Schedulability Analysis for Certification-friendly Multicore Systems

Jung-Eun Kim [*], Richard Bradford [†], Tarek Abdelzaher[*], Lui Sha [*]
[*]Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA
[†]Rockwell Collins, Cedar Rapids, IA 52498, USA

Abstract

This paper presents a new schedulability test for safety-critical software undergoing a transition from single-core to multicore systems - a challenge faced by multiple industries today. Our migration model, consisting of a schedulability test and execution model, is distinguished by three aspects consistent with reducing transition cost. First, it assumes externally-driven scheduling parameters, such as periods and deadlines, remain fixed (and thus known), whereas exact computation times are not. Second, it adopts a globally synchronized conflict-free I/O model that leads to a decoupling between cores, simplifying the schedulability analysis. Third, it employs global priority assignment across all tasks on each core, irrespective of application, where budget constraints on each application ensure isolation. These properties enable us to obtain a utilization bound that places an allowable limit on total task execution times. Evaluation results demonstrate the advantages of our scheduling model over competing resource partitioning approaches, such as Periodic Server and TDMA.

1 Introduction

THIS paper presents a schedulability test to support migration of safety-critical software from single-core to multicore systems. The work is motivated by the advent of multicore processors over the last decade, with increasing potential for efficiency in performance, power and size. This trend has made new single-core processors relatively scarce and as a result, has created a pressing need to transition to multicore processors. Existing previously-certified software, especially for safety- critical applications such as avionics systems, underwent rigorous certification processes based on an underlying assumption of running on a single-core processor. Providers of these certified applications wish to avoid changes that would lead to costly recertification requirements when transitioning to multicore processors.

Our paper provides a significant step toward supporting multicore solutions for safety-critical applications. It does this by building on three separate analysis methods that previously had not been applied together to multicore systems. These are:

- Utilization bound analysis using task period information,
- Conflict-free I/O scheduling, and
- Global priority assignment across all tasks on a core, irrespective of application (defined by a group of tasks), while enforcing application budgets

Our schedulability analysis can be viewed as an extension to the classical Liu and Layland (L&L) schedulability bound [1]. When known values of task periods are used in the analysis, the bound becomes even better (*i.e.*, less restrictive), often significantly so. This is because the L&L analysis makes worst-case assumptions about task periods; actual periods are unlikely to resemble the worst case (for example, the ratio of two task periods will often be a whole number, as opposed to the square root of 2, as derived by L&L).

Conflict-free I/O scheduling treats I/O transactions as non-preemptive and globally synchronizes them in a conflict-free schedule. In the analysis, I/O transactions are regarded as having the highest priority, since this is the most pessimistic assumption for other tasks' schedulability. This eliminates cross-core interference due to I/O and leads to a decoupling between cores, simplifying the schedulability analysis.

In addition, the model assigns CPU utilization budgets to each application (*i.e.*, a group of tasks), yet it schedules tasks globally across applications sharing a core. Evaluation in a single-core model showed that this architecture significantly improves schedulability over TDMA and Periodic Server, while maintaining isolation properties. We build on this model [2], providing an overview in Sec. 2.1.

Our utilization bound and global priority assignment with enforced application budgets are *complementary*; the former is useful early in the development process (indeed, even before coding begins) or during migration, whereas the latter is applicable when development is complete and all tasks' Worst Case Execution Times (WCET)s can be identified accurately. During development, and before the code is instrumented completely enough to determine WCETs with interference effects, developers can still execute the code under approximately worst-case conditions and measure processor idle time; this allows a quick and easy estimation of application utilization for comparison with the utilization bound.

2 Software Migration to Multicore Systems

This paper proposes a task execution model and a corresponding schedulability analysis test, motivated by the need to transition safety-critical software certified on single-core systems to multicore systems. Toward that end, we make three important assumptions motivated by likely transition realities and design choices: (i) task periods, deadlines, and I/O durations are known since they are tied to system specifications or derived from physical constraints and data size, but our schedulability analysis assumes exact execution times are not yet known, (ii) all I/O transactions are globally scheduled in a conflict-free manner, and (iii) global priority assignment with application budgets enforced is employed on individual cores. Our model attempts to remove all timing dependencies across applications to support portability of applications. We provide a solution to the schedulability problem given the above model.

2.1 Task Execution Model

Schedulability Analysis with Task Period Data: In this paper, we assume that an allocation of application software to cores has already taken place: we focus on scheduling instead. We are given M cores. In each core, m, we consider scheduling a set, $S_{(m)}$, of periodic tasks, where each task, $\tau_{m,i} \in S_{(m)}$, is described by a known period, $T_{m,i}$, a known relative deadline, $D_{m,i}$, and a known I/O duration, $IO_{m,i}$, but the worst case computation time of the task, denoted by $C_{m,i}$, may *not* be known. Once development is complete, the various factors and details that affect WCETs, including timing interference and delay due to shared resources (*e.g.*, bus, cache, memory), are assumed to be abstracted (by techniques such as [3], [4], [5], [6], [7]) and incorporated in the final WCETs. However, the utilization bound in our analysis framework allows for WCETs that are not yet known and still obtains a bound on allowable application utilization. We assume that tasks are indexed such that a lower number implies a higher priority in a core.

