priorities for CP, $\widehat{CP}$ that can make a task set on a processor schedulable where CP and $\widehat{CP}$ cannot. Finally, we perform experimental evaluations to compare the three HP, CP and $\widehat{CP}$ spin-lock approaches where we measure the improvement in response time of tasks under each approach versus the other.

2. RELATED WORKS

A non-exhaustive amount of work has been done for both variants of spin-based and suspension-based of lock-based resource sharing synchronization protocols. In the following we will briefly present the most related synchronization protocols used in multiprocessor platforms.

The Multiprocessor Stack Resource Policy (MSRP) was introduced in [13] for partitioned platforms. MSRP is an extension of Stack Resource Policy (SRP) [4] for multiprocessors and is a spin-based approach. Global resource waiting queues are FIFO-based.

The Multiprocessor Bandwidth Inheritance (M-BWI) protocol, an extension of the Bandwidth Inheritance (BWI) protocol, has been presented for global scheduling in [12]. However the M-BWI protocol is neutral to the underlying scheduling approach, since it can be implemented in both global and partitioned scheduling systems. M-BWI protocol is a spin-based synchronization protocol that can be used in open systems where tasks can dynamically be added or removed. The resource queues used in M-BWI are FIFO-based.

The Flexible Multiprocessor Locking Protocol (FMLP) introduced in [6] combines both spin-based and suspension-based approaches. Tasks spin for short resources when they are blocked on a global resource, whereas they are suspended for long resource requests. The resource classification for short and long is user defined under this approach. FMLP uses FIFO-based global resource queues and has been introduced for both partitioned and global scheduled systems. The partitioned FMLP, was later extended for fixed-priority scheduling in [7].

Distributed Priority Ceiling Protocol (DPCP) is a suspension-based synchronization protocol introduced in [20, 19] and has been developed for partitioned scheduling and the global resource waiting queues are priority-based. The Multiprocessor Priority Ceiling Protocol (MPCP) has been introduced for partitioned systems [20, 18]. MPCP a variant of the Priority Ceiling Protocol (PCP) [21] extended for multiprocessor partitioned scheduling platforms,is a suspension-based protocol. The global resource queues used in MPCP are priority-based. A recent variant of MPCP [15] developed for partitioned scheduling systems introduces some busy waiting that can decrease the blocking duration of higher priority tasks.

The $O(m)$ Locking Protocol (OMLP), which is a suspension-based synchronization protocol, has been proposed in [8], for both partitioned and global scheduling. In a suspension-oblivious protocol, suspensions are accounted in the execution time of tasks, which means that suspended jobs are assumed to occupy the processor. Brandenburg and Anderson showed that priority-based queues can introduce starvation for lower priority tasks [8]. As a result OMLP uses a hybrid queue structure consisting of a FIFO-based queue of a specified length m along with a priority-based queue. This design scheme causes OMLP to be confined to a fixed factor of blocking (*asymptotically optimal* [8]) as an advantage.

Multiprocessor Synchronization Protocol for Open Systems (MSOS), is another suspension-based synchronization protocol which has been developed for compositional independently-developed real-time applications [17]. The global resource waiting queues are FIFO-based, however, for the resource waiting queues within each application, both variants of FIFO-based and Priority-based queue have been investigated. MSOS was extended for priority-based applications later [2], where it is shown that it could improve the schedulability performance.

A recent work has investigated different queuing policies for spin lock multiprocessor systems [22]. Such investigation is out of the scope of this paper, and we hafocous on a fixed set-up for queuing policy, being FIFO-based global resource queues.

All aforementioned synchronization protocols have been introduced for spinning on highest and/or lowest priority level. In [1], we have introduced a flexible spin-lock model which introduces the possibility of using any arbitrary priority-level for spin-locking. Moreover, we have introduced two new spin-lock approaches that can have better schedulability performance compared to spin-lock approaches that use the highest priority level on a processor for spinning and suspend-based approaches for a specific class of tasks on a processor. In this paper, we investigate further spin-lock priorities for tasks on a processor. Note that, we select one unique spin-lock priority that is used by all tasks on a specific processor.

3. SYSTEM MODEL

Our system consists of m identical processors where a set of n sporadic tasks execute on. The scheduling policy used in this paper is 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 a tuple $< C_i, T_i >$ and consists of an infinite sequence of jobs. C_i denotes the worst-case execution time of task τ_i and T_i denotes the minimum inter-arrival time of τ_i. We assume implicit deadlines tasks, i.e. the deadline of a task denoted by D_i is equal to T_i. The priority of the task τ_i is denoted by ρ_i. U_i denotes the utilization of a task τ_i. We assume that a task τ_i has a priority higher than that of a task τ_j, if $i > j$ (i.e., $\rho_i > \rho_j$). We assume priorities are unique 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 more than one processor. The sets of local and global resources which are accessed by tasks on a processor P_k are denoted by $\mathcal{R}_{P_k}^{\mathrm{L}}$ and $\mathcal{R}_{P_k}^{\mathrm{G}}$, respectively. Similarly, we denote the set of local and global resources that are accessed by jobs of a task τ_i as $\mathcal{RS}_i^{\mathrm{L}}$ and $\mathcal{RS}_i^{\mathrm{G}}$, 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, $n_{i,q}^{\mathrm{G}}$ 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 requests is not the focus of this paper. Moreover, as it will be presented later in this section (Section 3.2), queues used for global resources are FIFO-based and global critical sections (*gcs*es i.e. the sections of a task that uses global resources) are non-preemptive.

Moreover, we categorize the delay that is introduced to any task due to resource sharing to two different blocking notions: (i) priority inversion blocking (pi-blocking) [19, 21] which is when a lower priority job is scheduled while a higher priority job on the same core is pending and (ii) acquisition delay that is incurred to a job of a task that has to wait to obtain a resource that is locked on a remote processor (i.e., a processor other than the task's allocated processor). The first term is also referred to as *Local Blocking* since it is incurred to a task by tasks on the same processor and the later terms is also referred to as *Remote Blocking* since it is due to tasks on a different processor. When a task is experiencing remote blocking it is said that the task is blocked on the resource. In this paper, we denote the B_i term as the total pi-blocking incurred to a task τ_i and we exclude the remote blocking from this term. Based on our flexible spin-lock model [1], a task spins when it is remotely blocked on a resource, so we will look at this term separately in the analysis (see Definition 9).

3.1 General definitions

Below, we present a set of definitions easing the presentation of the proofs.

DEFINITION 1. *The highest priority level on a processor P_k is denoted by $\rho^{\max}_{P_k}$ as follows:*

$$\rho^{\max}_{P_k} = \max_{\forall \tau_i \in \mathcal{T}_{P_k}} \{\rho_i\} + 1. \tag{1}$$

DEFINITION 2. *Local resource sharing protocols similar to SRP assign a ceiling to any local resource $R_l \in \mathcal{R}^{\mathrm{L}}_{P_k}$, where $ceil_{P_k}(R_l) = \max\{\rho_i | \, R_l \in \mathcal{RS}^{\mathrm{L}}_i\}$. [4]*

DEFINITION 3. *We denote the highest local ceiling of any global resource on a processor P_k as rc^{G}_{Pk}, where $rc^{\mathrm{G}}_{P_k} = \max\{\rho_i | \tau_i \in \mathcal{T}_{P_k} \wedge \mathcal{RS}^{\mathrm{G}}_i \neq \emptyset\}$.*

DEFINITION 4. *We denote the highest local ceiling of any resource on a processor (either local or global) P_k as rc^{LG}_{Pk}, where $rc^{\mathrm{LG}}_{P_k} = \max\{\rho_i | \tau_i \in \mathcal{T}_{P_k} \wedge \mathcal{RS}^{\mathrm{G}}_i \cup \mathcal{RS}^{\mathrm{L}}_i \neq \emptyset\}$.*

DEFINITION 5. *When a task spins on a processor to acquire a resource, its priority level might change during spinning to a new priority level due to the spin-lock approach that is used. We denote the $\rho^{\mathrm{spin}}_{P_k}$ as an arbitrary spin-lock priority level that is used for every task when spins on a processor P_k. Note that, in this paper, we consider $\rho^{\mathrm{spin}}_{P_k}$ where $rc^{\mathrm{G}}_{P_k} \leq \rho^{\mathrm{spin}}_{P_k} \leq rc^{\mathrm{LG}}_{P_k}$.*

DEFINITION 6. *We refer to pi-blocking that is incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ by lower priority tasks on the same core requesting local resources as* local blocking due to local resources *(*LBL*).*

DEFINITION 7. *We refer to pi-blocking that is incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ by lower priority tasks on the same core that request global resources as* local blocking due to global resources *(*LBG*).*

DEFINITION 8. *The maximum spin lock time that any task on a processor P_k may incur due to waiting for a global resource $R_q \in \mathcal{R}^{\mathrm{G}}_{P_k}$, to be released by remote processors is denoted by $spin_{P_k,q}$.*

DEFINITION 9. *The maximum spin lock time that a task τ_i may incur due to waiting for all its global resource requests is denoted by $spin_i$.*

DEFINITION 10. *Spinning approach A is said to* dominate *spinning approach B, if all of the task sets that are schedulable according to spinning approach B are also schedulable according to spinning approach A, and task sets exist that are schedulable according to A, but not according to B. (Similar to [11])*

DEFINITION 11. *Spinning approach A and B are* incomparable*, if there exist task sets that are schedulable according to spinning approach A, but not according to spinning approach B and vice versa. (Similar to [11])*

3.2 Resource sharing rules

This section presents the resource sharing rules based on the flexible spinning model presented in [1] for all synchronization protocols with spin-lock priorities in the range [*CP*, *HP*]. 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. However, the priority level on which the task spins is fixed for a core.

RULE 1. *Local resources are handled by means of SRP uniprocessor synchronization protocol.*

RULE 2. *For each global resource, a FIFO-based queue is used to enqueue the tasks waiting for the related resource.*

RULE 3. *Whenever a task τ_i on a core P_k requests a global resource that is used by tasks on other processors, it places its request in the related resource queue and performs a busy wait. The task will spin with a priority level $\rho^{\mathrm{spin}}_{P_k}$, where $rc^{\mathrm{G}}_{P_k} \leq \rho^{\mathrm{spin}}_{P_k} \leq \rho^{\max}_{P_k}$.*

RULE 4. *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 $\rho^{\max}_{P_k}$, i.e., becomes non-preemptive.*

RULE 5. *The priority of the task is changed to its original priority as soon as it finishes the global critical section and it becomes preemptable again.*

RULE 6. *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.*

Note that, conceptually, we view spinning as accessing a local *pseudo resource* [14] with a local resource ceiling equal to the spinning priority. In this way, we can treat spinning in the same way as a regular local resource access in ceiling-based resource-access protocols.

4. EXISTING APPROACHES RECAP

In this section we recapitulate existing spin-lock approaches and some lemmas and equations which also have been presented in [1]. We briefly present the commonality among all spin-lock protocols presented in this paper in Section 4.2 (i.e. the spin-lock protocols which use spin priority levels equal to or higher than rc^{G}_{Pk}). Next in Sections 4.3 and 4.4, we briefly recapitulate the blocking terms under *HP* and *CP* [1].

4.1 Recap of useful lemmas

Here we repeat some lemmas presented in [1] that also hold under $\widehat{CP}$ and will be used in later sections.

LEMMA 1. *A lower priority task τ_j cannot issue any resource request after a higher priority task τ_i arrives, where in case of spinning of τ_i, it spins with a priority higher than or equal the priority of the task itself. (Lemma 2 in [1]).*

LEMMA 2. *A lower priority task τ_j can incur pi-blocking to a higher priority task τ_i at most once, where in case of spinning of τ_i, it spins with a priority higher than or equal the priority of the task itself. (Lemma 3 in [1]).*

LEMMA 3. *For a task τ_i at most one LBL (recall Definition 6) can be incurred from lower priority tasks when SRP is used. (Lemma 4 in [1]).*

Note that, the above lemma is in fact a property of SRP [4].

LEMMA 4. *At most one task at a time can have a pending request on a global resource on a processor P_k when $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$. (Lemma 12 in [1]).*

Note that Lemma 4 holds for every spin-lock approach with a priority in the range [CP, HP], since for all these approaches $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$.

4.2 Commonality of all spin-lock approaches

In this section we present the commonalities for spin-lock approaches where $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$.

