On the Complexity of Worst-Case Blocking Analysis of Nested Critical Sections

Alexander Wieder Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS)

***Abstract*—Accurately bounding the worst-case blocking for finite job sets, a special case of the classic sporadic task model of recurrent real-time systems, using either nested FIFO- or priority-ordered locks on multiprocessors is NP-hard. These intractability results are obtained with reductions from the Multiple-Choice Matching problem. The reductions are quite general and do not depend on (1) whether the locks are spin- or suspension-based, or (2) whether global or partitioned scheduling is used, or (3) which scheduling policy is employed (as long as it is work-conserving).**

Further, we show that, for a special case in which the blocking analysis problem is NP-hard for FIFO- and priority-ordered locks, the problem for unordered spin locks with nested critical sections can be answered in polynomial time by solving a reachability problem on a suitably constructed graph, although (or rather, because) unordered locks do not offer any acquisition-order guarantees.

Finally, we identify several challenging open problems, pertaining both to circumventing the hardness results and to classifying the inherent difficulty of the problem more precisely.

I. Introduction

The classic mechanism used in real-time systems to ensure mutually exclusive access to *shared resources* such as shared storage, data structures, or I/O devices is *locking*, where conflicting accesses are serialized and jobs wait until it is their turn to access a contended resource. As a result, jobs are *blocked* upon requesting access to a resource that is already in use. Depending on the type of lock, blocked jobs either busy-wait (in the case of *spin locks*) or self-suspend (in the case of suspension-based locks, or *semaphores*) until access to the resource is granted.

Regardless of whether jobs spin or suspend, in the context of hard real-time systems, it is paramount that the worst-case duration of blocking incurred by any job is bounded *a priori*, lest the temporal correctness of the system is violated at runtime by unanticipated delays. The resulting *worst-case blocking analysis problem* is thus of fundamental importance to real-time systems that use locks—which is to say virtually any practical real-time system of non-trivial engineering complexity.

Case in point, our specific interest in the blocking analysis problem is motivated by the fact that, on multicore platforms, the AUTOSAR OS standard for automotive systems [1] mandates spin locks for inter-core synchronization. In recent work [28], we presented worst-case blocking analysis for various types of *non-nested* spin locks, including FIFO- and priority-ordered spin locks as well as unordered spin locks.

As the logical next step, we sought to extend our work to support accurate analysis of nested locks; however, this turned out to be much more challenging than anticipated. In fact, as we report in this paper, the analysis of FIFO- and priority-ordered locks is NP-hard if nested critical sections are allowed, and hence the problem is computationally intractable, unless $P = NP$. We obtained these results with reductions using only a single job per processor, and hence our findings are independent of the employed scheduler, whether preemptions are allowed, and whether blocking is realized with spinning or suspending.

Interestingly, in the same setting of job sets with dedicated processors, we find that the worst-case blocking analysis problem for the less desirable class of unordered locks—undesirable for use in practical systems because urgent jobs may incur significant delays due to starvation effects—can be solved in time polynomial in the number of jobs and critical sections.

In short, the hardness results presented herein show that non-pessimistic blocking bounds for the types of locks favored in practical multiprocessor real-time systems may have prohibitively high costs. This observation is a perhaps unpleasant surprise, as the economic incentives and severe resource constraints typically encountered in embedded applications generally make any pessimism highly undesirable. More easily analyzed, yet still well-performing alternatives to FIFO- and priority-ordered locks would thus arguably be desirable. To this end, we identify several interesting open problems, pertaining both to circumventing the hardness results and to classifying the inherent hardness more precisely, which we discuss in Sec. VI after formally establishing the claimed results in Secs. III–V.

We begin by introducing essential notation and defining our system model and assumptions.

II. Definitions and Assumptions

We next present the model that we assume throughout this work, review the multiple-choice matching problem used in our reductions, and precisely define the blocking analysis problem.

A. Jobs and Shared Resources

In this work, we consider a simplified variant of the sporadic task model. The sporadic task model considers a set of *tasks* that release *jobs* to be executed. For the sake of simplicity, we consider only a finite set of jobs, as the concept of tasks is not required to obtain the results presented in this paper. Analytically, the finite set of jobs can be considered as a special case of the sporadic task model and hence our results trivially extend to more expressive task models.

We denote the jobs in the system as $J_1, \ldots, J_n$ and consider their release times to be unknown (*i.e.*, as in the sporadic task model, the exact release times are discovered only at runtime). With multiple jobs, a *scheduling policy* has to decide which job is executed at any time. Since our reductions use only a single job per processor, we do not make any assumptions about the scheduling policy as long as each job is scheduled upon release until completion. We also do not make any assumptions on whether job preemptions are allowed (since no preemptions can occur with a single job per processor). Although our reductions generalize to other scheduling policies as well, we assume partitioned scheduling for the sake of simplicity.

Each job may access *shared resources*, which can be shared data structures, sensors, or other peripheral devices. The shared resources are serially reusable, that is, any resource can be accessed by at most one job at any time. We denote the shared resources in the system as $\ell_1, \ldots, \ell_{n_r}$, the set of all resources as Q, and their number as n_r. A request for resource ℓ_q issued by job J_x is denoted as $R_{x,q,s}$, where s is an index to distinguish multiple requests for ℓ_q issued by J_x. Note that s does not imply any particular order in which J_x is assumed to issue its requests. When the request $R_{x,q,s}$ for ℓ_q is granted, the job J_x executes a *critical section* of length at most $L_{x,q,s}$. We further let $L_{x,q}$ denote the maximum length of any request that J_x issues for ℓ_q, that is, $L_{x,q} = \max_s\{L_{x,q,s}\}$. For simplicity, we assume discrete time, and hence all time intervals and all bounds on critical section lengths have an integral length.

Requests can be *nested*, that is, a job holding a resource ℓ_q can issue a request for a different resource ℓ_p (where $p \neq q$) within the critical section accessing ℓ_q. All requests are *properly nested*, that is, at any time, only the resource that was acquired last and is still held can be released. To denote the nesting relation of requests we introduce the following notation: $R_{x,q,s} \triangleright R_{x,q',s'}$ denotes that the request $R_{x,q',s'}$ for $\ell_{q'}$ is directly nested (*i.e.*, not transitively) within the request $R_{x,q,s}$ for ℓ_q. For requests containing multiple nested requests, we use the following set notation: $R_{x,q,s} \triangleright \{R_{x,q'_1,s'_1}, \ldots, R_{x,q'_w,s'_w}\} \Leftrightarrow \forall 1 \leq j \leq w : \quad R_{x,q,s} \triangleright R_{x,q'_j,s'_j}$. For example, we express that two requests $R_{x,p,t}$ and $R_{x,p',t'}$ are nested within the request $R_{x,q,s}$ as $R_{x,q,s} \triangleright \{R_{x,p,t}, R_{x,p',t'}\}$.

We do not make any assumptions about the order of nested requests as long as the nesting relation as described above is preserved. To rule out deadlock, we assume the existence of a partial order $<$ on resources such that, for any two nested requests $R_{x,q,s}$ and $R_{x,q',s'}$, if $R_{x,q,s} \triangleright R_{x,q',s'}$ then $\ell_q < \ell'_q$. We assume that the critical section length of each request accounts for nested requests, but not for any blocking that might be incurred on resource contention. That is, the critical section length includes the lengths of all nested requests: $\forall R_x : L_x \geq \sum_{R_y, R_x \triangleright R_y} L_y$.

Since access to the shared resources can only be granted in mutual exclusion, a job issuing a request can be *blocked* by concurrent requests for the same resource. In this work, we consider *locks* in which jobs waiting for a contended resource may either busy-wait or suspend until gaining access. Since we assign only a single job to each processor, both options are analytically equivalent and our results apply to both types of locks. The way in which competing requests are served is defined by an *ordering policy* specific to the lock type used to protect the shared resources. In this work, we consider FIFO-ordered, priority-ordered, and unordered locks.

With FIFO-ordered locks, requests are served in the order in which they are issued (with ties broken arbitrarily), which ensures a straightforward property that is key to our reduction.

Lemma 1. *If a resource ℓ_q is protected by a FIFO-ordered lock, then a request $R_{x,q,s}$ issued by a job J_x for ℓ_q can be blocked by at most one request for ℓ_q from each other job in the system.*

Proof. Follows trivially since jobs are sequential and since later-issued requests cannot block in a FIFO queue. ■

In our reductions, we assign only a single job to each processor. In this setting, Lem. 1 also implies that each request can be blocked by at most one request from each other processor.

Priority-ordered locks consider a *locking priority* for each request and ensure that each request is blocked by at most one request for the same resource with lower locking priority.

Finally, unordered locks do not ensure any specific ordering of requests, and hence a request can be blocked by all concurrent requests for the same resource when unordered locks are used.

Next, we briefly introduce the *Multiple-Choice Matching Problem*, which we use in our reductions.

B. The Multiple-Choice Matching Problem

We show that the blocking analysis problem for task sets with shared resources and nested critical sections using a work-conserving partitioned scheduler is NP-hard. To this end, we reduce the *multiple-choice matching (*MCM*)* [13] problem, which is known to be NP-complete [13], to an instance of the blocking analysis problem studied herein.

For simplicity, we represent an undirected edge e between two vertices v_1 and v_2 as the set of its endpoints: $e = \{v_1, v_2\}$. The MCM problem is then defined as follows: given a positive integer k and an undirected graph $G = (V, E)$, where the set of edges E is partitioned into t pairwise disjoint subsets (*i.e.*, $E = E_1 \cup \cdots \cup E_t$), does there exists a subset $F \subseteq E$ with $|F| \geq k$ such that

- no two edges in F share the same endpoint: $\forall e_1, e_2 \in F, e_1 \neq e_2 : \; e_1 \cap e_2 = \emptyset$; and
- F contains at most one edge from each edge partition: $\forall i, 1 \leq i \leq t : \; |F \cap E_i| \leq 1$?

Note that a solution exists only if $k \leq t$. In Appendix A, we show that instances of the MCM problem with $k < t$ can be reduced to instances with $k = t$ without loss of generality. In the following, we hence use instances of the MCM problem with $k = t$ unless noted otherwise. Next, we formalize the problem of bounding the blocking that a job incurs in the worst case.

C. The Worst-Case Blocking Analysis Problem

For real-time tasks with strict timing requirements, the worst-case blocking duration of each task must be bounded a priori to ensure that all timing requirements are met. Such worst-case blocking bounds can be derived with a *blocking analysis*.