Conflict-free I/O: A key requirement for achieving isolation among cores is to ensure non-interference due to I/O. Hence, in this paper, I/O transactions are scheduled such that they are conflict-free. As a result, all I/O activity occurs strictly periodically and non-preemptively, which makes the implementation and analysis easier [8]. I/O scheduling thus reduces to choosing phasing for the I/O transactions. I/O sizes tend to be relatively short, hence their strictly periodic scheduling does not seriously degrade system schedulability - we show this property in the evaluation. I/O transactions are modeled as periodic tasks with period $T_{m,i}$ and execution time $IO_{m,i}$. To ensure isolation and due to their relatively small size, I/O transactions are analyzed as having the *highest priority*, and being globally scheduled in a conflict-free manner, such that

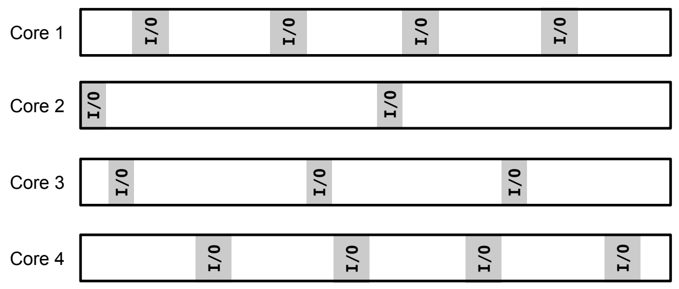


Fig. 1. Conflict-free I/O section schedule over multiple cores. I/O sections are non-preemptive and strictly periodic.

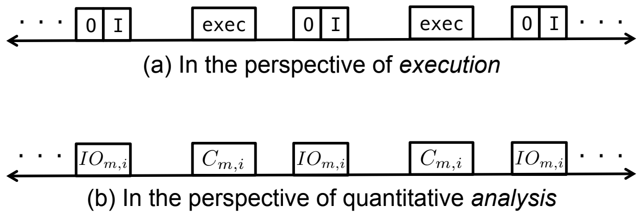

O I
exec
(a) In the perspective of *execution*
$IO_{m,i}$
$C_{m,i}$
(b) In the perspective of quantitative *analysis*

Fig. 2. Top: task execution model with I/O sections, bottom: quantitatively lumping input and output time in the analysis.

only a single section executes at a time *across all cores* as shown in Figure 1. Hence no I/O on any core will ever be blocked, preempted, or otherwise delayed by I/O from another core.

In our model, an I/O transaction must first occur to acquire input, the processing component of a task then runs, followed by I/O to deliver the output. The I/O transactions are supposed to occur strictly periodically at a pre-designated instant, even though *raw* I/Os from external sources are asynchronous. We assume that output and input occur at period boundaries back-to-back, thus combining the output and input into a contiguous interval of I/O processing. In this paper, we use the term *I/O section* to refer to such an object, having total duration, $IO_{m,i}$, for task $\tau_{m,i}$, as shown in Fig. 2. The I/O section's duration is relatively easy to bound since it depends mainly on data needs of control loops and so can be known. This duration can of course be affected by interference on shared resources such as bus and cache. We assume such interferences can be bounded by techniques such as [3], [4], [5], [6], [7] and incorporated into the I/O duration estimation.[1]

Processing tasks and I/O transactions constitute separately schedulable entities. The processing component runs at a known fixed priority value, $Pr_{m,i}$, whereas (as we explain later) I/O is regarded as *top priority*. Summing up I/O and execution time, we define the task's total utilization, $u_{m,i} = (C_{m,i} + IO_{m,i})/T_{m,i}$. The total number of tasks allocated to core m is $|S_{(m)}| = N_{(m)}$. Each task belongs to an application α_z, $(z = 1, \cdots, Z_{(m)})$, where $\alpha(\tau_{m,i})$ denotes the application to which task $\tau_{m,i}$ belongs to. Table 1 summarizes the notation used in this paper.

Global priority assignment, yet enforcing application budgets: Each application (*i.e.*, a group of tasks) is assigned to one core. Note that, in principle, an application might be allocated to span several cores. However, we do not expect this to be the common case when migrating safety-critical software certified on single-core systems to multicore platforms. This is because individual applications comprising the original single-core system must have been certified to run on a single-core. While the allocation of applications might change upon transition, we expect that in order to minimize re-certification cost, it makes sense that tasks belonging to the same application should be assigned together on the same core [2].

We further assume that Core m's utilization, U_m, is given by $U_m = \sum_{i=1}^{N_{(m)}} u_{m,i}$. At design time, each application α_z

1. Because of interference at run time, the start of an I/O section could be delayed slightly. We assume such delays (along with other context-switching delays) are captured in the assumed duration of an I/O section. Note that such interference could come only from non-I/O tasks; as explained herein, I/O sections cannot interfere with each other.

2. We would *always* expect system developers to avoid - if at all possible - breaking an application across cores. To do otherwise would invite additional complications without any additional benefit. Indeed, some processors have no shared cache between cores, so two threads of the same application running on different cores lose the advantage of caching, resulting in a significant performance loss. Meanwhile, much additional analysis would be required to manage the timing of thread execution and of resource availability on separate cores. Breaking large applications may become unavoidable for some future migrations, but it is outside the scope of this paper.