LEMMA 5. *Maximum LBL (recall Definition 7) that is incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ from lower priority tasks on the same processor, when $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$ is denoted as B^{L}_i and is upper bounded as follows.*

$$B^{\text{L}}_i = \max_{\substack{\forall j,l:\rho_j<\rho_i \wedge \tau_i,\tau_j \in \tau_{P_k} \\ \wedge R_l \in \mathcal{RS}^{\text{L}}_j \wedge \rho_i \leq ceil_{P_k}(R_l)}} \{Cs_{j,l}\}. \tag{2}$$

PROOF. In order to a task τ_i experiences LBL blocking (Definition 6) from a lower priority task τ_j, the lower priority task should use a local resource with ceiling higher than the priority of τ_i. Based on this and according to Lemma 3, (2) is inferred. □

LEMMA 6. *A task τ_i can experience at most one LBG (recall Definition 7) from lower priority tasks, when $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$.*

PROOF. By contradiction. We assume two tasks τ_j and τ_m with a lower priority than that of τ_i that incur LBG to τ_i. According to Lemma 1, τ_j and τ_m both can issue their requests only before τ_i arrives at time t_1 and their requests can contribute in delaying τ_i only if the resource is not granted before time t_1 to them. Let us assume τ_j has arrived earlier than τ_m and issue its request and start spinning since the resource was not granted. Now let after a while (before time t_1) task τ_m arrives. In order to τ_m issues its request, it must preempt τ_j while spinning. In order to τ_m preempts spinning, thus $\rho_m > \rho^{\text{spin}}_{P_k}$. Since $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$, this concludes that $\rho_m > rc^{\text{G}}_{P_k}$. By definition (Definition 4), τ_m does not use any global resource which contradicts our first assumption. This finishes the proof. □

When a task requests a global resource it may happen that the resource is in use by another task on another processor. Therefore, from the time that a task τ_i requests a global resource R_q which is not available at the moment of the request, it spins with priority $\rho^{\text{spin}}_{P_k}$ according to Rule 3 until it locks the resource. The maximum time that a task may spin, depends on the maximum time that any other processor can lock R_q. According to Lemma 4 multiple tasks cannot request and be granted to global resources. Therefore, one task only from each processor can request R_q at any time. Hence, from any P_k's remote processors, at most one longest gcs on R_q can delay any request on P_k for R_q. As a result, (i) the length of every FIFO-queue can be bounded by $m-1$ (m the number of processors in the platform), and (ii) every task has to wait for at most $m-1$ accesses to a global resource upon a request to that resource.

Followed by the above discussion, the maximum spinning time for a task τ_i which is $spin_{P_k,q}$ (recall Definition 8) is given by:

$$spin_{P_k,q} = \sum_{\forall P_r \neq P_k} \max_{\forall \tau_j \in \mathcal{T}_{P_r,q}} \{Cs_{j,q}\}. \tag{3}$$

$spin_i$ by definition (recall Definition 9) is calculated as follows:

$$spin_i = \sum_{\forall q: R_q \in \mathcal{RS}^{\text{G}}_i \wedge \tau_i \in \mathcal{T}_{P_k}} \{n^{\text{G}}_{i,q} \times spin_{P_k,q}\}. \tag{4}$$

The critical section length of a task and consequently its execution time will be increased by spinning. Therefore, the actual execution time of a task τ_i is denoted by $\acute{C}_i$ and is calculated as follows:

$$\acute{C}_i = C_i + spin_i. \tag{5}$$

4.3 *HP* spin-lock approach

Under the HP spin-lock approach $\rho^{\text{spin}}_{P_k} = \rho^{\max}_{P_k}$ (recall Definition 1 and Rule 6). This means that, from the time that a task requests a global resource it non-preemptively performs a busy wait if the resource is in use by another processor, until it locks the resource. This leads to tasks be non-preemptive while spinning. Similar spin-lock approach is used in MSRP [14] and FMLP for short resources, [6] and [7]. Below we briefly present the blocking terms that occur under this protocol which has been introduced in [14].

The Maximum LBL that is incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ under the HP spin-lock approach is denoted as $B^{\text{L}_{\text{HP}}}_i$.

$$B^{\text{L}_{\text{HP}}}_i = B^{\text{L}}_i. \tag{6}$$

where B^{L}_i is calculated according to Lemma 5.

The maximum LBG that is incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ under the HP spin-lock approach is denoted as $B^{\text{G}_{\text{HP}}}_i$ and is upper bounded as follows:

$$B^{\text{G}_{\text{HP}}}_i = \max_{\substack{\forall j,q:\rho_j<\rho_i \wedge \tau_i,\tau_j \in \mathcal{T}_{P_k} \\ \wedge R_q \in \mathcal{RS}^{\text{G}}_j}} \{Cs_{j,q} + spin_{P_k,q}\}. \tag{7}$$

The total pi-blocking (local blocking) that is incurred to a task τ_i under the HP spin-lock approach is denoted as B^{HP}_i and is calculated as follows:

$$B^{\text{HP}}_i = \max(B^{\text{L}_{\text{HP}}}_i, B^{\text{G}_{\text{HP}}}_i). \tag{8}$$

Note that, the delay that is incurred to a task τ_i due to waiting for locking global resources is considered as an inflation in task's worst-case execution time (5).

4.4 *CP* spin-lock approach

Under the *CP* spin-lock approach [1], $\rho^{\text{spin}}_{P_k} = rc^{\text{G}}_{P_k}$ (recall Definition 3). Hence, the spinning can only be preempted by higher priority tasks that do not use any global resources.

The maximum LBL that is incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ under the *CP* spin-lock approach is denoted as $B_i^{\text{L}_{\text{CP}}}$.

$$B_i^{\text{L}_{\text{CP}}} = B_i^{\text{L}}. \tag{9}$$

where B_i^{L} is calculated according to Lemma 5.

The maximum LBG that is incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ under the *CP* spin-lock approach is denoted as $B_i^{\text{G}_{\text{CP}}}$ and is upper bounded as follows:

$$B_i^{\text{G}_{\text{CP}}} = \max_{\substack{\forall j,q: \rho_j < \rho_i \wedge \tau_i, \tau_j \in \mathcal{T}_{P_k} \\ \wedge R_q \in \mathcal{RS}_j^{\text{G}}}} \left(Cs_{j,q} + \begin{cases} spin_{P_k,q} & \text{if } \rho_i \leq rc^{\text{G}}_{P_k} \\ 0 & \text{otherwise} \end{cases} \right). \tag{10}$$

The maximum total pi-blocking incurred to a task $\tau_i \in \mathcal{T}_{P_k}$ is calculated as follows, ([1], Theorem 3):

The total pi-blocking (local blocking) that is incurred to a task τ_i under the *CP* spin-lock approach is denoted as B_i^{CP} and is calculated as follows:

$$B_i^{\text{CP}} = \begin{cases} B_i^{\text{L}_{\text{CP}}} + B_i^{\text{G}_{\text{CP}}} & \text{if } \rho_i > rc^{\text{G}}_{P_k} + 1. \\ \max(B_i^{\text{L}_{\text{CP}}}, B_i^{\text{G}_{\text{CP}}}) & \text{if } \rho_i \leq rc^{\text{G}}_{P_k} + 1. \end{cases} \tag{11}$$

Under *CP* compared to the *HP*, the highest priority tasks that do not use any (global) resources, can have a smaller worst-case response time. However, *CP* and *HP* are incomparable in general, since one can outperform the other for tasks with different priorities [1].

4.5 Worst case response time

Worst case response time for a task τ_i is denoted by WR_i and is calculated as follows:

$$WR_i = \acute{C}_i + B_i + \sum_{\substack{\forall \rho_j > \rho_i \\ \tau_i, \tau_j \in \mathcal{T}_{P_k}}} \lceil (WR_i)/T_j \rceil \acute{C}_j, \tag{12}$$

where B_i is the maximum blocking duration incurred to a task τ_i on a processor P_k. Under different spin-lock approaches the related B_i term is used, e.g., under the *HP* spin-lock approach B_i^{HP} is used for B_i.

Note that for spin-lock priority levels where $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$ the delay imposed by a higher priority task τ_j due to resource waiting time which should have been added up to WR_i in RHS of (12) [3, 16, 9] is equal to zero. This waiting time has already been considered in the execution time of the task τ_j ($\acute{C}_j$) (5).

5. TIGHTER ANALYSIS FOR *CP*

In this section first with a simple example, we show that the analysis given in [1] is pessimistic and later we provide a tighter analysis for *CP* approach by providing Lemmas 8 and 9 for such analysis.

τ_i	$Cs_{i,l}$	$Cs_{i,g}$
τ_1	5	3
τ_2	8	2
τ_3	4	-
τ_4	x	-

Table 1: Example 1: Resource access by tasks of $\mathcal{T}_{P_k}$.

Consider a task set $\mathcal{T}_{P_k} = \{\tau_4, \tau_3, \tau_2, \tau_1\}$. Let $\mathcal{R}^{\text{L}}_{P_k} = \{R_\ell\}$ and $\mathcal{R}^{\text{G}}_{P_k} = \{R_g\}$.

In this case, under *CP* spin-lock approach, the spin-priority level $\rho^{\text{spin}}_{P_k} = rc^{\text{G}}_{P_k} = 2$. A job of τ_4 can be blocked by:

- τ_3 on R_ℓ *and* either τ_2 or τ_1 on R_g, or
- either τ_2 or τ_1 on R_ℓ.

The worst-case depends on the length of the critical sections, i.e.

$$B_4^{CP} = \max(Cs_{3,\ell} + \max(Cs_{2,g}, Cs_{1,g}), \max(Cs_{2,\ell}, Cs_{1,\ell})). \tag{13}$$

In example 1, we find

$$B_4 = \max(4 + \max(2,3), \max(8,5)) = 8. \tag{14}$$

In this case, according to (11) $B_4^{CP} = B_4^{\text{L}_{\text{CP}}} + B_4^{\text{G}_{\text{CP}}} = 8 + 3 = 11$. This latter value is pessimistic which motivates us to present tighter bounds under the *CP* approach.

LEMMA 7. *A task τ_i can experience at most one pi-blocking from any lower priority task τ_j with priority $\rho_j \leq rc^{\text{G}}_{P_k}$ where $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$.*

PROOF. First we prove that any lower priority task τ_j cannot issue its resource request after τ_i arrives. To prove this we assume two scenarios: (i) $\rho_i \leq rc^{\text{G}}_{P_k}$, and (ii) $\rho_i > rc^{\text{G}}_{P_k}$. The first case is proved according to Lemma 1. The only chance that, in the second case, a lower priority task τ_j can issue a request after τ_i arrives, is if τ_i's priority changes to a lower priority level (while its waiting for a global resource) due to a spin-lock approach. However, by definition, τ_i does not use any global resource (recall Definition 3). Therefore τ_i' priority is not changed by any spin-lock approach. Thus, τ_i preempts any lower priority task when it arrives. As a result in both cases, a lower priority task can only issue a request before the higher priority task arrives.

Now let us assume two scenarios, (1) τ_j requests a global resource, and (2) τ_j requests a local resource. In the first scenario, τ_j can incur blocking to τ_i if either τ_j's requested resource is remotely held and it spins with a priority higher than the priority of τ_i or it locks the resource and is executing non-preemptively within its critical section when τ_i arrives. Obviously, if τ_j is executing non-preemptively, no other lower priority task can run so that it can issue another request. Similarly, if τ_j is spinning with $\rho^{\text{spin}}_{P_k} \geq rc^{\text{G}}_{P_k}$, no task with priority equal to or lower than $rc^{\text{G}}_{P_k}$ can preempt τ_j so that it can issue more requests.