A trivial bound can easily be obtained by assuming that all requests issued while a job is pending can contribute to its blocking. Albeit valid, such a bound is clearly pessimistic and of limited use in practice. Therefore, we require the blocking analysis to yield tighter bounds that are more meaningful for actual applications. In particular, we require that:

- There exists no job arrival sequence and resulting schedule in which more blocking than determined by the analysis is incurred. (The bound is safe.)
- There exists a job arrival sequence and resulting schedule in which the blocking duration determined by the analysis is incurred. (The bound is tight.)

We denote the problem of computing blocking bounds as the the blocking analysis *optimization problem* BO and the outcome of the blocking analysis for a job J_i as $B_i = \mathrm{BO}(J_i)$. The corresponding blocking analysis *decision problem* is denoted as $\mathrm{BD}(J_i, B_i)$, which is the problem of deciding if there exists a job arrival sequence and resulting schedule in which J_i can be blocked for at least B_i time units.

The blocking analysis optimization problem can be reduced to the decision variant within polynomial time, and vice versa. Given the solution to the optimization problem $\mathrm{BO}(J_i)$, solutions to the decision problem $\mathrm{BD}(J_i, B_i)$ can be trivially obtained by returning *yes* if and only if $\mathrm{BO}(J_i) \geq B_i$.

Given an oracle for the blocking analysis decision problem $\mathrm{BD}(J_i, B_i)$, the solution to the optimization problem can be obtained by finding the maximal integral value of B'_i for which $\mathrm{BD}(J_i, B_i)$ evaluates to *yes*. This can be achieved by repeatedly evaluating $\mathrm{BD}(J_i, B_i)$ within a binary search over the interval $[0, B_i^{max}]$, where B_i^{max} is a trivial upper bound on the blocking that J_i can incur (e.g., the sum of all critical section lengths). Note that B_i^{max} grows exponentially with respect to the *size* of the problem instance c: $B_i^{max} = O(2^c)$. Here, c denotes the size of the binary representation of the problem instance. Since the binary search terminates after $O(\log_2 B_i^{max})$ steps, computing the solution to the optimization problem takes overall $O(\log_2 B_i^{max}) = O(\log_2 2^c) = O(c)$ steps with respect to the size of the problem instance c.

On uniprocessors, the blocking analysis optimization problem is trivial for many practical lock types: when using either non-preemptive critical sections, the *priority ceiling protocol* (PCP) [25], or the *stack resource policy* (SRP) [2], the worst-case blocking incurred by any job is generally limited to the length of one outermost critical section, and tight per-job bounds are easy to find. Similarly, appropriate blocking bounds under the *priority inheritance protocol* (PIP) [25] can be found using a simple dynamic programming approach (*e.g.*, see [18]).

On multiprocessors, however, the blocking analysis optimization problem for FIFO- or priority-ordered locks is NP-hard in the presence of nested critical sections, as we show in this paper. For brevity, we denote the blocking analysis decision problems for FIFO-ordered and priority-ordered locks as BD_F and BD_P, respectively. Further, we denote the blocking that a job J_x incurs in a particular schedule S (resulting from a particular job arrival sequence) as $B_x(S)$. We begin by reducing instances of the MCM problem to the BD_F problem.

III. Reduction of MCM to BD_F

In this section, we show that an algorithm that solves the blocking analysis problem for FIFO-ordered locks can be used to solve the MCM problem. Given an MCM problem, we construct a set of jobs issuing nested requests such that the worst-case blocking duration B_i encodes the answer to the MCM problem. Next, we define the jobs and requests used in the reduction.

A. An example BD_F instance

At a high level, the construction of the BD_F instance is best illustrated with an example. Consider the graph G_1 in Fig. 1. The corresponding BD_F instance is shown in Fig. 2(a).

We model vertices as shared resources and edges as nested requests. More specifically, edges are encoded as a request to a "dummy resource" ℓ_D that contains two nested requests to the resources representing the endpoints of the edge.

The two edge partitions in G_1 (shown as dashed or solid edges in Fig. 1) correspond to processors p_1 and p_2 on which two jobs J_1 and J_2 issue the requests that model the edges in G_1.

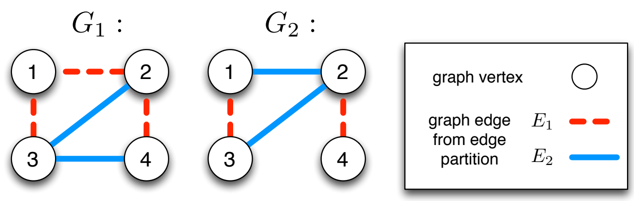


Fig. 1. Two example graphs of MCM problem instances. With $k = t = 2$, a matching solving the MCM problem for G_1 exists: $\{\{1, 2\}\{3, 4\}\}$. For G_2 no such matching exists.

The job J_3 on processor p_3 serves as a "probe": by solving the BD_F problem for J_3, which accesses only the dummy resource ℓ_D, we can infer whether G_1 admits an MCM of size two. Finally, the job J_4 on processor p_4 serves to *transitively* block J_3 by creating contention for all resources corresponding to vertices in G_1, as explained in more detail below.

B. Construction of the BD_F instance

Formally, given an MCM instance that consists of a graph $G = (V, E)$, t disjoint edge partitions $E_1, \ldots, E_t$ such that $E_1 \cup \cdots \cup E_t = E$, and $k = t$ (without loss of generality, see Appendix A), we construct a BD_F instance as follows.

For each vertex $v \in V$, there is one shared resource ℓ_v. In addition, there is a single *dummy resource* ℓ_D. We consider $t+2$ processors, $p_1, \ldots, p_{t+2}$, and $t+2$ jobs, $J_1, \ldots, J_{t+2}$, where each job J_j with $1 \leq j \leq t+2$, is assigned to processor p_j.

We construct requests with two basic critical section lengths: there are *short* and *long* critical sections, with the corresponding lengths of $\Delta_S \triangleq 1$ and $\Delta_L \triangleq 2 \cdot |V|$, respectively.

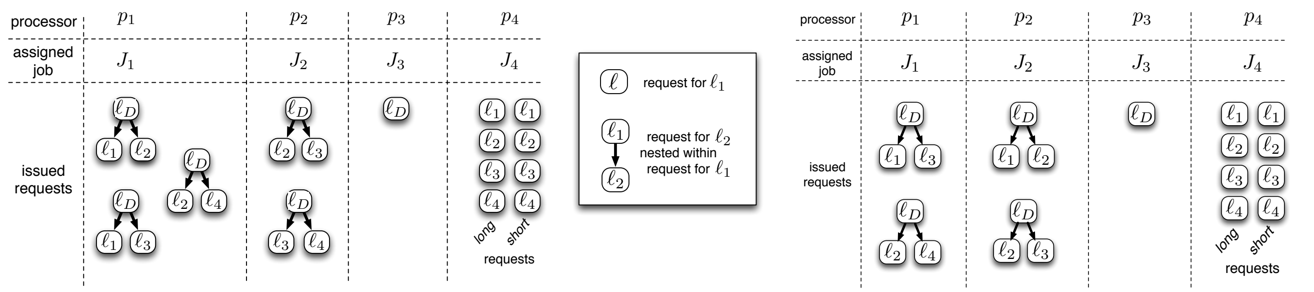
The jobs $J_1, \ldots, J_t$ issue requests for the dummy resource ℓ_D with nested requests to model edges, J_{t+1} issues a single request for ℓ_D, and J_{t+2} issues a short request (of length Δ_S) and a long request (of length Δ_L) for each resource ℓ_v corresponding to a vertex $v \in V$. More formally, the jobs issue requests as follows.

- Jobs $J_1, \ldots, J_t$: For each edge $e_i = \{v, v'\}$ in the edge partition E_j, job J_j issues three requests: one request $R_{j,D,i}$ for ℓ_D, one request $R_{j,v,i}$ for ℓ_v, and one request $R_{j,v',i}$ for $\ell_{v'}$. The critical section lengths are $L_{j,D} = 2 \cdot \Delta_L$, $L_{j,v} = \Delta_L$, and $L_{j,v'} = \Delta_L$, respectively. The requests are nested such that $R_{j,D,i} \triangleright \{R_{j,v,i}, R_{j,v',i}\}$.
- Job J_{t+1} issues one non-nested request $R_{t+1,D,1}$ for ℓ_D with critical section length $L_{t+1,D} = 1$.
- Job J_{t+2} issues for each resource ℓ_v with $v \in V$ two non-nested requests: $R_{t+2,v,1}$ and $R_{t+2,v,2}$. The critical section lengths are $L_{t+2,v,1} = \Delta_L$ and $L_{t+2,v,2} = \Delta_S$, respectively.

As the number of constructed jobs is linear in $t \leq |E|$ and the number of constructed requests is linear in $|V|$, the reduction of the MCM instance to an BD_F instance requires only polynomial time with respect to the size of the input graph.

C. Basic idea: J_{t+1}'s maximum blocking implies MCM answer

Recall that for a solution to the MCM problem to exist, there must be k matched edges, and each vertex in the graph must be adjacent to at most one matched edge. As we illustrate next with an example, this is equivalent to requiring that, in a schedule S in which J_{t+1} incurs the maximum blocking possible (*i.e.*, $B_{t+1}(S) = B_{t+1}$), J_{t+1} is transitively blocked in S by J_{t+2}



(a) BD_F problem constructed from G_1 and $k = t = 2$.

(b) BD_F problem constructed from G_2 and $k = t = 2$.

Fig. 2. BD_F problems constructed from G_1 and G_2.

with *exactly* $2k$ of its long critical sections and *none* of its short critical sections. Whether this is in fact the case can be inferred from B_{t+1} due to the specific values chosen for Δ_S and Δ_L.

Returning to the example BD_F instance shown in Fig. 2(a), note how the vertices $v_1, \ldots, v_4$ in G_1 correspond to the shared resources $\ell_1, \ldots, \ell_4$ in Fig. 2(a), and how edges in G_1 map to nested requests issued by J_1 and J_2. For instance, the dashed edge $\{1, 2\}$ in G_1 is represented as a request for ℓ_D issued by J_1 (which corresponds to E_1) that contains nested requests for ℓ_1 and ℓ_2. Similarly, the remaining dashed edges $\{1, 3\}$ and $\{2, 4\}$ are also represented by nested requests issued by J_1. The solid edges $\{2, 3\}$ and $\{3, 4\}$ are represented by similar requests issued by J_2 (which corresponds to E_2).

Crucially, all requests for the resources $\ell_1, \ldots, \ell_4$ issued by J_1 and J_2 are nested within a request for ℓ_D. This ensures that **(i)** J_3 can be transitively delayed by J_4's requests and that **(ii)** J_1 and J_2's requests for $\ell_1, \ldots, \ell_4$ cannot block each other since ℓ_D must be held in order to issue these requests.