TABLE 1
Notation

Symbol	Description	
$\tau_{m,i}$	task i in core m	
$T_{m,i}$	period of $\tau_{m,i}$	
$C_{m,i}$	computation time of $\tau_{m,i}$	not given
$Pr_{m,i}$	priority of $\tau_{m,i}$	
$IO_{m,i}$	duration of I/O transaction of $\tau_{m,i}$	
$\psi_{m,i}$	offset of I/O transaction of $\tau_{m,i}$	
$u_{m,i}$	utilization of $\tau_{m,i}$	
U_m	utilization of core m	
α_z	application z	
$\alpha(\tau_{m,i})$	application to which $\tau_{m,i}$ belongs	
B_z	CPU utilization budget assigned for α_z	
$\mathcal{A}^n_{(m)}$	a set of applications to which $\tau_{m,n}$'s higher priority tasks belongs but excluding to which $\tau_{m,n}$ belongs	
M	core count	
$N_{(m)}$	task count in core m	

is assigned a budget B_z, defined as the maximum CPU utilization allowable for the sum of its tasks. Hence, for each α_z, when development is complete, the code should satisfy

$$\sum_{\forall \tau_{m,i} s.t. \alpha(\tau_{m,i})=\alpha_z} u_{m,i} \leq B_z. \tag{1}$$

Observe that the budget, B_z, of application α_z is *a design-time constraint*, not a run-time resource partitioning mechanism. Compliance with application budgets is checked repeatedly throughout the software development process. In cases of noncompliance, either software must be refactored, or else new schedules must be computed. For fielded software, WCET bounds for individual tasks will be enforced, thereby indirectly enforcing application budget compliance. Such WCET-enforced tasks will be scheduled using regular fixed priority scheduling. Hence, this mechanism indirectly allows enforcement of resource budgets, without employing resource partitioning mechanisms at run-time. [2] The mechanism avoids inefficiencies of resource-partitioned systems, such as the Periodic Server and TDMA, arising due to priority inversion when a high-priority task in one partition must wait because the CPU is presently allocated to another partition (where a lower-priority task might be executing). See Figure 3. [3] Tasks' schedulability is analyzed in a fixed-priority fashion no matter which application they belong to.

2.2 An Equivalent Independent Task Model

Before developing our schedulability test, we note that the task model above can be transformed to one of scheduling independent tasks on each core. By assumption, we require I/O to be non-preemptible, and we require the following precedence constraints involving execution and I/O tasks to be satisfied for every invocation of every task: (i) the processing component does not begin until after the sub-task of acquiring input is complete, (ii) the sub-task of delivering the output does not begin until after the processing component is completed, and (iii) the sub-task of acquiring input for the next period's invocation of the task does not begin until after the sub-task of delivering the output from the current period's invocation is completed. (Note that the very first invocation of the task in the global schedule does not require a predecessor.) The following theorem shows that using the concept of I/O sections allows these precedence constraints to be satisfied automatically.

Theorem 1. *If a feasible schedule exists with I/O sections scheduled strictly periodically and conflict-free, then there exists a feasible schedule in which the precedence constraints in our task execution model are satisfied.*

3. In this experiment, I/O sections are considered. The detailed information can be found in Sec. 6.

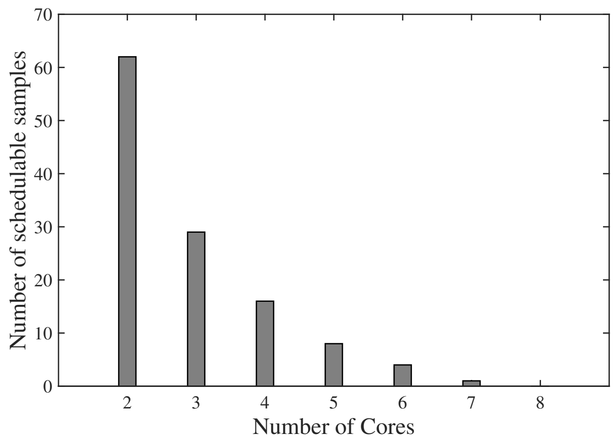


Fig. 3. Low scalability of TDMA: the number of instances schedulable by TDMA according to core count.

Proof. Consider an arbitrary task $\tau_{m,i}$. In a feasible schedule with I/O sections scheduled periodically and conflict-free, each invocation of $\tau_{m,i}$ gets a total I/O processing time of $I_{m,i}+O_{m,i}$ within each period. In addition, $\tau_{m,i}$ gets an allocation of at least $C_{m,i}$ units of processor time in each period. If I/O sections consist of an output sub-task followed by an input sub-task, and if each invocation of the processing task follows after the I/O section, then the precedence constraint (i) above is satisfied. Since the processing task gets at least $C_{m,i}$ units of processor time in each period, each invocation of the processing component can complete before the next I/O section begins; hence precedence constraint (ii) is satisfied. Finally precedence constraint (iii) is satisfied by the construction of I/O sections. $\square$

Accordingly, we are eliminating cross-core interference due to I/O and obtaining the utilization bound to place an allowable limit on total task execution times including other interference effects. As a result, our schedulability problem is distilled into two subproblems:

- Ensure that I/O sections are scheduled strictly periodically and conflict-free.
- Analyze task schedulability on each core separately.