In the second scenario, let us assume τ_j issues a request on a local resource. In order that τ_j can cause LBL to τ_i, the ceiling of the local resource that τ_j requests should be equal to or higher than ρ_i

(recall Definition 2). With such assumption, no other lower priority task than that of τ_i can preempt τ_j before completion to be able to issue more requests. Further, according to Lemma 3, a task τ_i can experience at most one LBL from any lower priority task. This implies that τ_i can be blocked by only one request of a lower priority task τ_j (either on a global resource or a local resource). This finishes the proof. □

LEMMA 8. *A task τ_i where $\rho_i \leq rc^{\mathrm{G}}_{P_k} + 1$ can experience at most one pi-blocking from any lower priority task τ_j if $\rho_j \leq rc^{\mathrm{G}}_{P_k}$, where $\rho^{\mathrm{spin}}_{P_k} \geq rc^{\mathrm{G}}_{P_k}$.*

PROOF. For a task τ_j with priority lower than that of τ_i, $\rho_j < \rho_i$. Since $\rho_i \leq rc^{\mathrm{G}}_{P_k} + 1$, then $\rho_j \leq rc^{\mathrm{G}}_{P_k}$. Thus, according to Lemma 7, τ_i may experience at most one blocking from any lower priority task. □

LEMMA 9. *A task τ_i with priority $rc^{\mathrm{G}}_{P_k} + 1 < \rho_i \leq rc^{\mathrm{LG}}_{P_k}$ can experience at most two pi-blocking if $\rho^{\mathrm{spin}}_{P_k} \geq rc^{\mathrm{G}}_{P_k}$: one LBL from any lower priority task τ_m, where $\rho_m > \rho^{\mathrm{spin}}_{P_k}$ and one LBG from any lower priority task τ_j, where $\rho_j \leq \rho^{\mathrm{spin}}_{P_k}$.*

PROOF. If $rc^{\mathrm{G}}_{P_k} + 1 < \rho_i \leq rc^{\mathrm{LG}}_{P_k}$ then task τ_i does not use any global resource (recall Definition 3). Therefore after it arrives its priority never changes (e.g. to a lower priority). Thus, it cannot be preempted by a lower priority task. Hence, τ_j can only issue a request before τ_i arrives.

According to Lemma 7, τ_i can experience at most one pi-blocking from a lower priority task τ_j with priority $\rho_j \leq rc^{\mathrm{G}}_{P_k}$. If τ_j issues a request on a local resource, according to Lemma 3, τ_i cannot experience any more LBL from any other lower priority task τ_m with priority $\rho_m > rc^{\mathrm{G}}_{P_k}$. According to Lemma 7, τ_j may request a global resource before τ_i arrives, and spins. Under this scenario, a lower priority task τ_m with priority $\rho_m > \rho^{\mathrm{spin}}_{P_k}$ can preempt τ_j during its spinning (with priority $\rho^{\mathrm{spin}}_{P_k}$). If τ_m preempts τ_j while spinning and issues a request on a local resource with ceiling higher than ρ_i just before τ_i arrives, it will delay τ_i at the time of arrival. Moreover, according to Lemma 3, τ_m can issue one request on a local resource, only. Under this scenario, when τ_j is granted its global resource (thus executes non-preemptively, Rule 4) it can further delay τ_i. As a result, τ_i can experience at most two pi-blocking, due to LBG and LBL blocking under such a scenario. □

LEMMA 10. *A task τ_i can experience at most one pi-blocking if $\rho_i > rc^{\mathrm{LG}}_{P_k}$ using a spin-lock approach where $\rho^{\mathrm{spin}}_{P_k} \geq rc^{\mathrm{G}}_{P_k}$. Moreover, this blocking is due to LBG blocking.*

PROOF. A task τ_i, by definition, does not use any local resource if $\rho_i > rc^{\mathrm{LG}}_{P_k}$ (recall Definition 4). For the same reason, no task with priority higher than that of τ_i also use any local resource. As a result, τ_i cannot experience any LBL blocking. On the other hand, according to Lemma 6, τ_i can experience at most one LBG from its lower priority tasks. This finishes the proof. □

LEMMA 11. *The maximum LBL blocking duration incurred to a task τ_i by a lower priority task τ_j on the same processor is denoted as $B^{\mathrm{L}}_{i,j}$ and is calculated as follows.*

$$B^{\mathrm{L}}_{i,j} = \max_{\substack{\forall l: R_l \in \mathcal{RS}^{\mathrm{L}}_j \\ \wedge \rho_j < \rho_i \leq ceil_{P_k}(R_l)}} \{Cs_{j,l}\}. \tag{15}$$

PROOF. It is directly inferred from SRP specification [4]. □

LEMMA 12. *The maximum LBG blocking duration incurred to a task τ_i by a lower priority task τ_j, on the same processor, using a spin-lock approach where $\rho^{\mathrm{spin}}_{P_k} \geq rc^{\mathrm{G}}_{P_k}$ is denoted as $B^{\mathrm{G}}_{i,j}(\rho^{\mathrm{spin}}_{P_k})$ and is calculated as follows.*

$$B^{\mathrm{G}}_{i,j}(\rho^{\mathrm{spin}}_{P_k}) = \max_{\substack{\forall q: R_q \in \mathcal{RS}^{\mathrm{G}}_j \\ \wedge \rho_j < \rho_i}} \left(Cs_{j,q} + \begin{cases} spin_{P_k,q} & \text{if } \rho_i \leq \rho^{\mathrm{spin}}_{P_k} \\ 0 & \text{otherwise} \end{cases} \right), \tag{16}$$

where $spin_{P_k,q}$ is calculated according to (3).

PROOF. When a task τ_j holds a global resource, according to Rule 4 it becomes non-preemptive on the processor, and therefore may block a higher priority task τ_i. The maximum blocking that a lower priority task τ_j can incur to τ_i is caused through its maximum global critical section. Hence, $\max_{\substack{\forall q: R_q \in \mathcal{RS}^{\mathrm{G}}_j \\ \wedge \rho_j < \rho_i}} \{Cs_{j,q}\}$ presents the maximum blocking that a lower priority task τ_j can cause to task τ_i, when τ_j executes within the critical section. However, under the worst-case scenario, when τ_j issues its request for a global resource, the resource might be remotely held by another task on another processor, thus τ_j spins until it acquires the resource. As a result, τ_i may in the worst case experience extra delay due to τ_j's spin-time. However, if $\rho_i > \rho^{\mathrm{spin}}_{P_k}$, the spinning cannot preempt τ_i. This scenario is formulated in (17). □

LEMMA 13. *The maximum LBG blocking duration incurred to a task τ_i by a lower priority task τ_j, on the same processor, under the CP spin-lock approach is denoted as $B^{\mathrm{G_{CP}}}_{i,j}$ and is calculated as follows.*

$$B^{\mathrm{G_{CP}}}_{i,j} = \max_{\substack{\forall q: R_q \in \mathcal{RS}^{\mathrm{G}}_j \\ \wedge \rho_j < \rho_i}} \left(Cs_{j,q} + \begin{cases} spin_{P_k,q} & \text{if } \rho_i \leq rc^{\mathrm{G}}_{P_k} \\ 0 & \text{otherwise} \end{cases} \right). \tag{17}$$

where $spin_{P_k,q}$ is calculated according to (3).

PROOF. Follows immediately from Lemma 12 since $\rho^{\mathrm{spin}}_{P_k} = rc^{\mathrm{G}}_{P_k}$. □

LEMMA 14. *The maximum LBG incurred to a task τ_i under the CP spin-lock approach is denoted as $B^{\mathrm{G_{CP}}}_i$ and is upper bounded as follows.*

$$B^{\mathrm{G_{CP}}}_i = \max_{\substack{\forall j: \rho_j < \rho_i \\ \wedge \tau_j \in \mathcal{T}_{P_k}}} \{B^{\mathrm{G_{CP}}}_{i,j}\}. \tag{18}$$

PROOF. Follows immediately from Lemma 6 and Corollary 13. □

Based on the aforementioned lemmas, we derive Theorem 1 as follows.

THEOREM 1. *The worst-case blocking of a task $\tau_i \in \mathcal{T}_{P_k}$ under the CP spin-lock approach is denoted as B_i^{CP} and is calculated as follows.*

$$B_i^{\mathrm{CP}} = \max\left(\max_{\substack{\forall j: rc^{\mathrm{G}}_{P_k}<\rho_j \\ \wedge \tau_j\in\mathcal{T}_{P_k}}}\{B^{\mathrm{L}}_{i,j}\} + B_i^{\mathrm{G_{CP}}},\ \max_{\substack{\forall j:\rho_j\le rc^{\mathrm{G}}_{P_k}\\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{\mathrm{L}}_{i,j}\}\right), \qquad (19)$$

where $B^{\mathrm{L}}_{i,j}$ and $B_i^{\mathrm{G_{CP}}}$ are calculated according to (15) and (14).

PROOF. We assume three scenarios: (i) $\rho_i \le rc^{\mathrm{G}}_{P_k}+1$, (ii) $rc^{\mathrm{G}}_{P_k}+1<\rho_i\le rc^{\mathrm{LG}}_{P_k}$, (iii) $rc^{\mathrm{LG}}_{P_k}<\rho_i$. In the first scenario, any task τ_j has a priority $\rho_j<\rho_i\le rc^{\mathrm{G}}_{P_k}+1$, i.e., $\rho_j<rc^{\mathrm{G}}_{P_k}+1$. Therefore, the term $\max_{\substack{\forall j: rc^{\mathrm{G}}_{P_k}<\rho_j \\ \wedge \tau_j\in\mathcal{T}_{P_k}}}\{B^{\mathrm{L}}_{i,j}\}=0$ in (19). Thus, the result of (19) will be the maximum value of the maximum LBG or LBL incurred by any lower priority task (calculated according to Lemmas 11 and 14). Therefore, first case reflects the same blocking scenario as in Lemma 8.

In the second scenario, according to Lemma 9 τ_i may experience at most two pi-blocking, one LBG incurred by a lower priority task with priority lower than $rc^{\mathrm{G}}_{P_k}$ (under *CP*) and one LBL incurred by a lower priority task with priority higher than $rc^{\mathrm{G}}_{P_k}$. This is formulated by the term $A=\max_{\substack{\forall j: rc^{\mathrm{G}}_{P_k}<\rho_j \\ \wedge \tau_j\in\mathcal{T}_{P_k}}} B^{\mathrm{L_{CP}}}_{i,j}+B_i^{\mathrm{G_{CP}}}$. Note that, the first term in summation function in A is resulted based on Lemmas 5, 11. The second term in summation function in A is calculated based on Lemma 14. Moreover according to Lemma 7, τ_i may experience at most one blocking (either LBL or LBG) incurred by any lower priority task τ_j with priority lower than $rc^{\mathrm{G}}_{P_k}$. Such blocking is formulated by the term $B=\max(\max_{\substack{\forall j:\rho_j\le rc^{\mathrm{G}}_{P_k}\\ \wedge\tau_j\in\mathcal{T}_{P_k}}} B^{\mathrm{L_{CP}}}_{i,j}, B_i^{\mathrm{G_{CP}}})$, considering Lemmas 5, 11 and 14. Moreover, according to Lemma 3, a task τ_i can experience at most one LBL from any lower priority task. Thus, the LBL term in A and B cannot happen together.

This results in the term $C=\max\Big(\max_{\substack{\forall j: rc^{\mathrm{G}}_{P_k}<\rho_j \\ \wedge \tau_j\in\mathcal{T}_{P_k}}}\{B^{\mathrm{L_{CP}}}_{i,j}\}+B_i^{\mathrm{G_{CP}}},$ $\max(\max_{\substack{\forall j:\rho_j\le rc^{\mathrm{G}}_{P_k}\\ \wedge\tau_j\in\mathcal{T}_{P_k}}} B^{\mathrm{L_{CP}}}_{i,j}, B_i^{\mathrm{G_{CP}}})\Big)$. As it can be observed, $B_i^{\mathrm{G_{CP}}}$ is a similar term in both sides of maximum function in C. Now let us assume two cases. (1) $B_i^{\mathrm{G_{CP}}}\ge B^{\mathrm{L_{CP}}}_{i,j}$. Thus $C=\max(\max_{\substack{\forall j: rc^{\mathrm{G}}_{P_k}<\rho_j \\ \wedge \tau_j\in\mathcal{T}_{P_k}}}\{B^{\mathrm{L_{CP}}}_{i,j}\}+B_i^{\mathrm{G_{CP}}}, B_i^{\mathrm{G_{CP}}})=\max_{\substack{\forall j: rc^{\mathrm{G}}_{P_k}<\rho_j \\ \wedge \tau_j\in\mathcal{T}_{P_k}}}\{B^{\mathrm{L_{CP}}}_{i,j}\}+B_i^{\mathrm{G_{CP}}}$, which is the first term in (19). (2) let us assume $B_i^{\mathrm{G_{CP}}}<B^{\mathrm{L_{CP}}}_{i,j}$, then C will lead to (19). As a result, (19) covers both (1) and (2) cases. This finishes the proof for the scenario (ii).