Consider the worst case for J_3, which is also illustrated in Fig. 3(a): J_3's request for ℓ_D is delayed by one (outer) request for ℓ_D from both J_1 and J_2 each, and the nested requests issued by J_1 and J_2 are in turn blocked by requests issued by J_4, which transitively delays J_3. Importantly, the total delay incurred by J_3 in the worst case is determined by *which requests of J_4 cause transitive blocking*—since J_4 accesses each $\ell_1, \ldots, \ell_4$ with a long critical section only once, J_4 can transitively delay J_3 for $4 \cdot \Delta_L$ time units *only if* J_4 (indirectly) conflicts with J_3 via four (*i.e.*, $2 \cdot k$) *distinct* resources.

In other words, if B_3 indicates that J_4 can transitively delay J_3 for $4 \cdot \Delta_L$ time units, then there exists a way to choose one outer request of J_1 (*i.e.*, an edge from E_1) and one outer request of J_2 (*i.e.*, an edge from E_2) such that the nested requests of J_1 and J_2 access four distinct resources (*i.e.*, no vertex is adjacent to both edges), which implies the existence of a valid MCM.

We illustrate this correspondence with two examples. For G_1 and $k = t = 2$, a valid MCM F indeed exists: $F = \{\{1, 2\}, \{3, 4\}\}$. Therefore, as shown in Fig. 3(a), there exists a schedule such that J_3 is blocked for a total of $B_3 = 8 \cdot \Delta_L$ time units, which includes $2 \cdot k \cdot \Delta_L = 4 \cdot \Delta_L$ time units of transitive blocking due to J_4. (The remaining $4 \cdot \Delta_L$ time units are an irrelevant artifact of the construction and due to J_1 and J_2's nested requests.) Hence, $\mathrm{BD}_F(J_3, 8 \cdot \Delta_L) = $ *yes*.

For G_2 with $k = t = 2$, no MCM exists: any combination of one dashed and one solid edge necessarily has one vertex in common. This is reflected in the derived BD_F instance, which is shown in Fig. 2(b). Job J_3 can be blocked for at most $7 \cdot \Delta_L + \Delta_S$ time units in total, as Fig. 3(b) illustrates, but not for $8 \cdot \Delta_L$ time units. In particular, J_3 is transitively delayed by J_4 for only $3 \cdot \Delta_L + \Delta_S$ time units in the depicted schedule since J_4 blocks J_3 twice with a request for ℓ_1. Hence, $\mathrm{BD}_F(J_3, 8 \cdot \Delta_L) = $ *no*.

In general, we observe that $\mathrm{BD}_F(J_{t+1}, 4 \cdot k \cdot \Delta_L) = $ *yes* if and only if a valid MCM exists. We formalize this argument in Theorem 1 below and begin by establishing essential properties of the constructed set of jobs and requests.

D. Properties of the constructed job set

First, we observe that the lengths of J_{t+2}'s critical sections enable us to infer from J_{t+1}'s blocking bound whether any short requests block J_{t+1} in a worst-case schedule.

Lemma 2. *Consider a schedule S in which J_{t+1} is blocked for $B_{t+1}(S) = B_{t+1}$ time units. If B_{t+1} is an integer multiple of Δ_L, then J_{t+1} is not blocked by any short request in S.*

Proof. By construction, only J_{t+2} issues short requests. In total, J_{t+2} issues $|V|$ short requests, each with a critical section length $\Delta_S = 1$. Therefore, J_{t+1} can be blocked for at most $|V| \cdot \Delta_S = |V|$ time units by these requests. Hence, if one or more short requests block J_{t+1} in S, then $B_{t+1}(S)$ is not an integer multiple of Δ_L as $\Delta_L = 2 \cdot |V| > |V| \cdot \Delta_S$. ■

Next, we establish a straightforward bound on the duration that any request for ℓ_D issued by a job $J_1, \ldots, J_t$ blocks J_{t+1}.

Lemma 3. *Each request for ℓ_D issued by a job J_j, where $1 \leq j \leq t$, blocks J_{t+1} for at most $4 \cdot \Delta_L$ time units.*

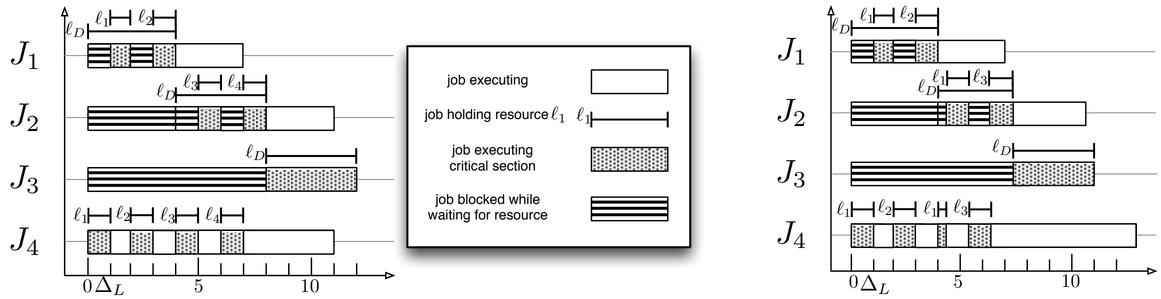
Proof. By construction, each request for ℓ_D from such a job J_j has a length of $2 \cdot \Delta_L$ time units and contains two nested requests for two resources ℓ_{v_1} and ℓ_{v_2}, where $\{v_1, v_2\} \subseteq V$. Also by construction, while J_j holds ℓ_D, it can encounter contention only from J_{t+2} (since all requests issued by jobs $J_1, \ldots, J_t$ are serialized by ℓ_D). In the worst case, each of J_j's nested requests is hence blocked only by J_{t+2}'s matching long request of length Δ_L. J_j thus releases ℓ_D after at most $4 \cdot \Delta_L$ time units. ■

From Lem. 3, we obtain an immediate upper bound on the total blocking incurred by J_{t+1} in any schedule.

Lemma 4. $B_{t+1} \leq 4 \cdot k \cdot \Delta_L$.

Proof. By construction, J_{t+1} issues only a single request for ℓ_D. By Lem. 1, J_{t+1} is blocked by at most one request for ℓ_D from each job J_j with $1 \leq j \leq t$. (J_{t+2} does not access ℓ_D.) By Lem. 3, each of these $t = k$ requests blocks J_{t+1} for at most $4 \cdot \Delta_L$ time units. Hence, $B_{t+1} \leq 4 \cdot k \cdot \Delta_L$. ■

Fig. 3(a) illustrates Lem. 4 for the BD_F instance constructed for G_1. In the depicted schedule, J_3 is blocked in total for $4 \cdot k \cdot$



(a) Schedule for the BD_F problem for G_1 in which J_3 is blocked for $4 \cdot k \cdot \Delta_L = 8 \cdot \Delta_L$ time units.

(b) Schedule for the BD_F problem for G_2 in which J_3 is blocked for $7 \cdot \Delta_L + \Delta_S$ time units.

Fig. 3. Example schedules for the constructed BD_F problems

$\Delta_L = 8 \cdot \Delta_L$ time units, and no other request can further block J_3. Note that none of the resources $\ell_1, \ldots, \ell_4$ is requested more than once within a request for ℓ_D from J_1 or J_2 that blocks J_3. In fact, as we show with the next lemma, this is generally the case if the job J_3 is blocked for $4 \cdot k \cdot \Delta_L$ time units.

Lemma 5. *Let S denote a schedule of the constructed job set. If $B_{t+1}(S) = 4 \cdot k \cdot \Delta_L$, then each resource ℓ_v with $v \in V$ is requested within at most one request for ℓ_D that blocks J_{t+1}.*

Proof. From Lem. 4, it follows that S is a worst-case schedule for J_{t+1}. Hence, if a job J_j with $1 \leq j \leq t$ blocks J_{t+1} with a request for ℓ_D, then each nested request therein encounters contention from J_{t+2}. (Otherwise, S would not be a worst-case schedule.) By Lem. 2, since $B_{t+1}(S)$ is an integer multiple of Δ_L, J_{t+1} is (transitively) blocked only by long requests in S. Since J_{t+2} issues only a single long request for each ℓ_v (with $v \in V$), this implies that each resource ℓ_v with $v \in V$ is requested within at most one request for ℓ_D that blocks J_{t+1}. ■

With Lem. 5 it can be shown that, if $\mathrm{BD}_F(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes*, then there is a matching such that no vertex is adjacent to more than one matched edge. To solve the MCM problem, we additionally have to show that such a matching contains exactly one edge from each edge partition. To this end, we next show that, if $B_{t+1}(S) = 4 \cdot k \cdot \Delta_L$, then exactly one request for ℓ_D (corresponding to an edge) from each of the jobs $J_1, \ldots, J_t$ (each corresponding to an edge partition) blocks J_{t+1}.

Lemma 6. *Let S denote a schedule of the constructed job set. If $B_{t+1}(S) = 4 \cdot k \cdot \Delta_L$, then each J_j with $1 \leq j \leq t$ blocks J_{t+1} with exactly one request for ℓ_D.*

Proof. By Lem. 1, each of the t jobs $J_1, \ldots, J_t$ can block J_{t+1} in S with at most one request for ℓ_D. (J_{t+2} does not access ℓ_D.) Further, by Lem. 3, a request for ℓ_D by a job J_j with $1 \leq j \leq t$ blocks J_{t+1} for at most $4 \cdot \Delta_L$ time units. Hence, J_{t+1} is blocked by at least $B_{t+1}(S)/4 \cdot \Delta_L = k = t$ such requests in S. Hence, each J_j with $1 \leq j \leq t$ blocks J_{t+1} exactly once in S. ■

With these lemmas in place, we next show that solving the BD_F problem for the constructed instance is equivalent to solving the MCM problem for the input instance.

Theorem 1. *A matching F solving the* MCM *problem exists if and only if* $\mathrm{BD}_F(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ yes.

Proof. We show the following two implications to prove equivalence:

- $\Longrightarrow$: If $\mathrm{BD}_F(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes*, then there exists a matching F solving the MCM problem.
- $\Longleftarrow$: If there exists a matching F solving the MCM problem, then $\mathrm{BD}_F(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes*.

$\Longrightarrow$: By the definition of BD_F, it follows from $\mathrm{BD}_F(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes* that there exists a schedule S such that $B_{t+1}(S) = 4 \cdot k \cdot \Delta_L$. We construct a matching F that solves the MCM problem from the requests for ℓ_D that block J_{t+1} in S.