3 Schedulability Analysis with Budget Constraints

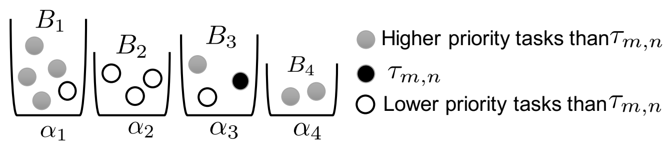
Per the discussion above, in this section, we analyze the schedulability of tasks on each core. We do so by analyzing schedulability of one task at a time, considering its application budget constraint.

3.1 Overview of Approach

A valid utilization bound for an *individual task*, say $\tau_{m,n}$ on core m, denoted by $U^{n}_{m,bound}$ means that it is schedulable whenever the overall utilization of the task set on core m satisfies $U_m \leq U^{n}_{m,bound}$. Since periods, relative deadlines, priorities, and I/O sections are known, the bound is computed by minimizing the utilization of a *critically schedulable*[4] task set on core m, $\sum_{i=1}^{n} u_{m,i}$, over all possible values of computation times $C_{m,i}$ for $1 \leq i \leq n$.

Consider the critical time zone[5] of task $\tau_{m,n}$, of application α_z, where the task arrives at time $t=0$ together with its all higher priority tasks. Suppose that the invocations in this time interval are a critically schedulable task set. Since scheduling is work-conserving, it follows that the time interval $0 \leq t < D_{m,n}$ is continuously busy. At the same time, budget constraints limit the utilization of all the tasks in α_z up to B_z. However, these two constraints conflict with each

4. A task set is critically schedulable if any increase in execution time of any task would make the set unschedulable [1].
5. A critical time zone is defined as the time interval between a critical instant and the end of the response to the corresponding request of the task [1].


B_1
B_2
B_3
B_4
α_1
α_2
α_3
α_4
● Higher priority tasks than $\tau_{m,n}$
● $\tau_{m,n}$
○ Lower priority tasks than $\tau_{m,n}$

(0. Test if there exists a feasible I/O schedule using the MILP formulation in Sec. 4.)
1. Calculate $U^n_{m,released}$ with α_1 and α_4.
2. Calculate $B^n_{m,total}$ which is $B_1 + B_3 + B_4$.
3. If $B^n_{m,total} \leq U^n_{m,released}$, $\tau_{m,n}$ is schedulable.

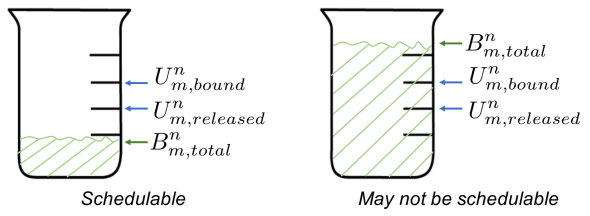


Fig. 4. The overview of schedulability analysis with budget constraints, and the relationship between $U^n_{m,released}$, $U^n_{m,bound}$ and $B^n_{m,total}$.

other since budgets could be too small to make the critical interval continuously busy with no gaps, and thus could make $U^n_{m,bound}$ not obtainable.

To tackle this issue, we release (*i.e.*, remove) the budget constraint for the application $\alpha(\tau_{m,n})$. Let the resultant bound be $U^n_{m,released}$. Note that removing a constraint in a minimization problem cannot lead to a higher-value solution, because the optimal solution to the problem *before* the removal remains feasible for the problem *after* the removal. Therefore,

$$U^n_{m,released} \leq U^n_{m,bound}. \tag{2}$$

Define set $\mathcal{A}^n_{(m)}$ as the set of applications, excluding $\alpha(\tau_{m,n})$, on core m containing higher priority tasks than $\tau_{m,n}$. Then, budget constraints for the applications are as follows,

$$\forall \alpha_z \in \mathcal{A}^n_{(m)}, \alpha(\tau_{m,i}) \neq \alpha_z, \sum_{i=1}^{n-1} u_{m,i} \leq B_z.$$

Let us denote $B^n_{m,total}$ as the budget sum of the applications to which $\tau_{m,n}$ and its higher priority tasks belong, *i.e.*,

$$B^n_{m,total} = \sum_{1 \leq z \leq Z_{(m)}} B_z, \ \forall \alpha_z \in (\mathcal{A}^n_{(m)} \cup \{\alpha(\tau_{m,n})\}).$$

Then, if $B^n_{m,total}$, is less than or equal to $U^n_{m,released}$, $\tau_{m,n}$ is determined to be schedulable by the following theorem.

Theorem 2. *If $\tau_{m,n}$ is compliant with its budget and $B^n_{m,total}$ is less than or equal to $U^n_{m,released}$, $\tau_{m,n}$ is schedulable.*

Proof. If $B^n_{m,total} \leq U^n_{m,released}$, by (2)

$$B^n_{m,total} \leq U^n_{m,released} \leq U^n_{m,bound},$$

Then $B^n_{m,total} \leq U^n_{m,bound}$. It means that $\tau_{m,n}$ is schedulable by the definition of utilization bound for schedulability. □

This test is applied to one task at a time, and a core is determined to be schedulable if all tasks on the core are schedulable. The procedure is illustrated in Fig. 4. In the next section, we show how to compute $U^n_{m,released}$.