In the third scenario where $rc^{\mathrm{LG}}_{P_k}<\rho_i$, τ_i, by definition, does not use any global resource (recall Definition 3), i.e., it cannot experience any LBL. Thus, (19) simplifies to $B_i^{\mathrm{G_{CP}}}$ which is the same result inferred by Lemma 10. As it can be seen, all three possible scenarios (i), (ii) and (iii) lead (19) to reflect the blocking results from Lemmas 8, 9 and 10. This finishes the proof. □

6. $\widehat{CP}$ SPIN-LOCK APPROACH

In this section, we introduce a variant of *CP* which we denote as $\widehat{CP}$ and later in Section 7 we show that it dominates the *HP* approach. Under this spinning approach, $\rho^{\mathrm{spin}}_{P_k}=rc^{\mathrm{LG}}_{P_k}$ where $rc^{\mathrm{LG}}_{P_k}$ is the highest priority level that any task use local or global resources (recall Definition 4). Next, we elaborate the blocking terms that can be introduced to a task under this spin-lock protocol.

LEMMA 15. *The maximum LBL blocking duration incurred to a task τ_i under the $\widehat{CP}$ spin-lock approach is denoted as $B_i^{\mathrm{L}_{\widehat{\mathrm{CP}}}}$ and is calculated as follows.*

$$B_i^{\mathrm{L}_{\widehat{\mathrm{CP}}}}=\max_{\forall j:\rho_j<\rho_i\wedge\,\tau_i,\tau_j\in\tau_{P_k}}\{B^{\mathrm{L}}_{i,j}\}. \qquad (20)$$

PROOF. It is inferred from Lemmas 5 and 11. □

LEMMA 16. *The maximum LBG blocking duration incurred to a task τ_i by a lower priority task τ_j on the same processor under the $\widehat{CP}$ spin-lock approach is denoted as $B^{\mathrm{G}_{\widehat{\mathrm{CP}}}}_{i,j}$ and is calculated as follows.*

$$B^{\mathrm{G}_{\widehat{\mathrm{CP}}}}_{i,j}=\max_{\substack{\forall q:R_q\in\mathcal{RS}^{\mathrm{G}}_j\\ \wedge\rho_j<\rho_i}}\left(Cs_{j,q}+\begin{cases} spin_{P_k,q} & \text{if } \rho_i\le rc^{\mathrm{LG}}_{P_k}\\ 0 & \text{otherwise}\end{cases}\right), \qquad (21)$$

where $spin_{P_k,q}$ is calculated according to (3).

PROOF. Directly inferred from Lemma 16 since $\rho^{\mathrm{spin}}_{P_k}=rc^{\mathrm{LG}}_{P_k}$ under the $\widehat{CP}$ approach. □

LEMMA 17. *The maximum LBG incurred to a task τ_i under the $\widehat{CP}$ spin-lock approach is denoted as $B_i^{\mathrm{G}_{\widehat{\mathrm{CP}}}}$ and is upper bounded as follows.*

$$B_i^{\mathrm{G}_{\widehat{\mathrm{CP}}}}=\max_{\substack{\forall j:\rho_j<\rho_i\\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{\mathrm{G}_{\widehat{\mathrm{CP}}}}_{i,j}\}. \qquad (22)$$

PROOF. Follows immediately from Lemmas 6 and 16. □

THEOREM 2. *The maximum total pi-blocking incurred to a task $\tau_i\in\mathcal{T}_{P_k}$ under $\widehat{CP}$ approach is calculated as follows:*

$$B_i^{\widehat{\mathrm{CP}}}=\max(B_i^{\mathrm{L}_{\widehat{\mathrm{CP}}}},B_i^{\mathrm{G}_{\widehat{\mathrm{CP}}}}). \qquad (23)$$

PROOF. Let assume two scenarios (i) $\rho_i\le rc^{\mathrm{LG}}_{P_k}$, (ii) $\rho_i>rc^{\mathrm{LG}}_{P_k}$. Since under $\widehat{CP}$, $\rho^{\mathrm{spin}}_{P_k}=rc^{\mathrm{LG}}_{P_k}$, under the first scenario, according to Lemma 2, τ_i may experience at most one blocking. This blocking can be due to either LBL or LBG, which is formulated in (23). In the second scenario, by definition, τ_i does not use any local resource, thus it cannot experience any LBL. However, it may still experience LBG from a lower priority task. (23) reflects this scenario as well. This finishes the proof. □

Since under $\widehat{CP}$, $\rho^{\mathrm{spin}}_{P_k}=rc^{\mathrm{LG}}_{P_k}\ge rc^{\mathrm{G}}_{P_k}$, therefore according to Lemma 4 $spin_{P_k,q}$ and $spin_i$ can be computed using (3) and (4), respectively. Moreover, similar to *HP* and *CP* approaches, under $\widehat{CP}$ spin-lock approach, the execution time of tasks should be inflated by spinning, as well. Thus, the actual execution time of tasks are calculated using (5).

7. COMPARING HP, CP AND $\widehat{CP}$

In this section we compare the HP, CP and $\widehat{CP}$ approaches. In Section 7.3, we show that $\widehat{CP}$ dominates HP approach. Later in Section 7.4, we show by means of two examples 1 and 2 that CP and $\widehat{CP}$ are incomparable (Definition 11). To facilitate the comparison of CP, $\widehat{CP}$ and HP approaches, first we derive a unified blocking equation that can cover spin-lock approaches that use spin priority levels from CP to HP, i.e., $rc^{G}_{P_k} \le \rho^{spin}_{P_k} \le \rho^{max}_{P_k}$, in the following section.

7.1 Unification of CP, $\widehat{CP}$ and HP blocking terms

In this section, we derive general blocking terms where assigning $\rho^{spin}_{P_k}$ equal to $\rho^{max}_{P_k}$, $rc^{G}_{P_k}$ or $rc^{LG}_{P_k}$ will result in the related blocking terms under HP, CP and $\widehat{CP}$, respectively.

$$B_i(\rho^{spin}_{P_k}) = \max\left(\max_{\substack{\forall j:\rho^{spin}_{P_k}<\rho_j \\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{L}_{i,j}\} + B^{G}_i(\rho^{spin}_{P_k}),\ \max_{\substack{\forall j:\rho_j\le\rho^{spin}_{P_k} \\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{L}_{i,j}\}\right), \tag{24}$$

where, $B^{L}_{i,j}$ is calculated according to (15) and,

$$B^{G}_i(\rho^{spin}_{P_k}) = \max_{\forall j:\rho_j<\rho_i\wedge\tau_i,\tau_j\in\mathcal{T}_{P_k}}\{B^{G}_{i,j}(\rho^{spin}_{P_k})\}, \tag{25}$$

where $B^{G}_{i,j}(\rho^{spin}_{P_k})$ is calculated according to (16).

LEMMA 18. *The blocking term presented in (24) covers blocking scenarios under any spin-lock approach where $rc^{G}_{P_k} \le \rho^{spin}_{P_k} \le \rho^{max}_{P_k}$.*

PROOF. We assume three scenarios for a task τ_i: (i) $\rho_i \le rc^{G}_{P_k}+1$, (ii) $rc^{G}_{P_k}+1<\rho_i\le rc^{LG}_{P_k}$, (iii) $\rho_i > rc^{LG}_{P_k}$.

Under the first scenario, we prove that (24) leads to the same blocking results as in Lemma 8. Since $\rho_j<\rho_i\le rc^{G}_{P_k}+1$ in the first scenario, and $\rho^{spin}_{P_k}\ge rc^{G}_{P_k}$, the result of the set $\forall j:\rho^{spin}_{P_k}<\rho_j$ in the term $\max_{\substack{\forall j:\rho^{spin}_{P_k}<\rho_j \\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{L}_{i,j}\}$ in (24) is equal to $\emptyset$, which makes this term equal to zero. Therefore, (24) is simplified to $A=\max\Big(B^{G}_i(\rho^{spin}_{P_k}),$ $\max_{\substack{\forall j:\rho_j\le\rho^{spin}_{P_k} \\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{L}_{i,j}\}\Big)$, which shows the maximum of one LBL and LBG term. Now we show that, for any $\rho^{spin}_{P_k}\ge rc^{G}_{P_k}$ included in A, still Lemma 8 is valid. This is proved, since under scenario (i) $\rho_i\le rc^{G}_{P_k}$, thus, for any lower priority task τ_j, $\rho_i<rc^{G}_{P_k}$ is satisfied. Hence, condition in Lemma 8 is valid for LBL term in A. For $\rho^{spin}_{P_k})\le rc^{G}_{P_k}$, the LBG term in (24) is calculated according to Lemma 12. As a result, (24) reflects the blocking results from Lemma 8 which is also valid for any spin-priority higher than $rc^{G}_{P_k}$.

Under the second scenario, we prove that the condition in (ii) results in (24). According to Lemma 9, τ_i may experience at most two blocking, one LBL and one LBG from any lower priority task. This results in $B=\max_{\substack{\forall j:\rho^{spin}_{P_k}<\rho_j \\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{L}_{i,j}\}+B^{G}_i(\rho^{spin}_{P_k})$. Moreover, according to Lemma 7, τ_i may experience at most one pi-blocking from any lower priority task, this is formulated by $C=\max(\max_{\substack{\forall j:\rho^{spin}_{P_k}<\rho_j \\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{L_{CP}}_{i,j}\}+$ $B^{G_{CP}}_i, \max(\max_{\substack{\forall j:\rho_j\le rc^{G}_{P_k} \\ \wedge\tau_j\in\mathcal{T}_{P_k}}} B^{L_{CP}}_{i,j}, B^{G_{CP}}_i))$. Since $rc^{G}_{P_k}\le\rho^{spin}_{P_k}$, the outcome set under the condition of $\forall j:\rho_j\le rc^{G}_{P_k}$ in C stays unchanged if $rc^{G}_{P_k}$ in this condition is substituted by such $\rho^{spin}_{P_k}$. It is easy to see that C, undeer both conditions of (1) $B^{G_{CP}}_i\ge B^{L_{CP}}_{i,j}$, and (2) $B^{G_{CP}}_i< B^{L_{CP}}_{i,j}$ gives the same result as in (24).

Under scenario (iii), we prove that (24), gives similar results as in Lemma 10. A task $\rho_i>rc^{LG}_{P_k}$, by definition does not use any local resource (recall Definition 4). Thus, (24) is simplified to $B^{G}_i(\rho^{spin}_{P_k})$, which is similar to the result derived by Lemma 10. This finishes the proof. $\square$

In order to compare the CP and $\widehat{CP}$ and HP spin-lock approaches, we divide the available priority levels on a processor P_k into three ranges and we compare the two approaches for tasks in these priority ranges separately. The priority ranges, also illustrated in Figure 1, are as follows: (A) $\forall\rho_i\mid\rho_i>rc^{LG}_{P_k}\wedge\tau_i\in\mathcal{T}_{P_k}$, (B) $\forall\rho_i\mid rc^{G}_{P_k}<\rho_i\le rc^{LG}_{P_k}\wedge\tau_i\in\mathcal{T}_{P_k}$ and (C) $\forall\rho_i\mid\rho_i\le rc^{G}_{P_k}\wedge\tau_i\in\mathcal{T}_{P_k}$.

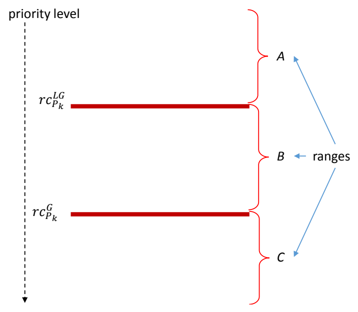

Figure 1: Priority ranges

7.2 CP versus HP

We have shown in [1] that CP and HP approaches are incomparable. In this section we specify the set of tasks on a processor P_k that using either HP or CP spin-lock approaches for them leads to similar blocking results. This facilitates the comparison of the two approaches later in Section 9.

LEMMA 19. *We assume two different spin-lock approaches 1 and 2 on P_k, with spin priority level as $\rho_1^{spin_{P_k}}$ and $\rho_2^{spin_{P_k}}$, respectively, where $\rho_1^{spin_{P_k}}\le\rho_2^{spin_{P_k}}$. For any task $\tau_i\in\mathcal{T}_{P_k}$, if $\rho_i\le\rho_1^{spin_{P_k}}$, then using either spin-lock approaches 1 or 2 will lead to similar worst case blocking results, which is upper bounded as follows for τ_i.*

$$B_i=\max\left(\max_{\substack{\forall q:R_q\in\mathcal{RS}^{G}_j \\ \wedge\rho_j<\rho_i}}(Cs_{j,q}+spin_{P_k,q}),\ \max_{\substack{\forall j:\rho_j\le\rho_i \\ \wedge\tau_j\in\mathcal{T}_{P_k}}}\{B^{L}_{i,j}\}\right). \tag{26}$$

where $spin_{P_k,q}$ and $B^{L}_{i,j}$ are calculated according to (3) and (15), respectively.