For each job J_j with $1 \leq j \leq t$, let $R_{j,D,s}$ denote the request for ℓ_D issued by J_j that blocks J_{t+1} in S. For brevity, let $edge(R_{j,D,s})$ denote the edge $\{v_1, v_2\}$ corresponding to $R_{j,D,s}$, and let F contain all edges represented by requests for ℓ_D that block J_{t+1}: $F \triangleq \bigcup_{1 \leq j \leq t} \{edge(R_{j,D,s})\}$.

By Lem. 6, exactly one request for ℓ_D from each job $J_1, \ldots, J_t$ blocks J_{t+1}; F hence contains $|F| = t$ edges in total and exactly one edge per edge partition. Further, by Lem. 5, for each resource ℓ_v with $v \in V$ at most one request for ℓ_v is nested within a blocking request for ℓ_D from any processor. Hence, each vertex $v \in V$ is adjacent to at most one edge in F. Therefore F is a matching solving the MCM problem.

$\Longleftarrow$: Let F be a matching solving the MCM problem for a graph $G = (V, E)$, edge partitions $E_1, \ldots, E_t$, and $k = t$. Consider a schedule S in which J_{t+1} is maximally (*i.e.*, for the full critical section length) blocked by each request for ℓ_D that corresponds to an edge in F. Since F is an MCM in G, F contains exactly one edge from each edge partition. Then, by construction, J_{t+1} is blocked by exactly one request for ℓ_D from each processor $p_j, 1 \leq j \leq t$.

As F is a matching, each vertex $v \in V$ is adjacent to at most one edge in F. Since vertices in the MCM instance correspond to resources in the BD_F instance, each resource ℓ_v with $v \in V$ is requested within at most one request for ℓ_D that blocks J_{t+1} in S. Then each request for ℓ_v with $v \in V$ nested within a blocking request for ℓ_D can be blocked by the long request for ℓ_v issued by J_{t+2}, and thus each blocking request for ℓ_D can block J_{t+1} for $4 \cdot \Delta_L$ time units. Since $k = t$ requests for ℓ_D in total block J_{t+1}, there exists a schedule S such that job J_{t+1} is blocked for $4 \cdot k \cdot \Delta_L$ time units. Then $\mathrm{BD}_F(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes*. ■

As described in Sec. III-B, the construction of the BD_F instance requires only polynomial time with respect to the MCM instance size. Since instances of the MCM decision problem can be solved via reduction to BD_F, and since the MCM problem is NP-complete, it follows that BD_F is NP-hard.

Next, we show that the blocking analysis decision problem for priority-ordered locks in the presence of nested critical sections on multiprocessors is NP-hard as well.

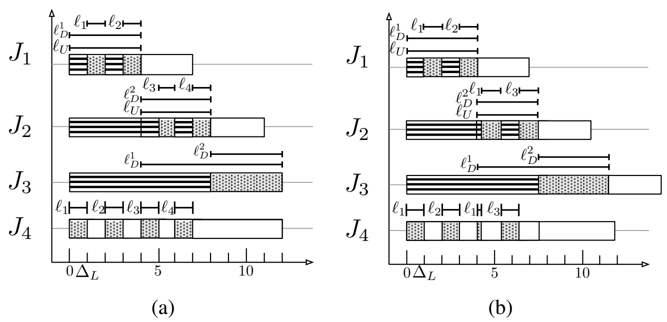


Fig. 5. **(a)** Schedule for the BD_P problem instance for G_1 in which J_3 is blocked for $4 \cdot k \cdot \Delta_L = 8 \cdot \Delta_L$ time units. **(b)** Schedule for the BD_P problem instance for G_2 in which J_3 is blocked for $7 \cdot \Delta_L + \Delta_S$ time units.

IV. Reduction of MCM to BD_P

The reduction to BD_P follows in large parts the same structure as the one for BD_F, but must deal with the slightly weaker progress guarantees offered by priority-ordered locks. With FIFO-ordered locks, each request can be blocked at most once by a request from each other processor (Lem. 1). This fact was exploited to ensure that exactly one edge in each edge partition of a given MCM instance is contained in a matching. Priority-ordered locks, however, do not have this ordering property, and hence the previous approach cannot be used directly. To ensure that one edge per partition is matched, we instead use multiple different dummy resources and an appropriate assignment of request priorities. Next, we explain the approach in detail.

A. Main differences to BD_F *reduction*

At a high level, the constructed BD_P instance is similar to the BD_F reduction, with the following exceptions.

- We use one dummy resource ℓ_D^j for each processor p_j with $1 \leq j \leq t$ (instead of the single global ℓ_D in BD_F).
- The job J_{t+1} issues a request for each dummy resource ℓ_D^j (instead of a single request for ℓ_D in BD_F).
- Each job J_j with $1 \leq j \leq t$ issues requests for the "local" dummy resource ℓ_D^j (instead of for the global ℓ_D in BD_F).
- An additional resource ℓ_U serializes requests of the jobs $J_1, \ldots, J_t$: each job J_j's requests for the dummy resource ℓ_D^j (with $1 \leq j \leq t$) are nested in a request for ℓ_U.

Figs. 4(a) and 4(b) show the BD_P instances constructed for the graphs G_1 and G_2, respectively, as given in Fig. 1.

The basic idea of the reduction of MCM to BD_P is the same as for the reduction to BD_F: the solution to the MCM problem can be inferred from J_{t+1}'s blocking bound. We illustrate the reduction of MCM to BD_P with two examples.

Recall that for graph G_1 and $k = t = 2$, a matching F solving the MCM problem exists: $F = \{\{1, 2\}, \{3, 4\}\}$. In the BD_P instance constructed for G_1 shown in Fig. 4(a), J_3 is blocked for $8 \cdot \Delta_L$ in the worst case, just as it is the case in the reduction to the BD_F problem presented in the previous section. Fig. 5(a) depicts a schedule in which J_3 incurs the worst-case blocking of $8 \cdot \Delta_L$. Notably, J_3 is not blocked by any short requests issued by J_4. As in the reduction to the BD_F problem, J_3 can only be blocked for $8 \cdot \Delta_L$ time units if no short requests block J_3, and no solution to the given MCM problem exists if any short requests block J_3 in a worst-case schedule.

We illustrate this property with the MCM problem for G_2 and $k = t = 2$, for which no solution exists. In the constructed BD_P instance for G_2 (shown in Fig. 4(b)), J_3 can thus be blocked for at most $7 \cdot \Delta_L + \Delta_S$ time units, as illustrated in Fig. 5(b).

In general, as we argue in the following, a matching solving an MCM problem exists if and only if, in the constructed BD_P instance, job J_{t+1} can be blocked for $4 \cdot k \cdot \Delta_L$ time units, and hence $\mathrm{BD}_P(J_{t+1}, 4 \cdot k \cdot \Delta_L) = $ *yes*.

B. Construction of the BD_P *instance*

Formally, given an MCM instance consisting of a graph $G = (V, E)$ and $k = t$ pairwise disjoint edge partitions $E_1, \ldots, E_t$ we construct a BD_P instance as follows.

There is one shared resource ℓ_v for each vertex $v \in V$. Instead of the single dummy resource in the construction for BD_F, there is one dummy resource ℓ_D^j for each processor p_j with $1 \leq j \leq t$, and an additional dummy resource ℓ_U. As in the BD_F reduction, there are $t+2$ processors $p_1, \ldots, p_{t+2}$ and $t+2$ jobs $J_1, \ldots, J_{t+2}$, where each such job J_j (with $1 \leq j \leq t+2$) is assigned to the corresponding processor p_j.

As in the BD_F reduction, the critical sections of these jobs are either short (*i.e.*, of length $\Delta_S \triangleq 1$) or long (*i.e.*, of length $\Delta_L \triangleq 2 \cdot |V|$), and graph edges are modeled as nested requests. In contrast to the reduction to BD_F, where all of these requests were nested within a request for the single dummy resource ℓ_D, the requests modeling an edge from edge partition E_j are nested within a request for the dummy resource ℓ_D^j. Further, each request for ℓ_D^j issued by a job J_j with $1 \leq j \leq t$ is nested within a request for ℓ_U. The jobs issue requests as follows.

- Jobs $J_1, \ldots, J_t$: For each edge $e_i = \{v, v'\}$ in the edge partition E_j, the job J_j issues four requests: one request $R_{j,U,i}$ for ℓ_U, one request $R_{j,D^j,i}$ for ℓ_D^j, one request $R_{j,v,i}$ for ℓ_v, and one request $R_{j,v',i}$ for $\ell_{v'}$, where $R_{j,U,i} \triangleright R_{j,D^j,i} \triangleright \{R_{j,v,i}, R_{j,v',i}\}$, and $L_{j,U} = 2 \cdot \Delta_L$, $L_{j,D^j} = 2 \cdot \Delta_L$, $L_{j,v} = \Delta_L$, and $L_{j,v'} = \Delta_L$.
- Job J_{t+1} issues one non-nested request $R_{t+1,D^j,1}$ for each dummy resource ℓ_D^j (where $1 \leq j \leq t$) with $L_{t+1,D^j} = 1$.
- Job J_{t+2} issues for each resource ℓ_v (where $v \in V$) two non-nested requests $R_{t+2,v,1}$ and $R_{t+2,v,2}$, where $L_{t+2,v,1} = \Delta_L$ and $L_{t+2,v,2} = \Delta_S$.

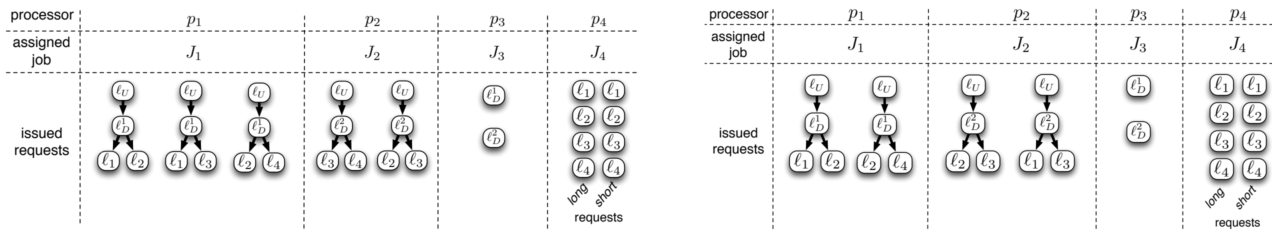
Since we use priority-ordered locks in the construction of the BD_P instance, a priority has to be assigned to each request. We use three priority levels: *high*, *medium*, and *low*. The requests issued by job J_{t+1} all have *high* priority, while the requests issued by $J_1, \ldots, J_t$ all have *medium* priority (which is strictly lower than *high* priority). The requests issued by J_{t+2} all have *low* priority (which is strictly lower than *medium* priority).