3.2 The Utilization Bound

It remains to compute the utilization bound, $U^{n}_{m,released}$, which is the lowest possible utilization when task $\tau_{m,n}$ and its higher priority tasks are critically schedulable. It is computed over all possible values of the executions times of the higher-priority tasks on the same core. This is formulated as a linear programming (LP) problem and solved by a standard LP solver.

[Constraint 1] – Critically schedulable:

For task $\tau_{m,n}$ and its higher priority tasks to be *critically schedulable*, computation times need to *fully* utilize the available processor time within the critical time zone from the critical instant to the deadline. Hence, as all higher priority tasks and $\tau_{m,n}$ release at time 0, collectively their maximum possible amount of computation times *must add up to* $D_{m,n}$:

$$C_{m,n} + \sum_{i=1}^{n-1} \lceil \frac{D_{m,n}}{T_{m,i}} \rceil C_{m,i} + \sum_{i=1}^{n} \lceil \frac{D_{m,n}}{T_{m,i}} \rceil IO_{m,i} = D_{m,n}.$$

[Constraint 2] – Fully utilized:

Even though Constraint 1 is satisfied, there could be empty gaps prior to $D_{m,n}$, which violates the assumption of fully (*i.e.*, continuously) utilizing the processor time. To prevent such a situation we need an additional constraint which checks, at every arrival ($l \cdot T_k$) of a task, if the cumulative demand up to time $l \cdot T_k$ is greater than or equal to $l \cdot T_k$ [9], [10]:

$$\begin{aligned} &\forall 1 \le k \le n, \forall 1 \le l \le \lfloor \tfrac{D_{m,n}}{T_{m,k}} \rfloor, \\ &C_{m,n} + \sum_{i=1}^{n-1} \lceil \frac{l \cdot T_{m,k}}{T_{m,i}} \rceil C_{m,i} + \sum_{i=1}^{n} \lceil \frac{l \cdot T_{m,k}}{T_{m,i}} \rceil IO_{m,i} \ge l \cdot T_{m,k}. \end{aligned} \tag{3}$$

For testing, $\tau_{m,n}$, $\sum_{k=1}^{n} \lfloor \frac{D_{m,n}}{T_{m,k}} \rfloor$ constraints are generated. This number can be reduced by using $\mathcal{P}_i(t)$ presented in [11], which is defined as follows (see (6) in [11]):

$$\mathcal{P}_0(t) = \{t\}, \;\; \mathcal{P}_i(t) = \mathcal{P}_{i-1}(\lfloor \frac{t}{T_i} \rfloor T_i) \cup \mathcal{P}_{i-1}(t).$$

Accordingly, not all the arrivals at $l \cdot T_k$ ($\forall 1 \le k \le n, \forall 1 \le l \le \lfloor \frac{D_{m,n}}{T_{m,k}} \rfloor$) but only the subset of them, $t \in \mathcal{P}_{n-1}(D_{m,n})$, are considered as follows:

$$\begin{aligned} &t \in \mathcal{P}_{n-1}(D_{m,n}), \\ &C_{m,n} + \sum_{i=1}^{n-1} \lceil \frac{t}{T_{m,i}} \rceil C_{m,i} + \sum_{i=1}^{n} \lceil \frac{t}{T_{m,i}} \rceil IO_{m,i} \ge t. \end{aligned} \tag{4}$$

In the worst case, the number of constraints can be 2^n [11]. However, set $\mathcal{P}_i(t)$ generates redundantly identical values. Hence we remove the redundancy and thus have fewer constraints. Fig. 5 shows the average number of constraints generated by the original formulation (3) of Constraint 2 and by (4) (but after removing redundant values).

Finally, we formulate the LP problem of finding $U^{n}_{m,released}$ for a given task $\tau_{m,n}$ as follows:

[Find $\mathbf{U^{n}_{m,released}}$] Minimize $\sum_{i=1}^{n} u_{m,i}$

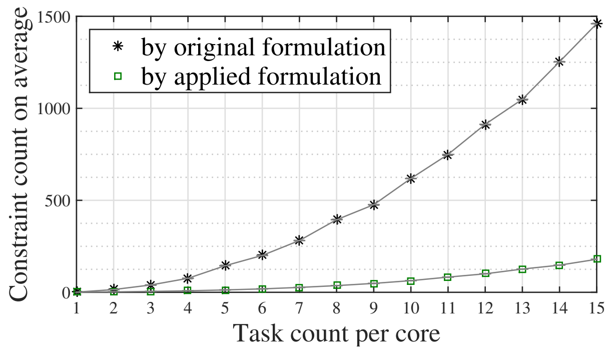


Fig. 5. Comparison of the number of constraints generated by original formulation of Constraint 2 vs. redundancy removed from (4). The result is shown for the data from Fig. 7 in the evaluation section.

Subject to:

- $\sum_{i=1}^{n-1} u_{m,i} \leq B_z, \qquad \forall \alpha_z \in \mathcal{A}^n_{(m)}, \alpha(\tau_{m,i}) \neq \alpha_z.$
- $C_{m,n} + \sum_{i=1}^{n-1} \lceil \frac{D_{m,n}}{T_{m,i}} \rceil C_{m,i} + \sum_{i=1}^{n} \lceil \frac{D_{m,n}}{T_{m,i}} \rceil IO_{m,i} = D_{m,n}.$
- $C_{m,n} + \sum_{i=1}^{n-1} \lceil \frac{t}{T_{m,i}} \rceil C_{m,i} + \sum_{i=1}^{n} \lceil \frac{t}{T_{m,i}} \rceil IO_{m,i} \geq t,$

  where $t \in \mathcal{P}_{n-1}(D_{m,n})$.