PROOF. It is inferred from (24) considering: (i) the common result of the sets $\forall j : \rho_j \leq \rho_1^{\text{spin}_{P_k}}$ and $\forall j : \rho_j \leq \rho_2^{\text{spin}_{P_k}}$ under spin-lock approaches 1 and 2, respectively, is similar to $\forall j : \rho_j < \rho_i$ when $\rho_i\rho_1^{\text{spin}_{P_k}}$, (ii) the output result of the sets $\forall j : \rho_j > \rho_2^{\text{spin}_{P_k}}$ and $\forall j : \rho_j > \rho_1^{\text{spin}_{P_k}}$ under spin-lock approaches 1 and 2, respectively, is equal to $\emptyset$, since $\rho_j < \rho_i \leq \rho_1^{\text{spin}_{P_k}} \leq \rho_2^{\text{spin}_{P_k}}$, and (iii) the fact that the condition $\rho_i \leq \rho^{\text{spin}_{P_k}}$ in (16) is satisfied under both spin-lock approaches. □

COROLLARY 1. *For any task $\tau_i \in \mathcal{T}_{P_k}$, if $\rho_i \leq rc^{\text{G}}_{P_k}$, (i.e., range C in Figure 1) then using either CP or HP will lead to similar worst case blocking results as in (26).*

PROOF. It is directly inferred from Lemma 19. □

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

In the following, we show that $\widehat{CP}$ dominates *HP* and all the spin lock approaches in between. Moreover, we specify the set of tasks on a processor P_k that using either *HP* or $\widehat{\mathbf{CP}}$ spin-lock approaches for them leads to similar blocking results which eases the comparison of the two approaches later in Section 9.

COROLLARY 2. *For any task $\tau_i \in \mathcal{T}_{P_k}$ if $\rho_i \leq rc^{\text{LG}}_{P_k}$, (ranges B and C in Figure 1) then using either $\widehat{CP}$ or HP will lead to similar worst case blocking results as in (26).*

PROOF. It is directly inferred from Lemma 19. □

LEMMA 20. *We assume two different spin-lock approaches* 1 *and* 2 *on P_k, with spin priority level as $\rho_1^{\text{spin}_{P_k}}$ and $\rho_2^{\text{spin}_{P_k}}$, respectively, where $\rho_1^{\text{spin}_{P_k}} \leq \rho_2^{\text{spin}_{P_k}}$, $\rho_1^{\text{spin}_{P_k}} \geq rc^{\text{G}}_{P_k}$ and $\rho_2^{\text{spin}_{P_k}} \geq rc^{\text{LG}}_{P_k}$. For any task $\tau_i \in \mathcal{T}_{P_k}$, if $\rho_i > \rho_2^{\text{spin}_{P_k}}$, then using either spin-lock approaches* 1 *or* 2 *will lead to similar worst case blocking results, which is upper bounded as follows for τ_i.*

$$B_i = \max_{\substack{\forall q: R_q \in \mathcal{RS}^{\text{G}}_j \\ \wedge \rho_j \leq rc^{\text{G}}_{P_k}}} Cs_{j,q}. \tag{27}$$

PROOF. It is inferred from (24) considering: (i) any task τ_i with priority $\rho_i > \rho_2^{\text{spin}_{P_k}} \geq rc^{\text{LG}}_{P_k}$, by definition does not use any local resource, thus, it does not experience any LBL from any lower priority task, (ii) the condition $\rho_i \leq \rho^{\text{spin}_{P_k}}$ in (16) is not satisfied under both spin-lock approaches, and (iii) LBG can be incurred from any task with priority lower than $rc^{\text{G}}_{P_k}$. □

LEMMA 21. *$\widehat{CP}$ dominates and any spin-lock approach that uses a spin priority level higher than the spin-lock priority used by $\widehat{CP}$, including HP approach (Recall Definition 10).*

PROOF. We assume a spin-lock approach *A* with spin-lock priority $\rho_A^{\text{spin}_{P_k}}$ where $\rho_A^{\text{spin}_{P_k}} > rc^{\text{LG}}_{P_k}$. According to Definition 10, in order to $\widehat{CP}$ dominates such spin lock approach *A*, any task set that is schedulable using *A* should also be schedulable using $\widehat{CP}$, however, there may exist tsk set that is schedulable using $\widehat{CP}$ but not *A*. Moreover, according to Lemmas 19 and 20, for any task τ_i with priority $\rho_i \leq rc^{\text{LG}}_{P_k}$ or $\rho_i > \rho_A^{\text{spin}_{P_k}}$, respectively, using either spin-lock approaches lead to similar blocking results. Therefore, we assume a task τ_i with priority $rc^{\text{LG}}_{P_k} < \rho_i \leq \rho_A^{\text{spin}_{P_k}}$. According to Definition 4, such a task τ_i does not use any local (or global) resource. Therefore, τ_i cannot experience any LBL. It can be seen from (24) that τ_i will experience one LBG only. However, according to (16), under *A*, τ_i will have an extra $spin_{P_k,q}$ term compared to $\widehat{CP}$, since the condition $\rho_i \leq \rho^{\text{spin}}_{P_k}$ is satisfied under *A* and not $\widehat{CP}$. Hence, the maximum blocking term and accordingly the worst case response time (12) under *A* is larger than under $\widehat{CP}$, $WR_i^{\text{A}} > WR_i^{\widehat{\text{CP}}}$, where WR_i^{A} and $WR_i^{\widehat{\text{CP}}}$ denoted the worst-case response times under *A* and $\widehat{CP}$, respectively. Therefore, if τ_i is schedulable under *A*, i.e., $WR_i^{\text{A}} \leq D_i$, it defiantly meets its deadline under $\widehat{CP}$, as well, i.e., $WR_i^{\widehat{\text{CP}}} \leq D_i$. However, if τ_i is schedulable under $\widehat{CP}$, if $D_i = WR_i^{\widehat{\text{CP}}}$, then $WR_i^{\text{A}} > D_i$ and it misses its deadline under *A*. *A* can be any spin-lock approach that uses a spin-priority higher than $rc^{\text{LG}}_{P_k}$, including *HP*. This finishes the proof. □

COROLLARY 3. *If $rc^{\text{G}}_{P_k} = rc^{\text{LG}}_{P_k}$, then CP dominates HP.*

PROOF. Follows immediately from Lemma 21. □

7.4 $\widehat{CP}$ versus *CP*

In the following, first we specify the set of tasks on a processor P_k that using either *CP* or $\widehat{CP}$ spin-lock approaches for them leads to similar blocking results which simplifies the comparison of the two approaches later in Section 9. Secondly, we show by means of two Examples 1 and 2 that for a task in priority range *B*, the *CP* and $\widehat{CP}$ approaches are incomparable (Definition 11).

COROLLARY 4. *If $rc^{\text{G}}_{P_k} \leq rc^{\text{LG}}_{P_k}$, for any task $\tau_i \in \mathcal{T}_{P_k}$ if $\rho_i \leq rc^{\text{G}}_{P_k}$, or $\rho_i > rc^{\text{LG}}_{P_k}$ (ranges C and A in Figure 1) then using either CP or $\widehat{CP}$ will lead to similar worst case blocking results.*

PROOF. It is directly inferred from Lemmas 19 and 20. □

Next, we show by means of two Examples 1 and 2, that for a task τ_i (τ_4 in these examples) in priority range *B*, the *CP* and $\widehat{CP}$ approaches are incomparable (Definition 11).

In Table 2, the specification of the task set used in the Examples 1, 2 and 3 (Example 3 will be explained later in Section 8), is shown. Note that tasks τ_3 and τ_7 have two different critical section lengths for local resource R_l and global resource R_g and consequently two different C_3 and C_7 values in the examples. In Example 1, $Cs_{3,l} = 1$ and $C_3 = 2$ and in Examples 2 and 3 $Cs_{3,l} = 3$ and $C_3 = 4$. $Cs_{7,g} = 5.5$ and $C_7 = 7$ in the examples 1 and 3 and $Cs_{7,g} = 2.5$ and $C_7 = 4$ in the example 2. Note that, in the figures of three examples, the priority levels 1 to 6 shows the priority levels of tasks τ_1 to τ_6 on processor P_1. The priority level 7 shows the highest priority level on core P_1 (Definition 1). Such priority level is not shown for processor P_2 since only one task exists on P_2. Moreover, each task executes on its own priority level unless the priority of the task is

P_k	τ_i	C_i	$Cs_{i,l}$	$Cs_{i,g}$
P_1	τ_1	4	-	3
	τ_2	1	-	1
	τ_3	2, 4, 4	1, 3, 3	-
	τ_4	3	-	-
	τ_5	1	1	-
	τ_6	1	-	-
P_2	τ_7	7, 4, 7	-	5.5, 2.5, 5.5

Table 2: Task set specification in Examples 1, 2 and 3

increased. In this case, we show the execution of the task on the boosted priority level. E.g., task τ_1, normally executes on priority level 1, but when it spins with priority level 2 in Figure 2.(a), it moves to priority level 2. When τ_1 is granted to its resource later at time 6, its priority is further boosted to higher than any normal priority on P_1 and its execution moves to priority level 7. Note that, in all the three figures related to Examples 1, 2 and 3, the task arrival and deadline shown in each priority level is related to the task of that priority level. E.g, in Figure 2.(b), the deadline on priority level 5 is related to the task τ_5, which meets its deadline in this figure.

EXAMPLE 1. *Under the CP approach (see Figure 2.(a)), when task τ_1 requests R_g which is locked by task τ_7 on P_2, it spins with priority level $rc^{\mathrm{G}}_{P_1} = 2$. Thus, task τ_3 can preempt τ_1 when it arrives at time 2. Task τ_4 cannot run when it arrives at time 3 since τ_3 has requested R_l with ceiling 5 slightly before time 3 (SRP rule). Thus, τ_4 has to wait until τ_3 finishes its lcs at time 4. Even though τ_4 is further preempted by arrival of the higher priority task τ_6 at time 5, and later by non-preemptive execution of τ_1 (which is granted to R_g) at time 6, it meets its deadline at time 11. As it can be seen, under this approach, τ_4 does not wait for spinning time of τ_1 on R_g. However, under the same scenario, if $\widehat{CP}$ approach is used (see Figure 2.(b)), τ_4 cannot anymore run when it arrives (at time 3) since under this approach, τ_1 is spinning with priority level $rc^{\mathrm{LG}}_{P_1} = 5$. As a result τ_4 misses its deadline.*

EXAMPLE 2. *Under the CP spin-locking approach (see Figure 3.(a)), task τ_1 requests the global resource R_g at time 1 that is used by task τ_7 on P_2 , thus, spins with priority level $rc^{\mathrm{G}}_{P_1} = 2$. Therefore, (similar to Figure 2.(a)), task τ_3 can preempt it when it arrives at time 2 and issues a request on the local resource R_l at time 3. However, τ_3 cannot immediately use R_l, since τ_1 is granted to R_g at time 3, therefore τ_1 executes using R_g non-preemptively (Rule 4). As soon as τ_1 finishes its gcs part, first τ_6 runs which its execution was delayed from its arrival at 5 and then τ_3 will start the lcs part with priority level 5. The execution of τ_4 which has arrived at time 3 is delayed until time 10 where τ_3 finishes its lcs part. As a result, τ_4 will miss its deadline at time 11 under this scenario. Under the same scenario τ_4 meets its deadline using $\widehat{CP}$ as can be seen in Figure 3.(b). The reason is that under $\widehat{CP}$, τ_1 that is blocked on R_g spins with priority level $rc^{\mathrm{LG}}_{P_1} = 5$ thus, does not let τ_3 to start running so that it can issue its local resource request. As a result, τ_4 will only experience LBG delay from τ_1 and meets its deadline.*

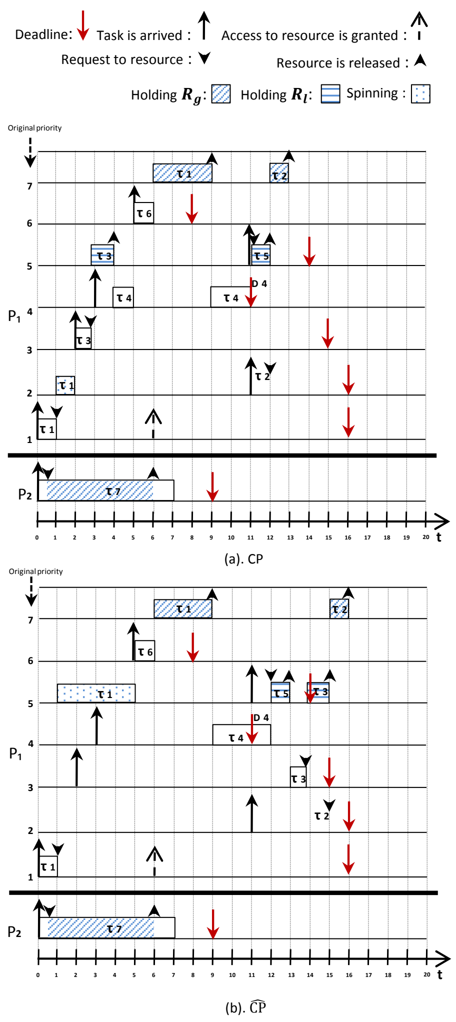

Figure 2: Example 1: The global resource R_g is used by tasks τ_1 and τ_2 on P_1 and τ_7 on P_2. Task τ_3 and τ_5 on P_1 use the local resource R_l. τ_4 and τ_6 do not use any resource. Case (a) illustrates the *CP* spin-lock approach and case (b) the $\widehat{CP}$ spin-lock approach. Task τ_4 misses its deadline under $\widehat{CP}$.