As with the BD_F reduction, reducing an MCM instance to the BD_P problem requires only polynomial time with respect the input size as the number of constructed jobs is linear in $t \leq |E|$ and the number of constructed requests is linear in $|V|$.

C. Properties of the constructed job set

The choice of critical section length of the requests issued by J_{t+2} allows us to infer from J_{t+1}'s blocking bound whether J_{t+1} is blocked by any short requests in a worst-case schedule.

Lemma 7. *Consider a schedule S in which J_{t+1} is blocked for $B_{t+1}(S) = B_{t+1}$ time units. If B_{t+1} is an integer multiple of Δ_L, then J_{t+1} is not blocked by any short request in S.*



(a) BD_P problem constructed from G_1 and $k = t = 2$.

(b) BD_P problem constructed from G_2 and $k = t = 2$.

Fig. 4. BD_P problems constructed from G_1 and G_2.

We provide a proof of this and the following lemmas in Appendix B, as they are structurally analogous to those discussed in the previous section. In the next lemma, we state a bound on the blocking duration that J_{t+1} can incur due to any single request for ℓ_D issued by one of the jobs $J_1, \ldots, J_t$.

Lemma 8. *Each request for ℓ_D^j issued by a job J_j, where $1 \leq j \leq t$, blocks J_{t+1} for at most $4 \cdot \Delta_L$ time units.*

We provide a proof in Appendix B. Lem. 8 leads to a straightforward upper bound on the total blocking incurred by J_{t+1} in any schedule.

Lemma 9. $B_{t+1} \leq 4 \cdot k \cdot \Delta_L$.

We provide a proof in Appendix B. Lem. 9 is illustrated in Fig. 5(a) for the BD_P instance constructed from G_1. In this schedule, J_3 is blocked for $4 \cdot k \cdot \Delta_L = 8 \cdot \Delta_L$ time units in total, and J_3 cannot be further blocked by any other request. Just as it is the case with the BD_F reduction (recall Fig. 3(a)), none of the resources $\ell_1, \ldots, \ell_4$ is requested more than once within the requests for ℓ_D^1 and ℓ_D^2 issued by J_1 and J_2 that block J_3. As stated next, this is generally the case if J_3 is blocked for $4 \cdot k \cdot \Delta_L$ time units.

Lemma 10. *Let S denote a schedule of the constructed job set. If $B_{t+1}(S) = 4 \cdot k \cdot \Delta_L$, then each resource ℓ_v with $v \in V$ is requested within at most one request for any resource ℓ_D^j with $1 \leq j \leq t$ that blocks J_{t+1}.*

The proof, similar to the proof of Lem. 5 for BD_F, is given in Appendix B. If $\mathrm{BD}_P(J_{t+1}, 4 \cdot k \cdot \Delta_L) = $ *yes*, then Lem. 10 allows inferring the existence of a matching such that no two matched edges share a vertex, and that exactly one edge from each edge partition is contained in the implied matching.

Lemma 11. *Let S denote a schedule of the constructed job set. If $B_{t+1}(S) = 4 \cdot k \cdot \Delta_L$, then each J_j with $1 \leq j \leq t$ blocks J_{t+1} with exactly one request for ℓ_D^j.*

We provide a proof in Appendix B. With the stated lemmas, it can be shown that solving the provided MCM problem instance is equivalent to solving the constructed BD_P instance.

Theorem 2. *A matching F solving the* MCM *problem exists if and only if* $\mathrm{BD}_P(J_{t+1}, 4 \cdot k \cdot \Delta_L) = $ yes.

We provide a proof in Appendix B.

Since instances of the MCM problem with $k = t$ can be solved by solving the constructed BD_P instance, and since the MCM problem is NP-complete, BD_P is NP-hard.

V. Unordered Locks: A Tractable Special Case

In contrast to priority-ordered and FIFO-ordered locks, unordered locks do not ensure any specific ordering of requests. As a consequence, each request can be blocked by any remote request for the same resource, unless both requests are issued within outer critical sections accessing the same resource. Interestingly, this rules out reductions similar to those given in Secs. III and IV. To demonstrate this, we establish in this section that, in a special case that matches the setup used to establish the hardness results in the preceding two sections, the blocking analysis optimization problem for unordered spin locks can be solved in polynomial time.

Our reductions in Secs. III and IV are oblivious to the scheduling policy employed since at most one job is assigned to each processor. In this section, we consider a similar setting for the analysis of unordered nested locks to rule out any effects related to the scheduling policy. Specifically, we assume that

- job release times are unknown (just as before),
- each job is assigned to its own dedicated processor,
- jobs can issue their requests at any point in their execution and in any order, and
- no minimum nor maximum separation between the releases of any two jobs or any two critical sections can be assumed.

Note that the reductions given in Secs. III and IV match these assumptions, that is, this restricted special case suffices to show NP-hardness of the blocking analysis problem for FIFO- and priority-ordered locks in the presence of nested critical sections.

In the following, we show that, with unordered locks, this special case can be solved in polynomial time, which establishes that reductions similar to those given in Secs. III and IV are inapplicable to this class of locks.[1] Without loss of generality, we focus on computing the blocking bound for job J_1.

Our approach relies on constructing a "blocking graph" in which requests are encoded as vertices, and the nesting relationship as well as the potential blocking between two requests are encoded as edges. In the following, we show how to construct the blocking graph such that the blocking optimization problem reduces to a simple reachability check.

A. An example blocking graph

We first consider the illustrative example provided in Fig. 6(a). Job J_1 issues two requests for the resource ℓ_2, one of which is nested within a request for ℓ_1. The jobs J_2 and J_3 issue nested and non-nested requests for ℓ_1, ℓ_2, ℓ_3, and ℓ_4 as shown in Fig. 6(a). The solid edges in Fig. 6(a) are *nesting edges* that encode the nesting relationship of requests.

To connect in the blocking graph all requests that can block each other, we iteratively consider each resource one by one.

[1]To be clear, it does not establish a tractability result for the unrestricted general case, as the general case requires addressing further issues unrelated to locking *per se* (*e.g.*, precisely characterizing the possible interleavings of multiple jobs on each processor) that we chose to exclude here.

First, we consider all requests for resource ℓ_1.

Requests for ℓ_1: In the example shown in Fig. 6(a), J_1's request for ℓ_1 can be blocked by all of J_2's and J_3's requests for ℓ_1. This is indicated by the dashed edges in Fig. 6(a) that point from J_1's request for ℓ_1 to J_2 and J_3's requests for ℓ_1. The resulting blocking graph now incorporates all blocking effects caused by requests for resource ℓ_1.

Requests for ℓ_2: In the next step, we extend the blocking graph by including edges to encode blocking due to requests for ℓ_2. J_1's non-nested request for ℓ_2 can be blocked by all other requests for ℓ_2 issued by J_2 and J_3, that is, J_2's nested request for ℓ_2 and J_3's nested and non-nested request for ℓ_2. J_1's nested request for ℓ_2 can be blocked by J_3's non-nested requests for ℓ_2, but cannot be blocked by the nested requests for ℓ_2 issued by J_2 or J_3. The reason is that J_1's nested request for ℓ_2 is nested within a request for ℓ_1, and hence it cannot be blocked by any other request for ℓ_2 also nested within a request for ℓ_1. Note that J_3's non-nested request for ℓ_2 can block J_2's nested request for ℓ_2, and hence transitively block J_1. Fig. 6(b) shows the blocking graph that encodes all blocking due to requests for ℓ_1 and ℓ_2.

Requests for ℓ_3: Although J_1 does not access ℓ_3, jobs J_2 and J_3 do, and their requests can cause transitive blocking for J_1. In particular, the nested request for ℓ_3 issued by J_2 is nested within a request for ℓ_1 that can block J_1. This nested request for ℓ_3 can be blocked by J_3's request for ℓ_3, which can then transitively block J_1. In Fig. 6(c), this is illustrated with an additional dashed arrow from J_2's nested request for ℓ_3 to J_3's request for ℓ_3.

Note that J_2's non-nested request for ℓ_3 cannot block J_1: it is not issued within a request that already blocks J_1, nor can it transitively delay a request that blocks J_1. In particular, if J_3's request for ℓ_3 blocks J_1, then it does so transitively by blocking J_2's request for ℓ_3 that is nested within a request for ℓ_1 (which in turn blocks J_1). In this case, however, J_2's non-nested request for ℓ_3 is either already completed or not issued yet, as otherwise two of J_2's outermost requests would be pending at the same time, which is not possible.

Requests for ℓ_4: The requests for ℓ_4 cannot block J_1 as they are not nested within any request that can block J_1, nor does J_1 issue any requests for ℓ_4. Hence, although the requests for ℓ_4 issued by J_2 and J_3 can block each other, they cannot block J_1.

We denote the resulting graph as *blocking graph*, since by construction it has the property that a vertex is reachable from J_1 if and only if the corresponding request can block J_1. We formalize this property in Lemmas 12 and 13.

B. Blocking graph construction

In the following, we let $e = (v_1, v_2)$ denote a directed edge from vertex v_1 to vertex v_2. Recall that we require the existence of a partial order $<$ such that if a request $R_{x,q',s'}$ issued by J_x is nested within a request $R_{x,q,s}$, then $q < q'$. Let $\ell_1, \ldots, \ell_{n_r}$ denote a sequence of shared resources that satisfies the partial order on requests. That is, a request for ℓ_i cannot be nested in any requests for ℓ_j with $j > i$.

The blocking graph is a directed, acyclic graph $G = (V, E)$ that is constructed as follows. The set of vertices V consists of one vertex for each request $R_{x,q,r}$ issued by any job J_x in the system: $V = \{v_{x,q,r} | \exists R_{x,q,r}\}$.