The return value of the problem above is $U^n_{m,released}$ which minimizes the total utilization of all higher priority tasks and $\tau_{m,n}$. If $B^n_{m,total} \leq U^n_{m,released}$, then $\tau_{m,n}$ is determined to be schedulable by Theorem 2.

4 Conflict-Free I/O

Since a bus is shared by multiple cores commonly on multicore processors, unpredictable interference among I/O sections could occur if conflicted. Hence we schedule I/O sections in a way that only one I/O section must be executed at a time *across all cores*. The I/O sections are non-preemptive and strictly periodic as can be seen in [12], [13], [14], [15], [16]. In [17], a necessary and sufficient condition that any two non-preemptive and strictly periodic intervals do not overlap each other was presented. We apply this condition to our problem for any two I/O sections of $\tau_{p,i}$ and $\tau_{q,j}$ on *any* core p and q as follows (core p and q may or may not be same):

$$\begin{aligned} IO_{p,i} \leq (\psi_{q,j} - \psi_{p,i}) \mod \texttt{gcd}(T_{p,i}, T_{q,j}) \\ \leq \texttt{gcd}(T_{p,i}, T_{q,j}) - IO_{q,j} \end{aligned}$$

where $\psi_{*,x}$ denotes the initial offset of $IO_{*,x}$ (appearing at every $\psi_{*,x} + kT_{*,x}$, $k = 0, 1, \ldots$) and `gcd` is the greatest common divisor function. The current form of the inequality above is not linear due to the modulo operation. Hence, we reformulate it as the following mixed integer linear programming problem:

$$IO_{p,i} \leq (\psi_{q,j} - \psi_{p,i}) - \texttt{gcd}(T_{p,i}, T_{q,j}) \cdot K \tag{5}$$

$$(\psi_{q,j} - \psi_{p,i}) - \texttt{gcd}(T_{p,i}, T_{q,j}) \cdot K \leq \texttt{gcd}(T_{p,i}, T_{q,j}) - IO_{q,j} \tag{6}$$

where K is a new real-valued variable bounded in

$$\left\lfloor \frac{1 - T_{p,i}}{\texttt{gcd}(T_{p,i}, T_{q,j})} \right\rfloor - 1 \leq K \leq \left\lfloor \frac{T_{q,j} - 1}{\texttt{gcd}(T_{p,i}, T_{q,j})} \right\rfloor + 1.$$

If a feasible conflict-free I/O schedule exists, we individually obtain $U^n_{m,released}$ to test schedulability of each task on each core, as explained in the previous section (Sec. 3).

5 An Illustrative Example

TABLE 2
Task set for an illustrative example

		budget		period	deadline	I/O
core 1	application 1	0.5	$\tau_{1,2}$	12	12	2
			$\tau_{1,3}$	16	16	1
	application 2	0.25	$\tau_{1,1}$	8	8	1
core 2	application 1	0.9	$\tau_{2,1}$	24	21	1

To illustrate how to apply the approach explained in the previous sections, let us consider an example which has 4 tasks in three applications running on two cores, as shown in Table 2.

We first check if a conflict-free I/O schedule exists. For this, we solve the mixed integer linear program in (5) and (6) for every pair among $IO_{1,1}$, $IO_{1,2}$, $IO_{1,3}$ and $IO_{2,1}$. Then, one of possible solutions is: $\psi_{1,1} = 1$, $\psi_{1,2} = 2$, $\psi_{1,3} = 5$, and $\psi_{2,1} = 16$. Hence the existence of a global I/O schedule is checked. Below we shall compute $U^3_{1,released}$ which is the theoretical bound on utilization for checking the schedulability of task $\tau_{1,3}$ on core 1.

The objective function of the optimization problem (presented at the end of Sec. 3) is

$$\sum_{i=1}^{3} \frac{C_{1,i}}{T_{1,i}} + \frac{IO_{1,i}}{T_{1,i}} = \frac{C_{1,1}}{8} + \frac{C_{1,2}}{12} + \frac{C_{1,3}}{16} + \frac{1}{8} + \frac{2}{12} + \frac{1}{16}.$$

The budget constraint is:

$$\frac{C_{1,1} + IO_{1,1}}{T_{1,1}} = \frac{C_{1,1} + 1}{8} \leq 0.25 \Rightarrow C_{1,1} \leq 1$$

Note that $\tau_{1,1}$ is the only higher-priority task than $\tau_{1,3}$ in other applications on the same core.