8. INTERMEDIATE SPIN-LOCK APPROACH

In Section 7, we showed that the two *CP* and $\widehat{CP}$ spin-lock approaches are incomparable. In this section, we show by means of an example (Example 3) that if *CP* and $\widehat{CP}$ spin-lock approaches cannot make a task set schedulable on a core, there may exist an intermediate spin-lock approach which can make the task set schedulable. This intermediate spin-lock approach uses a priority level between the two priority levels that are used by the *CP* and $\widehat{CP}$ approaches.

EXAMPLE 3. *As illustrated in Figures 4.(a) and (b), task τ_4 misses its deadline under both CP and $\widehat{CP}$ spin-lock approaches. In Figure 4.(a), τ_4 misses its deadline due to the normal execution of task τ_6 in form of interference delay, and lcs (local critical section i.e., the section of a task that uses local resource) and gcs of tasks τ_3 and τ_1, respectively. In Figure 4.(b), τ_4 misses its deadline due to the spinning time of τ_1 on R_g, the gcs of τ_1 and interference of τ_6. It is shown in Figure 4.(c) that if $\rho^{\mathrm{spin}}_{P_k} = 3$, i.e., τ_1 spins on priority level 3 all tasks including task τ_4 will meet their deadlines. When task τ_1 spins on priority level 3, task τ_3 cannot start running,*

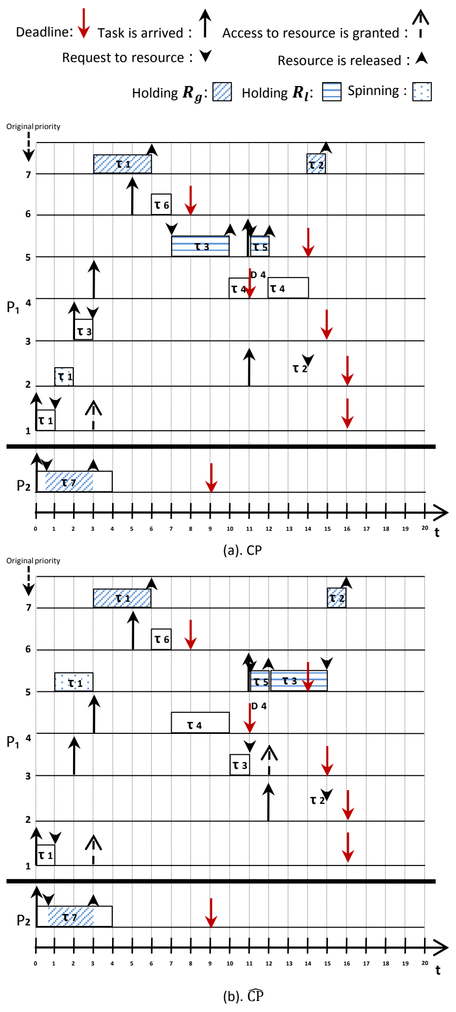

Figure 3: Example 2: The global resource R_g is used by tasks τ_1 and τ_2 on P_1 and τ_4 on P_2. Task τ_3 and τ_5 on P_1 use the local resource R_l. τ_4 and τ_6 do not use any resource. Case (a) illustrates the CP spin-lock approach and case (b) the $\widehat{CP}$ spin-lock approach. Task τ_4 misses its deadline under CP.

which implies that it cannot issue its request on R_l and its delay to τ_4 is prevented similar to scenario under Figure 2.(a). On the other hand, τ_4 can preempt τ_1's spinning when it arrives at time 3. Therefore, τ_4 does not have to wait for the waiting time of τ_1 on R_g similar to the scenario in Figure 3.(b). τ_4 only experiences a delay from normal execution of τ_6 (in form of interference) and non-preemptable gcs execution of task τ_1. Thus,τ_4 finishes at time 10 before its deadline.

8.1 Key trade-off factors

In this section, we elaborate the factors that may cause longer delays due to resource sharing under each $\widehat{CP}$ and CP spin-lock approaches.

According to Corollary 4, for any task on a processor P_k that has a priority within ranges A or C (Figure 1), using $\widehat{CP}$ and CP leads to similar results. However, as also shown in Examples 1 and 2, for a task that has a priority in range B, using $\widehat{CP}$ and CP may lead to different results (task τ_4 in Examples 1 and 2). Assuming a task

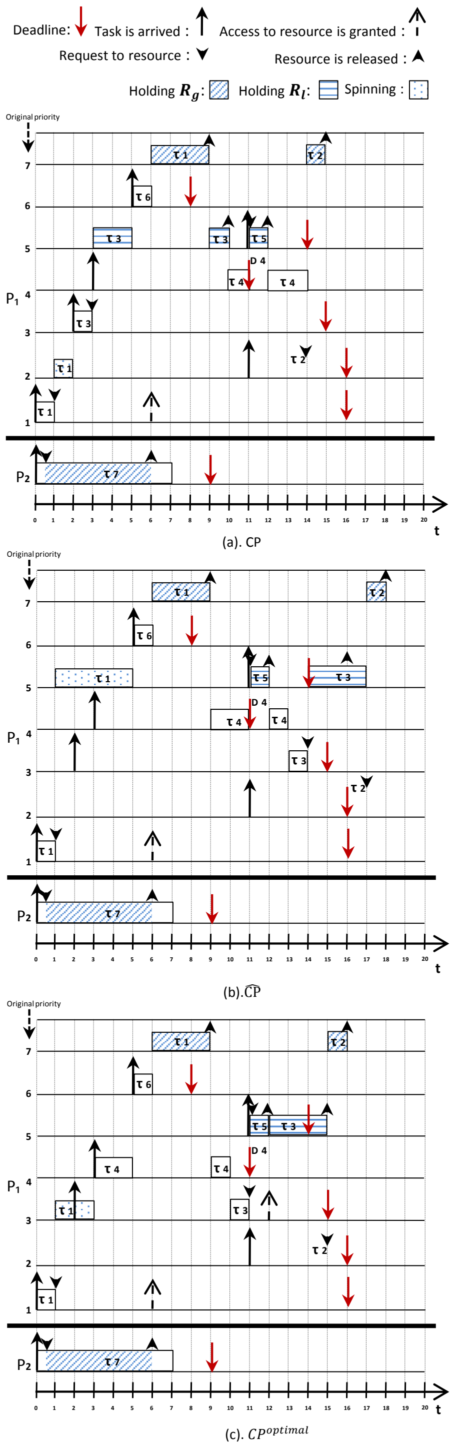

Figure 4: Example 3: The global resource R_g is used by tasks τ_1 and τ_2 on P_1 and τ_7 on P_2. Task τ_3 and τ_5 on P_1 use the local resource R_l. Tasks τ_4 and τ_6 do not use any resource. Case (a) illustrates the CP spin-lock approach, case (b) illustrates the $\widehat{CP}$ spin-lock approach and case (c) illustrates an intermediate spinning which uses a spin priority level between CP and $\widehat{CP}$ approaches. Task τ_4 misses its deadline under both CP and $\widehat{CP}$ while it is schedulable under spin-lock protocol using an intermediate spin-lock priority.

τ_i with a priority within range B, by looking at RHS of (24), we can observe that if $\rho^{\text{spin}}_{P_k} = rc^{\text{LG}}_{P_k}$ (i.e. $\widehat{CP}$ approach is used), the term $B^{\text{L}}_{i,j}$ in the first argument of the maximum function will be equal to zero. The reason is that, a task τ_j where $\rho_j < \rho_i$ does not exist such that the condition of this term (i.e. $\rho^{\text{spin}}_{P_k} = rc^{\text{LG}}_{P_k} < \rho_j$) is satisfied, since in this range $\rho_i \leq rc^{\text{LG}}_{P_k}$. However, this term is non-zero under CP approach since the condition $\rho^{\text{spin}}_{P_k} = rc^{\text{G}}_{P_k} < \rho_j < \rho_i$ can be satisfied. In other words, under CP, there may exist a task τ_j (τ_3 in Figure 2.(a)) with a lower priority than that of τ_i (τ_4 in Figure 2.(a)) that can cause LBL to τ_i due to requesting a local resource R_l with ceiling higher than the priority of τ_i. This implies that, using CP may cause an extra LBL term (the $B^{\text{L}}_{i,j}$ term in plus function in (24)) besides the LBG term ($B^{\text{G}}_i(\rho^{\text{spin}}_{P_k})$ term in (24)) compared to using $\widehat{CP}$. This scenario can be seen in Figure 3 where τ_4 experiences LBL by τ_3 and LBG by τ_1 under CP whereas it experiences only LBG by τ_1 under $\widehat{CP}$. On the other hand, the term $B^{\text{G}}_i(\rho^{\text{spin}}_{P_k})$ in (24) can be smaller for a task τ_i in range B using CP, by looking at (16). In (16), if CP is used, $spin_{P_k,q}$ term is zero, since $\rho_i > rc^{\text{G}}_{P_k}$ (in range B). This is not the case under $\widehat{CP}$. In other words, a task τ_i belonging to range B, have to wait for a lower priority task's waiting time for its requested resource under $\widehat{CP}$ but not under CP since $\rho_i <\leq rc^{\text{LG}}_{P_k}$ for a task τ_i in range B. This scenario can be seen in Figure 2 where τ_4 is delayed by spinning of τ_1 (waiting time of τ_1 for R_g) under the $\widehat{CP}$ which is not the case 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 spinning time of a lower priority task. These two parameters determine the trade-off factors of the two approaches. One conclusion from this discussion is that if the extra LBL is not incurred under CP, then using CP is dominant to using $\widehat{CP}$ since under $\widehat{CP}$ a task may be delayed longer due to the lower priority tasks spinning.

Example 3, showed that a spin-lock priority level between the priority levels used by CP and $\widehat{CP}$, (i.e., $rc^{\text{G}}_{P_k}$ and $rc^{\text{LG}}_{P_k}$) may exist that can make a task set schedulable where CP and $\widehat{CP}$ cannot. To determine such spin-lock priority the priority levels between CP and $\widehat{CP}$ can be explored to find such spin-lock priority, if any. The complexity of such a search is linear and equal to the number of the tasks that has priority levels between $rc^{\text{G}}_{P_k}$ and $rc^{\text{LG}}_{P_k}$.

9. EVALUATION

In this section we present the experimental results of comparing the HP, CP and $\widehat{CP}$ spin lock approaches. According to Corollaries 1, 2 and 4, it is enough to compare the worst case response time of tasks in range B when comparing CP and $\widehat{CP}$, range A when comparing $\widehat{CP}$ and HP and range $A \cup B$ when comparing CP and HP spin-lock approaches. In our experiments, we therefore only consider tasks in the related range , effectively removing tasks that have similar results under the compared approaches. In our experiments we measure the improvement in worst case response time of tasks of one approach compared to another. We use $RTI(a,b)$to denote the *response time improvement* under approach a compared to approach b. We denote RTI_i(a,b) for a task τ_i as $\frac{(WR^a_i - WR^b_i)}{max(WR^a_i, WR^b_i)} \times 100$, where WR^a_i and WR^b_i denote the worst case response time of task τ_i under approaches a and b, respectively. For a randomly generated task set, we show the percentage of tasks as a function of RTI(a,b). We show how different systems parameters such as number of processors in the platform, task set utilization, number of tasks in a task set and local and global critical section lengths can affect the RTI factor for tasks under different approaches.