As shown in Fig. 6(c), we construct G with two kinds of edges: *nesting edges* E^n (shown as solid arrows) and *interference edges* E^i (shown as dashed arrows). With nesting edges we model the nesting relation among requests in G, and with interference edges we model direct or transitive blocking of J_1's requests. The set of nesting edges is defined as follows:

$$E^n = \{(v_{x,p,w}, v_{x,q,r}) | R_{x,p,w} \triangleright R_{x,q,r}\}.$$

The set of interference edges is defined inductively by considering requests for only one resource in each step, as we did in the example in Sec. V-A. We first define the subset E^i_1 of E^i that contains only edges to requests for ℓ_1. (E^i_1 corresponds to the dashed edges in Fig. 6(a).) Formally, an edge $(R_{x,q,v}, R_{y,q,w})$ is in E^i_1 if and only if $R_{x,q,v}$ is a request for ℓ_1 issued by J_1 and $R_{y,q,w}$ is a remote request (because J_1 cannot block itself) for ℓ_1:

$$(v_{1,1,v}, v_{y,1,w}) \in E^i_1 \iff \exists R_{1,1,v} \wedge \exists R_{y,1,w} \wedge y \neq 1.$$

Based on E^i_1, we define $G_1 = (V, E_1)$ to be the blocking graph with the edges $E_1 = E^i_1 \cup E^n$, similar to Fig. 6(a). Recall that n_r denotes the number of shared resources. We define E^i_t with $1 \leq t \leq n_r$ to be the set of all interference edges among requests for the resources ℓ_1 up to ℓ_t. We define the respective blocking graph G_t with $1 \leq t \leq n_r$ accordingly: $G_t = (V, E^n \cup E^i_t)$. Intuitively, the (partial) blocking graph G_t considers all requests for the resources $\ell_1, \ldots, \ell_t$ and the resulting potential blocking.

Before we show how the set E^i_{t+1} can be constructed from E^i_t, we introduce the following notation and definitions.

- The predicate $\mathit{reachable}(G, J_x, v_{x,q,r})$ holds if and only if a path in G from a request issued by J_x to the request $R_{x,q,r}$ exists. All requests of J_x are defined to be reachable.
- The set of resources that job J_x must hold when it issues the request $R_{x,q,v}$ is given by $\mathit{held}(R_{x,q,v})$. For instance, in the example illustrated in Fig. 6, job J_2 must already hold a lock on the resource ℓ_1 when it requests ℓ_2,
- Given a partial blocking graph G' and a set of requests W, $G' \setminus W$ denotes the graph that results from removing (from G') all vertices corresponding to requests in W or (transitively) nested within requests in W.
- The *conflict set* of a request $R_{y,t,s}$ is given by $\mathit{conf}(R_{y,t,s}) = \{R_{z,u,v} \mid \ell_u \in \mathit{held}(R_{y,t,s}) \vee z = y\}$, that is, the conflict set contains requests that either are also issued by the same job or that pertain that to a resource that J_y must already hold to issue $R_{y,t,s}$.

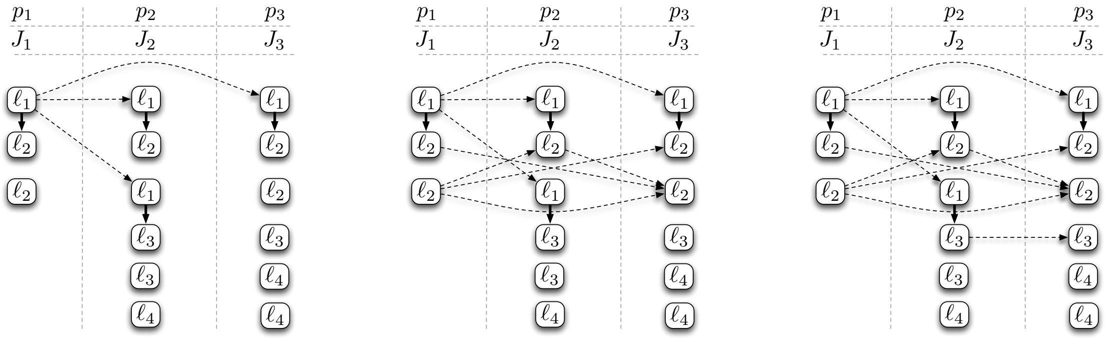
Based on the notion of the conflict set, we define the set of *non-conflicting* edges $E^{i,NC}_t$ for a resource ℓ_t with $2 \leq t \leq n_r$:

$$E^{i,NC}_t = \{(v_{x,t,r}, v_{y,t,s}) \mid x \neq y \wedge \mathit{reachable}(G_{t-1} \setminus \mathit{conf}(R_{y,t,s}), J_1, v_{x,t,r})\}.$$

In other words, $E^{i,NC}_t$ is the set of all edges $(v_{x,t,r}, v_{y,t,s})$ such that $R_{x,t,r}$ and $R_{y,t,s}$ are issued by different jobs and $v_{x,t,r}$ is reachable without visiting any vertices corresponding to requests conflicting with $R_{y,t,s}$.

With the definition of $E^{i,NC}_t$ in place, the inductive construction of the set of interference edges E^i_t for $2 \leq t \leq n_r$ is straightforward: $E^i_t = E^i_{t-1} \cup E^{i,NC}_t$. First, E^i_t contains all edges also in E^i_{t-1}, as considering the resource ℓ_t can only add interference edges. Second, E^i_t contains all non-conflicting edges $E^{i,NC}_t$ that make non-conflicting requests for ℓ_t reachable.

In particular, all of J_1's requests are reachable by definition,



(a) Considered resources: ℓ_1. (b) Considered resources: ℓ_1 and ℓ_2. (c) Considered resources: $\ell_1, \ldots, \ell_4$.

Fig. 6. Construction of the blocking graph for jobs $J_1, \ldots, J_3$. Dashed arrows indicate how J_1 can be directly or transitively blocked by remote requests.

and hence $E_t^{i,NC}$ also contains all edges connecting J_1's requests for ℓ_t with requests for ℓ_t issued by other jobs.

Since G_t reflects possible blocking due to all requests for $\ell_1, \ldots, \ell_t$, $G_{n_r} = G$ is actually the full blocking graph. By construction, G yields a blocking bound for J_1, as argued next.

C. Blocking analysis

To start with, we argue that all requests reachable in G can contribute to the delay experienced by J_1.

Lemma 12. *Under the assumed job model, there exists a schedule in which J_1 waits (i.e., is blocked) while any request $R_{x,q,r}$ (with $x \neq 1$) that is reachable in G is executed.*

Proof. We construct a schedule that is possible under the assumed job model in which J_1 waits while each such request is executed.

Consider the graph G' that extends G with an additional vertex v_S that connects to all of J_1's outermost requests (and to no other requests). Using v_S as the root, we construct a spanning tree T in G' (or, rather, the connected component that includes v_S), with the following property: each path in T from a request issued by J_1 to a reachable request $R_{x,q,r}$ contains at most one request, or a consecutive subsequence of nested requests, from each other job. (Such a path exists for each reachable $R_{x,q,r}$ since multiple non-nested requests from the same job are in conflict, *i.e.*, not included in $E_t^{i,NC}$.)

Let $p = v_S, v_1, \ldots, v_k$ denote the sequence of vertices in T visited by a pre-order traversal of T. Consider a schedule in which the requests are issued in the order $v_1, \ldots, v_k$. The request corresponding to v_1 is issued at time 0, and the other requests are issued as follows: if an interference edge between a request v_i and a request v_{i+1} exists, then v_{i+1} is issued at the same time as v_i; if a nesting edge between a request v_i and a request v_{i+1} exists, then v_{i+1} is issued as soon as all previously issued requests nested within v_i completed (or immediately after issuing v_i if no other requests nested in v_i were issued previously). In the resulting schedule, assuming that requests that are issued at the same time are serialized such that J_1's waiting time is maximized, J_1's requests wait while all other requests are being executed. Finally, such a schedule is legal under the assumed job model (and hence must be accounted for by an answer to the blocking analysis optimization problem) since neither a minimum nor a maximum separation between any two requests can be assumed. ∎

Conversely, each request $R_{x,q,r}$ that can block J_1 is reachable in G, as we show next.

Lemma 13. *If there exists a schedule S in which J_1 cannot proceed until a request $R_{x,q,r}$ (with $x \neq 1$) is complete (i.e., if $R_{x,q,r}$ blocks J_1), then $v_{x,q,r}$ is reachable in G.*

Proof. By contradiction. Suppose $R_{x,q,r}$ is the first request to block J_1 that is not reachable in G. There are three cases.

Case 1: $R_{x,q,r}$ *directly* blocks J_1 (*i.e.*, J_1 requested ℓ_q concurrently with $R_{x,q,r}$). Then there exists a request $R_{1,q,s}$ issued by J_1, and hence the edge $(v_{1,q,s}, v_{x,q,r})$ is included in G by the definition of $E_q^{i,NC}$.

Case 2: $R_{x,q,r}$ *transitively* blocks J_1 (*i.e.*, there exists a job J_y with $y \neq x$ that requested ℓ_q concurrently with $R_{x,q,r}$ and J_y blocks J_1 either directly, transitively, or due to nesting). Then there exist two requests $R_{y,q,s}$ and $R_{y,u,v}$ issued by J_y, where $R_{y,u,v}$ blocks J_1 and $R_{y,u,v} \triangleright R_{y,q,s}$. Since, by initial assumption, $R_{x,q,r}$ is the first request that both blocks J_1 and is not reachable in G, $v_{y,u,v}$ is reachable in G. Further, since nesting is well-ordered according to $>$, $v_{y,u,v}$ is also reachable in G_{q-1}. The edge $(v_{y,q,s}, v_{x,q,r})$ is hence included in G by the definition of $E_q^{i,NC}$. (The fact that $R_{y,q,s}$ and $R_{x,q,r}$ are issued concurrently implies that $\ell_u \notin held(R_{x,q,r})$.)

Case 3: $R_{x,q,r}$ blocks J_1 due to being *nested* in a blocking request (*i.e.*, there exists a request $R_{x,u,v}$ that blocks J_1 either directly, transitively, or due to nesting, and $R_{x,u,v} \triangleright R_{x,q,r}$). Then, by the definition of E^n, there exists an edge $(v_{x,u,v}, v_{x,q,r})$ in G. Further, since, $R_{x,q,r}$ is the first request that both blocks J_1 and is not reachable in G, $v_{x,u,v}$ is reachable in G.

In each case, there exists an edge from a reachable vertex to $v_{x,q,r}$, which is thus reachable, too. Contradiction. ∎

From Lemmas 12 and 13, we immediately obtain that, under the assumed job model, the solution to the blocking analysis optimization problem for J_1, namely B_1, is given by the sum of the lengths of all outermost reachable requests in G (*i.e.*, reachable requests not nested within other reachable request). Further, B_1 can be computed in polynomial time.

Theorem 3. *The construction of the blocking graph and the computation of the blocking bound B_1 can be carried out in polynomial time with respect to the size of the input.*

Proof. Clearly, V and E^n can be constructed in polynomial time with respect to the number of requests. The computation of the interference edges E^i is performed iteratively for each resource, hence $|Q| = n_r$ partial blocking graphs are computed. In each iteration, each possible edge in the graph (*i.e.*, at most $O(|V|^2)$ edges) has to be considered, and for each of them, the reachability of a set of vertices has to be checked, which takes at most $O(|V|^3)$ steps. Hence, the blocking graph can be constructed in polynomial time with respect to the input size.