For $\tau_{1,3}$ to be critically schedulable:

$$C_{1,3} + \lceil\frac{16}{8}\rceil C_{1,1} + \lceil\frac{16}{12}\rceil C_{1,2} + \lceil\frac{16}{8}\rceil \cdot 1 + \lceil\frac{16}{12}\rceil \cdot 2 + \lceil\frac{16}{16}\rceil \cdot 1 = 16$$

$$C_{1,3} + 2C_{1,1} + 2C_{1,2} = 9$$

For $\tau_{1,3}$'s critical zone to be fully utilized:

$$C_{1,3} + \lceil\frac{8}{8}\rceil C_{1,1} + \lceil\frac{8}{12}\rceil C_{1,2} + \lceil\frac{8}{8}\rceil \cdot 1 + \lceil\frac{8}{12}\rceil \cdot 2 + \lceil\frac{8}{16}\rceil \cdot 1 \geq 8,$$

$$C_{1,3} + \lceil\frac{12}{8}\rceil C_{1,1} + \lceil\frac{12}{12}\rceil C_{1,2} + \lceil\frac{12}{8}\rceil \cdot 1 + \lceil\frac{12}{12}\rceil \cdot 2 + \lceil\frac{12}{16}\rceil \cdot 1 \geq 12.$$

Then,

$$C_{1,3} + C_{1,1} + C_{1,2} \geq 4,$$

and

$$C_{1,3} + 2C_{1,1} + C_{1,2} \geq 7.$$

Finally, solving the linear program above results in the computation times of $C_{1,1} = 0$; $C_{1,2} = 2$; $C_{1,3} = 5$. It should be noted that a solution to this optimization problem does not necessarily correspond to a realistic task set; instead, it gives us a limiting case that defines a sufficient condition for schedulable utilization. The corresponding utilization bound is

$$U^3_{1,released} = \frac{C_{1,1}}{8} + \frac{C_{1,2}}{12} + \frac{C_{1,3}}{16} + \frac{1}{8} + \frac{2}{12} + \frac{1}{16} \simeq 83.333\%.$$

Comparing this with the allowed total budget on core 1, we have $75\% \leq 83.333\%$. This means $\tau_{1,3}$ is schedulable as long as its execution time when development is complete (and also estimated bounds on any inter-core interference are taken into account) is compliant with its application budget. For instance, suppose $C_{1,2}$ and $C_{1,3}$ are finally bounded to 1 and 3, respectively. This is good because the utilization is under the budget. On the other hand, if $C_{1,2} = 2$ and $C_{1,3} = 2$, they are not compliant with the budget and thus the task set might not be schedulable.

The bounds $U^1_{1,released}$ and $U^2_{1,released}$ are computed similarly. Trivially, $U^1_{1,released}$ = 100%, and thus $\tau_{1,1}$ is schedulable. For $U^2_{1,released}$, we can obtain $C_{1,1} = 1$ and $C_{1,2} = 6$. The resulting bound $U^2_{1,released}$ is

$$\frac{1+1}{8} + \frac{2+6}{12} \simeq 91.667\%.$$

Since $B^2_{1,total} = 75\%$, $\tau_{1,2}$ is schedulable. As a result, since all three tasks are schedulable, core 1 is schedulable. In addition, since $\tau_{2,1}$ is schedulable as its utilization bound is 87.5% while $B^1_{2,total} = 0.9$, core 2 might not be schedulable.

In this example, the worst-case scenario ended up with $C_{1,1} = 0$. As mentioned earlier, such solutions are derived to obtain the extreme utilization 'bound' with the worst-case combination; no lower bound can be found by other combinations. In other words, if there is any increase on the resulting task execution times, the utilization bound would not be lower than the current utilization. For example, if we increase $C_{1,1}$ to 1 in the above example, and solve for the other execution times, we could get $C_{1,2} = 1$ and $C_{1,3} = 5$. The resulting utilization is then

$$\frac{C_{1,1}}{8} + \frac{C_{1,2}}{12} + \frac{C_{1,3}}{16} + \frac{1}{8} + \frac{2}{12} + \frac{1}{16} = 87.5\%$$

which is higher than what is computed before (83.333%). Hence, it is not the lowest-utilization critically-schedulable scenario. In short, we only aim to obtain the lowest 'bound' to compare for schedulability.

6 Evaluation

We first evaluate the impact of I/O on schedulability then compare our new schedulability test for individual cores to the Liu and Layland bound, showing the impact of using information on periods and deadlines. Lastly, we apply these results in comparing the count of schedulable instances using our approach, Periodic Server and TDMA. The results show that our approach can schedule more instances since it can avoid any unnecessary priority inversion. We evaluate with synthetic data since actual avionics systems have yet to be certified when *multiple cores* are running. However, we model tasks similar to actual ones - tasks are periodic, deadline-constrained, and contain I/O sections. All experiments ran on an Intel Core i7-2600 CPU 3.40 GHz with 16GB RAM, using IBM ILOG Cplex Optimizer for the linear programs, and solved essentially instantaneously.