9.1 Experimental Setup

In each experiment we randomly generate task sets for each processor of the platform. A platform contains m processors , where m is selected from the set $\{4, 8, 12, 16\}$. For each experiment 100 platforms are generated. The task set size is the same for each processor and is selected from the range [10,100] with steps of 30. The task set utilization is also the same for each processor and is selected from the set $\{0.4, 0.6, 0.8, 1\}$. The UUnifast algorithm [5] 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 based on $C_i = U_i T_i$. Deadlines of tasks are selected randomly according to a uniform distribution in the range $[C_i + \alpha(T_i - C_i), T_i]$ with $\alpha = 0.5$ as the default [10].

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 $Cs_q = \beta C_i$, 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 platform is set to 3.

In our basic system configuration, the number of processors $m = 4$, the task set utilization per core is 0.8, the number of tasks on each processor is 20, and $\beta = 0.2$.

9.2 Results

As mentioned early in this section, we measure the improvement of response time of tasks (RTI value) under one approach versus another using bar charts. In all graphs presented in this section, the distribution of the tasks for the measured RTI is shown. The X-axis in the graphs represent the RTI value of one approach versus the other 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. This means that a value x_i in the X-axis is a representative value for the range (x_{i-1}, x_i]. Both X-axis and Y-axis are represented in percentage. 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 approach a compared to approach b. Note that a positive RTI value for a graph representing $RTI(a,b)$, shows that response times under approach a are larger compared to approach b. Similarly a negative RTI value shows that response times are smaller under approach a compared to approach b.

The results in Figures 5, 10 and 15 show the variation in distribution of tasks for the measured RTI values for different numbers of processors in the generated multiprocessor platform. The results shown in Figures 6, 11 and 16 shows the variation in distribution of tasks for the measured RTI values for different numbers of tasks per task set. The results shown in Figures 7, 12 and 17 show the variation in distribution of tasks for the measured RTI values for different task set utilizations. The results shown in Figures 8, 13 and 18 show the variation in distribution of tasks for the measured RTI values for different length of local critical sections and Figures 9, 14 and 19 show the variation in distribution of tasks for the measured RTI values versus changing the length of global critical sections. Next we will consider the results in more detail.

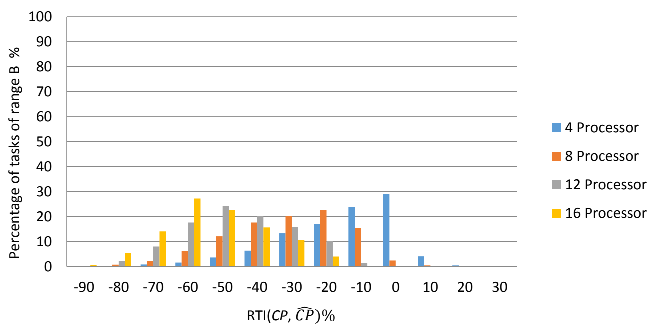

Figure 5: Distribution of tasks according to response time improvement under CP compared to $\widehat{CP}$ for different numbers of processors.

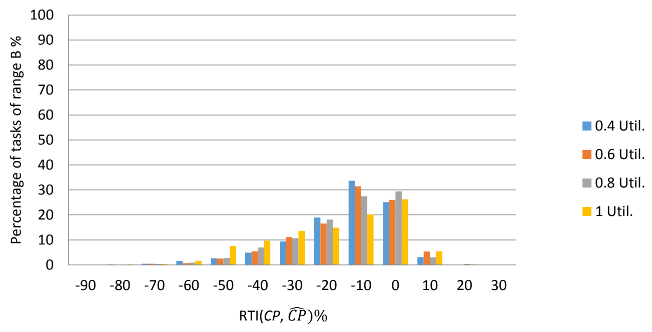

Figure 7: Distribution of tasks according to response time improvement under CP compared to $\widehat{CP}$ for different task set utilizations.

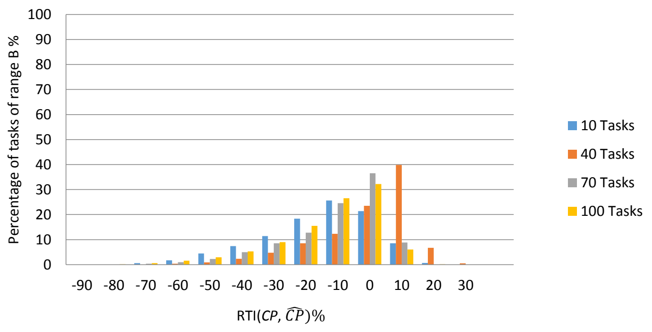

Figure 6: Distribution of tasks according to response time improvement under CP compared to $\widehat{CP}$ for different numbers of tasks.

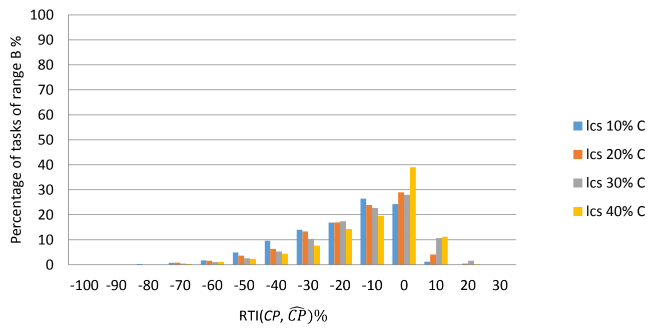

Figure 8: Distribution of tasks according to response time improvement under CP compared to $\widehat{CP}$ for different lcs lengths.

9.2.1 *Evaluation results of CP versus $\widehat{CP}$*

Figure 5 shows that increasing the number of processors in the platform will lead to more tasks having larger improvements in their response times under the CP approach compared to $\widehat{CP}$ approach. As it can be seen in the figure, when the number of processors is 4 in the platform, around 5% of tasks have 0 to 10 % improvement in response time under $\widehat{CP}$ compared to CP and for 8 processors in the platform, less than 2% of tasks have between 0 to 10% improvement under $\widehat{CP}$ compared to CP. For 12 and 16 processors, the experiments show that the CP approach outperforms the $\widehat{CP}$ approach. For example, it can be seen that for 16 number of processors, around 27% of the tasks have around 60% to 70% improvement in the response time and around 5% of tasks have even 80% improvement in the response time under CP compared to $\widehat{CP}$. This observation can be confirmed by revisiting (24). The only term that changes by changing the number of processors is $B_i^{\mathrm{G}}(\rho_{P_k}^{\mathrm{spin}})$ in (24) where according to (16) the term will include $spin_{P_k,q}$ under $\widehat{CP}$ and not under CP. It can easily be concluded from (3) that $spin_{P_k,q}$ is positively correlated to an increasing the number of processors, i.e., $spin_{P_k,q}$ cannot decrease by increasing the number of processors. The same effect can also be seen in Figure 9, where by increasing the global critical sections length, CP outperforms the $\widehat{CP}$. This is due to the fact that by increasing the global critical sections length $spin_{P_k,q}$ is increased so will $B_i^{\mathrm{G}}(\rho_{P_k}^{\mathrm{spin}})$ under the $\widehat{CP}$ approach.

However, the graphs in Figures 6, 7 and 8 show a different effect under the CP and $\widehat{CP}$ approaches. In these experiments it is observed that by increasing the task set utilization, the number of tasks per task set and local critical section lengths the performance of CP decreases compared to $\widehat{CP}$. Note that, by better performance we mean higher improvement in terms of response time duration which refers to shorter response times. The reason is that by increasing the number of tasks on a processor, the number of tasks in range B may increase as well, which can lead in an increase in $\max_{\substack{\forall j:\rho_{P_k}^{\mathrm{spin}}<\rho_j \\ \wedge\tau_j\in\mathcal{T}_{P_k}}} B_{i,j}^{\mathrm{L}}$ term in (24). The reason is that for a task τ_i, the number of related lower priority (ρ_j) tasks may increase and contribute to this term). Note that, this term is zero under $\widehat{CP}$.

The same aforementioned effect, i.e., $\widehat{CP}$ outperforms the CP, happens by also increasing the local critical section lengths due to an increase in $B_{i,j}^{\mathrm{L}}$ term. The same effect holds also when the task set utilization are increased. By increasing the task set utilization we directly increase the tasks execution times. Thus, indirectly the critical section lengths are increased since, $Cs_q = \beta C_i$.

The interesting observation when comparing CP versus $\widehat{CP}$, is that we have both positive and negative RTI values confirms that CP and $\widehat{CP}$ are incomparable as also already shown by the examples in Figures 2 and 3.

9.2.2 *Evaluation results of CP versus HP*

Figures 10, 11, 12, 13 and 14 show similar results as Figures 5, 6, 7, 8 and 9 for the same reasons as mentioned in Section 9.2.1. This is due to the fact that HP behaves in a similar way towards CP

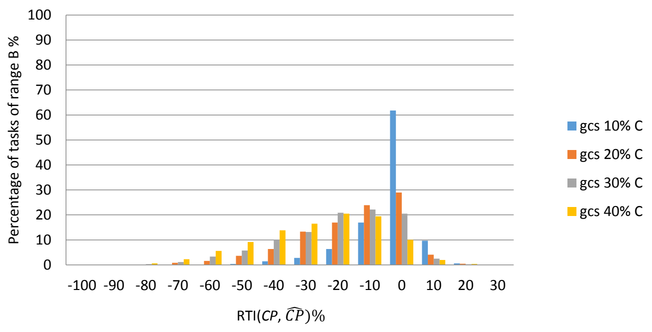

Figure 9: Distribution of tasks according to response time improvement under CP compared to $\widehat{CP}$ for different gcs lengths.

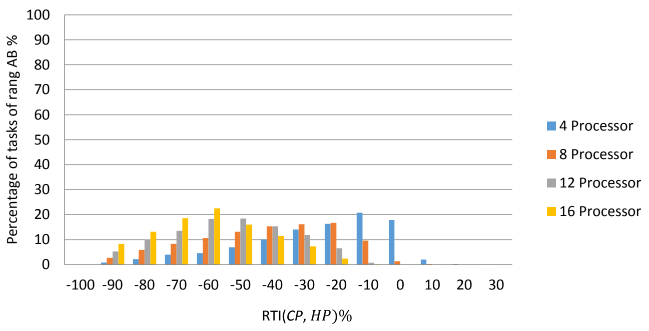

Figure 10: Distribution of tasks according to response time improvement under CP compared to HP for different numbers of processors.

as $\widehat{CP}$ does, since $\rho_{P_k}^{\text{spin}}$ is at higher priority level under HP compared to CP similar as $\widehat{CP}$ compared to CP. Thus the above discussions also hold. However, it can be seen that the distribution of tasks is smoother when comparing CP and HP than when comparing CP and $\widehat{CP}$. It is interesting to observe that some tasks have improvements under the CP approach compared to HP which are presented by negative RTI values in the graphs while some tasks have improvement in their response times under the HP approach compared to the CP approach which are presented by positive RTI values in the related graphs. This confirms the claim in [1] of incomparability of CP and HP spin-lock approaches.

9.2.3 *Evaluation results of $\widehat{CP}$ versus HP*

From Figure 15 it can be seen that by increasing the number of processors the performance of HP compared to $\widehat{CP}$ decreases. The reason is that $spin_{P_k,q}$ increases for HP by increasing the number of processors. Similarly, as can be seen in Figure 19, increasing the global critical section length will also decrease the performance under the HP approach. This is due to the fact that an increase in the global critical section length also leads to an increase in $spin_{P_k,q}$, however the increase is less compared to when the number of processors is increased. Figures 16, 17 and 18 show that changing the number of tasks in the task set, the task set utilization or the local critical section length does not cause a change in the performance. The reason is that , as described in Lemma 2, using HP or $\widehat{CP}$ approach will only make a difference for tasks in range A, and for a task in this priority range, the term $B_{i,j}^{\text{L}}$ is zero. This is due to the fact that tasks do not use any resource in this range. Thus changing the aforementioned parameters do not affect the performance

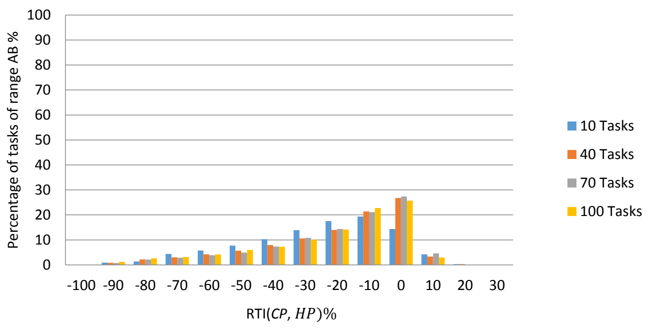

Figure 11: Distribution of tasks according to response time improvement under CP compared to HP for different numbers of tasks.