Given the blocking graph, computing the set of reachable requests takes at most $O(|V|^3)$ steps, and determining whether a request is outermost with respect to the set of reachable requests requires only polynomial time as well. Hence, under the assumed job model, the blocking analysis optimization problem for unordered spin locks can be solved in polynomial time, even in the presence of nested critical sections. ■

As a final remark, note that the job model restrictions stated at the beginning of Sec. V (in particular, the absence of minimum and maximum separation constraints and the assumption of dedicated processors) are required for Lem. 12 (which establishes tightness), but not for Lem. 13 (which establishes soundness). The analysis remains thus sufficient (but not necessary) if said job model restrictions are lifted (*e.g.*, by considering the sporadic task model with minimum job inter-arrival times).

VI. Implications, Context, and Related Work

We did not set out to find intractability results. Rather, our motivation at the start of this project, intended as a continuation of our prior work on the analysis of spin locks in AUTOSAR [28], was to derive "reasonably fast" and "reasonably accurate" blocking analysis for nested spin locks. However, such analysis proved hard to find, which led us to the present results.

Nonetheless, with the continued adoption of embedded multicore platforms, solving the original problem remains important. To this end, we comment on the context, implications, and related and future work in this section.

A. Synchronization, Scheduling, and Complexity

The intersection of the fields of real-time systems and computational complexity has received substantial attention in the past decades (*e.g.*, see [26] for a survey of classic results), and intractability results for a variety of feasibility problems (*e.g.*, [5, 11, 17, 20, 24]) and commonly used schedulability analysis methods have been established (*e.g.*, [5, 9, 10]).[2] For instance, it is well known that the feasibility problem for periodic task sets is intractable [17]. Similarly, it has long been known that the feasibility problem in the presence of semaphores that ensure mutual exclusion constraints is intractable as well [20].

In this context, it is important to note that this paper pertains to a conceptually much simpler problem: rather than posing the difficult optimization problem of *finding* a feasible schedule (*i.e.*, a resource allocation policy under which no deadlines are missed), we are considering the much more restricted problem of determining the maximum blocking (*i.e.*, *not* overall schedulability) for a *given* lock type (*i.e.*, resource allocation policy). And indeed, on uniprocessors, the maximum blocking problem *is* simple for all of the commonly used real-time locking policies [2, 25], as already pointed out in Sec. II-C. The (at least to us) surprising observation made in this paper is that even this highly simplified problem is intractable on multiprocessors for FIFO- and priority-ordered locks if nesting is allowed.

However, we are certainly not the first to report intractability results related to multiprocessor real-time locking protocols. For example, Lortz and Shin [19] considered priority-ordered locks and studied the problem of assigning locking priorities to tasks such that no deadlines are missed (*i.e.*, feasible priorities), which they found to be intractable. Further, in recent work more closely related to bin packing, Hsui *et al.* [14] showed the problem of mapping critical sections to cores on many-core platforms to be intractable for several different objective functions.

Finally, in other related work on the complexity of synchronization, various hardness results have been obtained in the context of locking in database systems (*e.g.*, [21, 22, 29]). However, to the best of our knowledge, the problem of determining the maximum possible blocking on multiprocessors in the presence of nested critical sections has not been studied to date.

B. Practical Implications and Future Work

From a practical point of view, it may admittedly seem that the intractability results established in this paper do not provide a major stumbling block, as after all many commonly employed schedulability analyses—such as fixed-priority response-time analysis [10, 15] and processor-demand analysis [5, 9]—are strictly speaking intractable [9, 10]. However, these techniques exhibit their inherent hardness only in rare cases that are usually not encountered in the real world, so that their formally intractable nature is practically speaking less relevant.

In contrast, we have to date been unable to extend our prior analysis [28] in such a way that both our performance and accuracy expectations are met, which could be taken to suggest that the blocking analysis problem is not "for practical purposes still easy." Another supporting fact for this conjecture is that the runtime of the blocking analysis for the *Multiprocessor Bandwidth Inheritance protocol* [12] is super-exponential in the task set and resource model size if nesting is permitted [12].

However, obviously not all possible approaches have yet been exhausted. For instance, satisfiability (SAT) [13] and satisfiability modulo theories (SMT) [3] are examples of hard problems for which mature, highly optimized solvers have been engineered (due to their relevance in software verification) that, in many cases, can solve large instances quickly [6]. For this reason, it will be interesting to attempt leveraging SAT/SMT solvers for the blocking analysis problem, which we identify as the first open problem that we seek to highlight.

P1 Is it possible to reduce the nested critical section blocking analysis problem to SAT or SMT such that the resulting SAT or SMT instances are "easy" for existing solvers?

More formally, the challenge to derive blocking analysis for nested critical sections that strikes a balance between accuracy and speed might hint at inherent approximation hardness.

P2 Does there exist a polynomial time approximation scheme (PTAS) for the blocking analysis of nested critical sections?

As a starting point, we note that the MCM problem itself is APX-complete [16] (*i.e.*, no PTAS for the MCM exists). However, it is still unclear whether this applies to the blocking analysis of nested FIFO- and priority-ordered locks as well.

In a more practical direction, in some contexts such as safety-critical systems, it may be desirable to strike different tradeoffs than those reflected by FIFO- or priority-ordered spin locks or suspension-based locking protocols. For instance, it may be preferable to accept locking protocols that permit less concurrency if as a result accurate analysis becomes tractable. This

[2] Given task or job set, the *feasibility problem* asks whether there exists a schedule such that all deadlines are met, whereas the *schedulability problem* asks whether a specific scheduling (or resource allocation) policy will yield a schedule such that all deadlines are met (see *e.g.* [4] for a concise introduction).

raises the interesting challenge of designing locking protocols tailored to enable accurate worst-case blocking analysis.

P3 Is it possible to design a locking protocol with strong ordering guarantees that permits tractable analysis of nested critical sections?

In other words, what exactly are the properties that make a locking protocol "hard" to analyze? To this end, it may be illuminating to observe that our reductions from the MCM to the blocking decision problem for FIFO- and priority-ordered locks make use of two encoding tricks. First, by grouping two nested requests within an outer critical section that can block the job under analysis, we encode a "both or none" constraint, which is essential to our encoding of edges. Second, the FIFO- and priority order are used to encode "at most one" constraints.

Interestingly, the analysis of nested critical sections assuming unordered spin locks with a dedicated processor for each job seems to be tractable precisely because we cannot encode an "at most one" constraint. Similarly, the analysis of *non*-nested critical sections under any of the considered lock types seems to be tractable because we cannot encode "both or none" constraints without nesting. Formalizing the exact dividing line in this spectrum and investigating the impact of allowing multiple jobs per processor on the analysis complexity of unordered spin locks are interesting challenges.

In a similar vein, it will be interesting to study more closely the *real-time nested locking protocol* (RNLP) [27], which supports nested critical sections with explicit nesting rules. The RNLP, which has been proposed by Ward and Anderson [27] in the context of asymptotic blocking optimality [8], a concept that is intuitively related to the accurate analysis problem, serializes requests in FIFO order, but also delays certain nested requests under contention, and is hence not work-conserving (*i.e.*, jobs may block on available locks). This reduces concurrency and makes it difficult to encode "both or none" constraints, which makes the RNLP an interesting starting point for future studies.

P4 Does the RNLP permit accurate, yet tractable worst-case blocking analysis of nested critical sections?

Finally, an interesting question arises in the context of Ridouard *et al.*'s intractability results concerning the analysis of self-suspensions [24]. Their results state that the feasibility problem for implicit-deadline periodic tasks with self-suspensions, as they arise under partitioned scheduling when employing suspension-based real-time locking protocols such as the FIFO-ordered FMLP$^+$ [7] or the priority-ordered MPCP [23], is NP-hard. The exact feasibility analysis of suspension-based multiprocessor real-time locking protocols is thus intractable, even in the absence of nested critical sections. In future work, it would be interesting to explore if the same holds true for the analysis of spin-based locking protocols.

References

[1] "AUTOSAR Release 4.1, Specification of Operating System," http://www.autosar.org, 2013.

[2] T. Baker, "Stack-based scheduling for realtime processes," *Real-Time Systems*, vol. 3, no. 1, pp. 67–99, 1991.

[3] C. Barrett, L. Moura, and A. Stump, "Smt-comp: Satisfiability modulo theories competition," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science. Springer, 2005.

[4] S. Baruah and J. Goossens, "Scheduling real-time tasks: Algorithms and complexity," *Handbook of scheduling: Algorithms, models, and performance analysis*, vol. 3, 2004.

[5] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *RTSS '90*, 1990.

[6] N. Bjørner and L. de Moura, "$z3^{10}$: Applications, enablers, challenges and directions," in *CFV '09*, 2009.

[7] B. Brandenburg, "The FMLP$^+$: An asymptotically optimal real-time locking protocol for suspension-aware analysis," in *ECRTS'14*, 2014.

[8] B. Brandenburg and J. Anderson, "Optimality results for multiprocessor real-time locking," in *RTSS'10*, 2010, pp. 49–60.

[9] F. Eisenbrand and T. Rothvoß, "EDF-schedulability of synchronous periodic task systems is coNP-hard," in *SODA '10*, 2010.

[10] ——, "Static-priority real-time scheduling: Response time computation is NP-hard," in *RTSS '08*, 2008.

[11] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese, "Scheduling periodic tasks in a hard real-time environment," in *ICALP '10*, 2010.

[12] D. Faggioli, G. Lipari, and T. Cucinotta, "Analysis and implementation of the multiprocessor bandwidth inheritance protocol," *Real-Time Syst.*, vol. 48, no. 6, 2012.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[14] P.-C. Hsiu, D.-N. Lee, and T.-W. Kuo, "Task synchronization and allocation for many-core real-time systems," ser. EMSOFT '11, 2011.

[15] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[16] V. B. Le and F. Pfender, "Complexity results for rainbow matchings," *CoRR*, vol. abs/1312.7253, 2013.

[17] J. Y.-T. Leung and M. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information Processing Letters*, vol. 11, no. 3, 1980.

[18] J. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[19] V. B. Lortz and K. G. Shin, "Semaphore queue priority assignment for real-time multiprocessor synchronization," *IEEE Trans. Softw. Eng.*, vol. 21, no. 10, 1995.

[20] A. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[21] C. H. Papadimitriou, "The serializability of concurrent database updates," *Journal of the ACM*, vol. 26, no. 4, pp. 631–653, 1979.

[22] ——, *The Theory of Database Concurrency Control*. Computer Science Press, Inc., 1986.