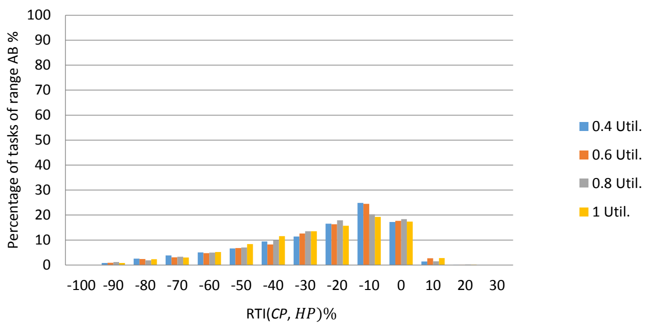

Figure 12: Distribution of tasks according to response time improvement under CP compared to HP for different task set utilizations.

of any of the approaches.

The interesting observation here is that there are no positive RTI values in any of the graphs in this section. The reason has already been mentioned in Lemma 21. Since $\widehat{CP}$ dominates HP, response times cannot be improved under the HP compared to $\widehat{CP}$.

10. CONCLUSION AND FUTURE WORK

In this paper, we investigated different alternatives of spin-lock priorities for tasks on a multi-core platform with the aim to improve the response times of a set of tasks on a core. We assumed a fixed spin-lock priority per core and focused on spin-lock priorities higher than or equal to the highest local ceiling of global resources on a core. The spin-lock approaches corresponding with the boundaries of this range of priorities, denoted by CP (ceiling priority) and HP (highest priority), respectively, are known to be incomparable [1], i.e. neither approach dominates the other. We presented a new spin-lock approach, called $\widehat{CP}$, with accompanying schedulability analysis. $\widehat{CP}$ is based on a spin-lock priority equal to the maximum of the local ceilings of global and local resources. We proved by means of both analysis and evaluation results that $\widehat{CP}$ dominates HP and showed that $\widehat{CP}$ and CP are incomparable. To enable a fair comparison between $\widehat{CP}$ and CP, we improved on the existing schedulability analysis for CP and presented a unified analysis blocking terms in the range of spin-lock priorities [CP, HP].

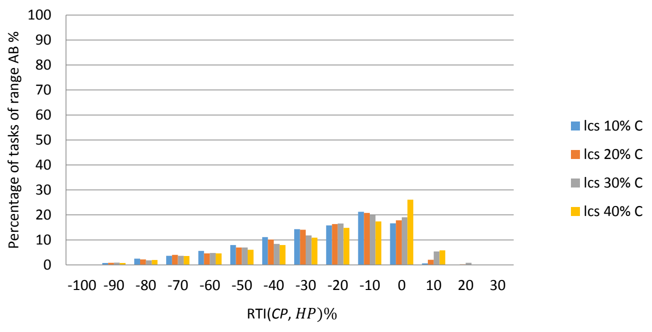

Figure 13: Distribution of tasks according to response time improvement under CP compared to HP for different lcs lengths.

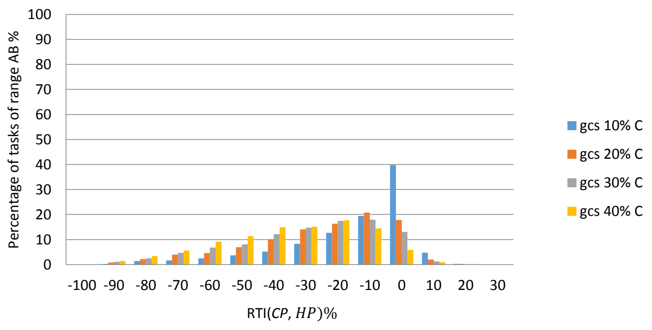

Figure 14: Distribution of tasks according to response time improvement under CP compared to HP for different gcs lengths.

Finally, we showed that if a task set is unschedulable under both CP and $\widehat{CP}$ on a processor, there may exist a spin-lock approach that uses a priority level in between the priorities used by CP and $\widehat{CP}$ which can make the task set schedulable. We showed that the complexity of finding this spin-lock approach is linear and can be a small value.

We have shown by means of experimental results that although the $\widehat{CP}$ and CP approaches are incomparable, under specific system configurations tasks can obtain up to 70% improvement in their response times under the CP approach compared to the $\widehat{CP}$ spin-lock approach. Similarly, under the CP spin-lock approach tasks can gain up to 90% improvements in their response times compared to the traditional HP approach, although it has been shown in [1] that they are incomparable. It can be viewed from the evaluation results that in general, more tasks can have shorter response times under the CP spin-lock approach compared to HP and $\widehat{CP}$ approaches. Further, experimental results confirmed that $\widehat{CP}$ dominates the HP approach and showed up to 90% improvement in response time of tasks under $\widehat{CP}$.

Towards optimizing the spin-lock priority 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 priority levels where spinning happens at a priority level equal to or higher than the highest local ceiling of the global resources accessed on a processor. We leave the later steps as future work.

11. REFERENCES

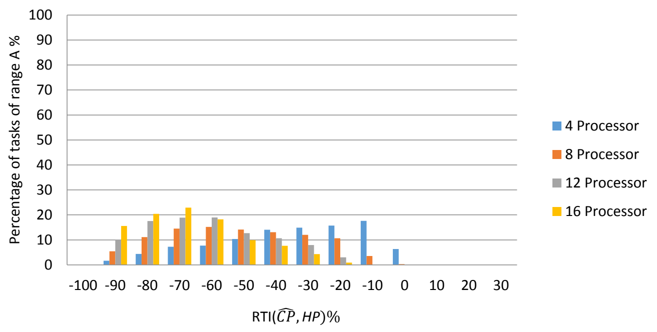

Figure 15: Distribution of tasks according to response time improvement under HP compared to $\widehat{CP}$ for different numbers of processors.

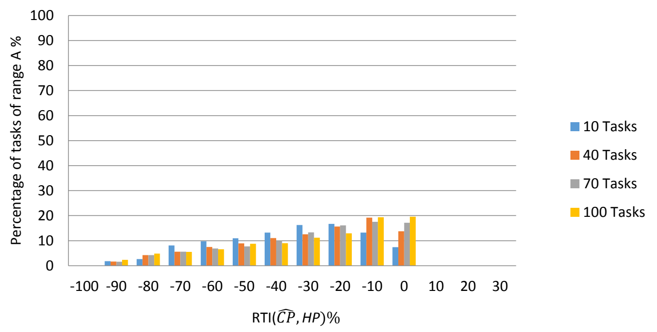

Figure 16: Distribution of tasks according to response time improvement under HP compared to $\widehat{CP}$ for different numbers of tasks.

[1] S. Afshar, M. Behnam, R. Bril, and T. Nolte. Flexible spin-lock model for resource sharing in multiprocessor real-time systems. In 9th *International Symposium on Industrial Embedded Systems (SIES)*.

[2] S. Afshar, N. M. Khalilzad, F. Nemati, and T. Nolte. Resource sharing among prioritized real-time applications on multiprocessors. In 6th *International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, Dec. 2013.

[3] N. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real-time scheduling: The deadline-monotonic approach. In *IEEE Workshop on Real-Time Operating Systems and Software*, pages 133–137, 1991.

[4] T. Baker. Stack-based scheduling of real-time processes. *Journal of Real-Time Systems*, 3(1):67–99, 1991.

[5] E. Bini and G. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[6] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A flexible real-time locking protocol for multiprocessors. In 13th *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, (RTCSA)*, pages 47–56, Aug 2007.

[7] B. Brandenburg and J. Anderson. An implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP real-time synchronization protocols in LITMUS^{RT}. In 14th *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages

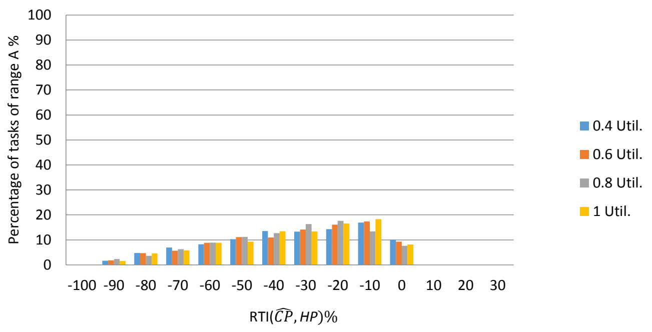

Figure 17: Distribution of tasks according to response time improvement under HP compared to $\widehat{CP}$ for different task set utilizations.

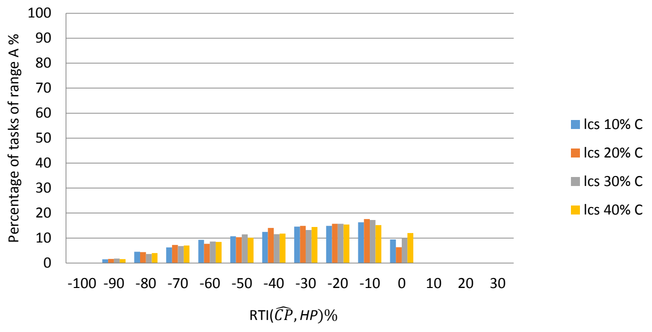

Figure 18: Distribution of tasks according to response time improvement under HP compared to $\widehat{CP}$ for different lcs lengths.

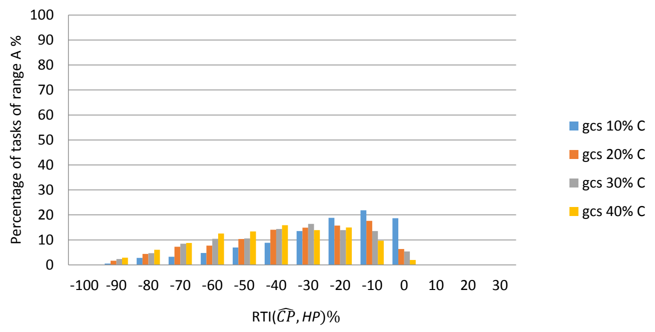

Figure 19: Distribution of tasks according to response time improvement under HP compared to $\widehat{CP}$ for different gcs lengths.

185–194, Aug. 2008.

[8] B. Brandenburg and J. Anderson. Optimality results for multiprocessor real-time locking. In 31st *IEEE Real-Time Systems Symposium (RTSS)*, pages 49–60, Dec. 2010.

[9] B. B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2011.

[10] R. Davis and M. Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In 33^rd *IEEE Real-Time Systems Symposium (RTSS)*, pages 39–50, Dec 2012.

[11] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35:1–35:44, Oct. 2011.

[12] D. Faggioli, G. Lipari, and T. Cucinotta. The multiprocessor bandwidth inheritance protocol. In 22nd *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 90–99, July 2010.

[13] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In 9th *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS).*, pages 189–198, May 2003.

[14] P. Gai, G. Lipari, and M. Di Natale. Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip. In 22nd *IEEE Real-Time Systems Symposium (RTSS)*, pages 73–83, Dec 2001.

[15] K. Lakshmanan, D. de Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In 30th *IEEE Real-Time Systems Symposium, (RTSS)*, pages 469–478, Dec. 2009.

[16] L. Ming. Scheduling of the inter-dependent messages in real-time communication. In *International Workshop on Real-Time Computing Systems and Applications*, 1994.

[17] F. Nemati, M. Behnam, and T. Nolte. Independently-developed real-time systems on multi-cores with shared resources. In 23rd *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 251–261, Jul. 2011.

[18] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In 10th *International Conference on Distributed Computing Systems (ICDCS).*, pages 116–123, May 1990.

[19] R. Rajkumar. *Synchronization in Real-Time Systems: A Priority Inheritance Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[20] R. Rajkumar, L. Sha, and J. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Real-Time Systems Symposium (RTSS)*, pages 259–269, Dec 1988.

[21] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, Sep 1990.

[22] A. Wieder and B. Brandenburg. On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks. In 34th *IEEE International Real-Time Systems Symposium, (RTSS)*, pages 45–56, Dec 2013.