[23] R. Rajkumar, "Real-time synchronization protocols for shared memory multiprocessors," *Proc. 10th Intl. Conf. on Distributed Computing Systems*, pp. 116–123, 1990.

[24] F. Ridouard, P. Richard, F. Cottet, and K. Traoré, "Some results on scheduling tasks with self-suspensions," *Journal of Embedded Computing*, 2006.

[25] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority inheritance protocols: an approach to real-time synchronization," *IEEE Trans. Comput.*, vol. 39, no. 9, pp. 1175–1185, 1990.

[26] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, "Implications of classical scheduling results for real-time systems," *Computer*, vol. 28, no. 6, pp. 16–25, 1995.

[27] B. Ward and J. Anderson, "Supporting nested locking in multiprocessor real-time systems," in *ECRTS '12*, 2012.

[28] A. Wieder and B. Brandenburg, "On spin locks in AUTOSAR: Blocking analysis of FIFO, unordered, and priority-ordered spin locks," in *RTSS '13*, 2013.

[29] M. Yannakakis, "A theory of safe locking policies in database systems," *Journal of the ACM*, vol. 29, no. 3, pp. 718–740, 1982.

Appendix

A. Generality of the $k = t$ MCM Problem

We establish that instances of the MCM problem with $k < t$ can be reduced to instances with $k = t$, which allows us to focus on the latter case in Secs. III and IV without loss of generality.

Let $G = (V, E)$ be an undirected graph with t disjoint edge partitions $E = E_1 \cup \cdots \cup E_t$, and let k be a positive integer. In the general MCM problem, we have $k \le t$ (the problem is trivial if $k > t$). If $k = t$, the two problems are identical. If $k < t$, we construct a complete bipartite graph $G_D = (V_D, E_D)$ as follows. Let $g = t - k$. We introduce $g + t$ new vertices $V_D = \{v_1^p, \ldots, v_g^p, v_1^h, \ldots, v_t^h\}$ and $g \cdot t$ new edges $E_D = \{\{v_i^p, v_j^h\} | 1 \le i \le g \wedge 1 \le j \le t\}$. Note that, since V and V_D are disjoint, the constructed graph G_D is unconnected to G. Further, by definition of E_D, G_D is bipartite as no edge

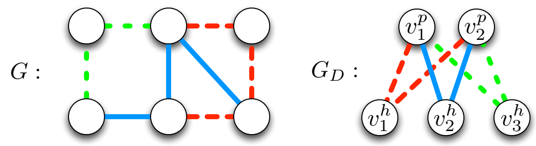


Fig. 7. Construction of the graph G_D for an instance of the MCM problem for graph G with $k = 1$ and $t = 3$ edge partitions (indicated by edge pattern).

between any two vertices $\{v_i^p, v_{i'}^p\} \subseteq \{v_1^p, \ldots, v_g^p\}$ exists and no edge between any two vertices $\{v_i^h, v_{i'}^h\} \subseteq \{v_1^h, \ldots, v_t^h\}$ exists. We let $G' = (V', E')$ denote the graph that results from merging the sets of vertices and edges of G and G_D, respectively: $V' = V \cup V_D$ and $E' = E \cup E_D$. Further, we define edge partitions $E'_1, \ldots, E'_t$ as follows:

$$\forall j, 1 \leq j \leq t: \; E'_j = E_j \cup \{\{v_i^p, v_j^h\} | 1 \leq i \leq g\}.$$

The construction of the graph $G_D = (V_D, E_D)$ is illustrated with an example in Fig. 7. Note that G_D by construction always permits a matching of size g. Due to this property, a solution to the original MCM instance in G with $k < t$ is implied by a solution to the MCM problem in G' assuming $k = t$, as we show in the following lemma.

Lemma 14. *A solution to the* MCM *problem for G' with $k' = t$ exists if and only if a solution to the original* MCM *problem for G with k exists.*

Proof. Let F' be a matching solving the MCM problem for G' with $k' = t$. By construction of the edge partitions, a matching F_D with $|F_D| = g$ solving the MCM problem in G_D always exists. Further, g is the maximum size of any valid matching in G_D. Hence, if F' solves the MCM problem for G' with $k' = k + g$, F' contains at most g edges from E_D, and removing them from F' leads to a matching F in G with size $|F'| - g = k + g - g = k$, solving the original MCM problem.

Similarly, let F be a matching solving the MCM problem for G with k. Since a matching of size g on G_D always exists and a matching of size k on G exists by assumption, it follows from the construction of G' that a matching of size k' solving the MCM problem for G' with $k' = t$ exists. ∎

B. Reduction of MCM to BD$_P$

Here we provide proofs for the lemmas stated in Sec. IV.

Proof of Lem. 7. Analogous to Lem. 2. By construction, there exist only $|V|$ short requests (issued by J_{t+2}), each of length $\Delta_S = 1$. Since $\Delta_L = 2 \cdot |V| > |V| \cdot \Delta_S$, if any of the short requests block J_{t+1} in S, then $B_{t+1}(S)/\Delta_L$ is not integer. ∎

Proof of Lem. 8. Analogous to Lem. 3. By construction, each request for ℓ_D^j from such a job J_j has a length of $2 \cdot \Delta_L$ time units and contains two nested requests for two resources ℓ_{v_1} and ℓ_{v_2}, where $\{v_1, v_2\} \subseteq V$. Also by construction, while J_j holds ℓ_D^j, it can encounter contention only from J_{t+2} (since all requests issued by jobs $J_1, \ldots, J_t$ are serialized by ℓ_U). In the worst case, each of J_j's nested requests is hence blocked only by J_{t+2}'s matching long request of length Δ_L. J_j thus releases ℓ_D^j after at most $4 \cdot \Delta_L$ time units. ∎

Proof of Lem. 9. Analogous to Lem. 4. By construction, J_{t+1} issues only a single request for each resource ℓ_D^j with $1 \leq j \leq t$. Since J_{t+1}'s requests have higher priority than the requests issued by the jobs $J_1, \ldots, J_t$, each of the requests for ℓ_D^j with $1 \leq j \leq t$ issued by J_{t+1} can be blocked by at most one request for ℓ_D^j from J_j. By Lem. 8, each of these $t = k$ requests blocks J_{t+1} for at most $4 \cdot \Delta_L$ time units. Hence, $B_{t+1} \leq 4 \cdot k \cdot \Delta_L$. ∎

Proof of Lem. 10. Analogous to Lem. 5. From Lem. 9, it follows that S is a worst-case schedule for J_{t+1}, and thus if a job J_j with $1 \leq j \leq t$ blocks J_{t+1} with a request for ℓ_D^j, then each nested request therein encounters contention from J_{t+2}.

By Lem. 7, since $B_{t+1}(S)$ is an integer multiple of Δ_L, J_{t+1} is blocked only by long requests in S. Since J_{t+2} issues only a single long request for each ℓ_v (with $v \in V$), this implies that each resource ℓ_v with $v \in V$ is requested within at most one request for any ℓ_D^j with $1 \leq j \leq t$ that blocks J_{t+1}. ∎

Proof of Lem. 11. Analogous to Lem. 6. Since J_{t+1}'s requests have higher priority than the requests of jobs $J_1, \ldots, J_t$, each of J_{t+1}'s requests for a resource ℓ_D^j with $1 \leq j \leq t$ can be blocked at most once by a request for ℓ_D^j issued by J_j. By Lem. 8, each request for ℓ_D^j from J_j with $1 \leq j \leq t$ can block J_{t+1} for at most $4 \cdot \Delta_L$ time units. Hence, J_{t+1} is blocked by exactly one request from each processor $p_1, \ldots, p_t$. ∎

Proof of Thm. 2. Analogous to Theorem 1. We show the following two implications to prove equivalence:

- $\Longrightarrow$: If BD$_P(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes*, then there exists a matching F solving the MCM problem.
- $\Longleftarrow$: If there exists a matching F solving the MCM problem, then BD$_P(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes*.

$\Longrightarrow$: It follows from BD$_P(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes* that there exists a schedule S such that $B_{t+1}(S) = 4 \cdot k \cdot \Delta_L$. We construct an MCM F from the requests that block J_{t+1} in S.

For each job J_j with $1 \leq j \leq t$, let $R_{j,D^j,s}$ denote the request for ℓ_D^j issued by J_j that blocks J_{t+1} in S. Let $edge(R_{j,D^j,s})$ denote the edge $\{v_1, v_2\}$ corresponding to $R_{j,D^j,s}$, and let F contain all edges represented by requests for the resources ℓ_D^j with $1 \leq j \leq t$ that block J_{t+1}: $F \triangleq \bigcup_{1 \leq j \leq t} \{edge(R_{j,D^j,s})\}$.

By Lem. 11, each job J_j with $1 \leq j \leq t$ blocks J_{t+1} with exactly one request for ℓ_D^j; F hence contains $|F| = t$ edges in total and exactly one edge per edge partition. By Lem. 10, for each resource ℓ_v with $v \in V$ at most one request for ℓ_v is nested within a blocking request for any resource ℓ_D^j with $1 \leq j \leq t$. Hence, each vertex $v \in V$ is adjacent to at most one edge in F. F is thus a matching solving the MCM problem.

$\Longleftarrow$: Let F be a matching solving the MCM problem for a graph $G = (V, E)$, edge partitions $E_1, \ldots, E_t$ and $k = t$. Consider a schedule S in which J_{t+1} is blocked by each request for ℓ_D^j with $1 \leq j \leq t$ that corresponds to an edge in F. Since F is an MCM in G, F contains exactly one edge from each edge partition. Then, by construction, J_{t+1} is blocked from each processor $p_j, 1 \leq j \leq t$ by exactly one request for ℓ_D^j.

As F is a matching, each vertex $v \in V$ is adjacent to at most one edge in F. Since vertices in the MCM instance correspond to resources in the BD$_P$ instance, each resource ℓ_v with $v \in V$ is requested within at most one request for any of the resources $\ell_D^1, \ldots, \ell_D^t$ that blocks J_{t+1} in S. Then each request for ℓ_v with $v \in V$ nested within a blocking request for a resource ℓ_D^j with $1 \leq j \leq t$ can be blocked by the long request for ℓ_v issued by J_{t+2}, and thus each blocking request for a resource ℓ_D^j with $1 \leq j \leq t$ can block J_{t+1} for $4 \cdot \Delta_L$ time units. Since $k = t$ requests for the resources $\ell_D^1, \ldots, \ell_D^t$ in total block J_{t+1}, there exists a schedule S such that job J_{t+1} is blocked for $4 \cdot k \cdot \Delta_L$ time units, and hence BD$_P(J_{t+1}, 4 \cdot k \cdot \Delta_L) =$ *yes*. ∎