6.1 I/O schedulability

We compare the cases of non- and harmonic periods:

- Non-harmonic periods: we generated 1,000 task sets from ten groups having, respectively, total I/O utilization (across the entire system), 0-10%, $\cdots$, 90-100%, 100 instances per group. For each instance, there are 4 cores, and up to 60 tasks. Each task period is a divisor of 54000 which is $2^4 \cdot 3^3 \cdot 5^3$ and I/O duration is randomly drawn but not

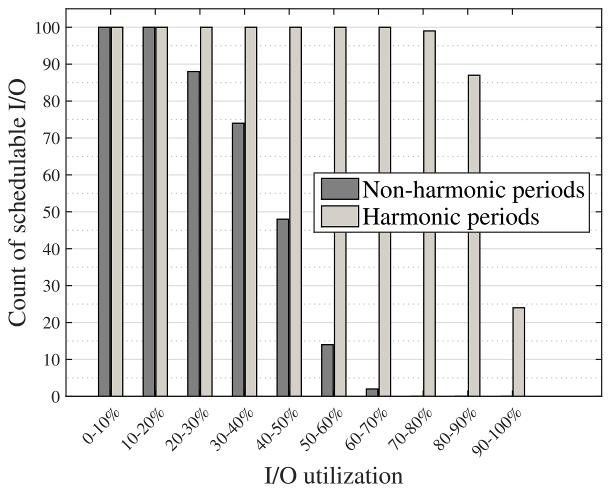


Fig. 6. Number of task set instances that have schedulable I/O when task periods are non-harmonic vs. harmonic.

longer than half of the `gcd` of the periods.

- HARMONIC PERIODS: instances are generated same as the non-harmonic period case, but task periods are harmonic (each period is either a divisor or a multiple of any other).

Figure 6 shows the number of task sets that have a feasible schedule of I/O section for different I/O utilization. As the I/O load increases, the schedulable rate of non-harmonic periods decreases, and there is no single schedulable instance when utilization is over 70%. Harmonic I/Os can achieve much higher utilization; nevertheless, because of their non-preemptivity, the I/O sections cannot always be schedulable even when the utilization is 100%. Beyond the comparison between harmonic and non-harmonic cases, we can see that relatively lower I/O utilization would not impact the entire system schedulability. Even in the non-harmonic case, I/O sections are always schedulable with I/O load of up to about 20%. As we have found that I/O utilization in practice is small (*e.g.*, less than 5%), it is reasonable to conclude our model of non-preemptive I/O sections is not overly restrictive.

6.2 Utilization Bound

As an alternative to existing resource partitioning schemes, we used a global scheduling approach that ignores application boundaries and schedules tasks according to their fixed priorities on each core. For per-core bound, we have obtained

$$\min_{1 \leq n \leq N_{(m)}} U^{n}_{m,released}$$

on each core m. In Figure 7, each point corresponds to per-core bound for each task set. For this experiment, we used the same parameters as the non-harmonic period case in Sec. 6.1 above, but kept I/O utilization in 1%-5%. We generated 1,500 task sets with evenly-distributed numbers of tasks per core, *i.e.*, 100 instances per task count. For comparison, the Liu and Layland utilization bound is plotted as well (line plot), when deadlines are equal to periods.

As seen from the result, the bounds calculated by our approach are above the L&L bound. This is because the analysis takes advantage of the information on task periods. In practice, as development proceeds and teams gain more information on task WCETs, they can recalculate the utilization. As long as the eventual task WCETs yield utilization levels that comply with the bounds, the tasks will be schedulable.

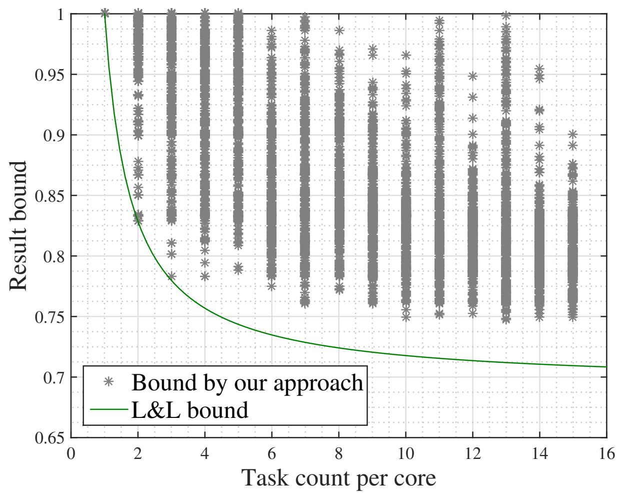


Fig. 7. Utilization bounds per core with known periods (points) are higher than the Liu and Layland bound (line) (when deadline is equal to period).

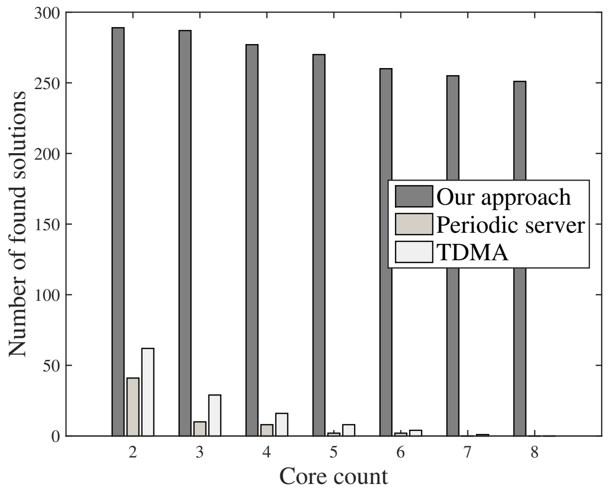


Fig. 8. The number of instances schedulable by our approach, Periodic Server, and TDMA.

6.3 Our Approach vs. Other Resource Partitioning Mechanisms

The same parameters were used as in the previous cases except that here we vary core count from 2 to 8 – we generated 300 instances for each core count. Fig. 8 shows the number of task set instances that are schedulable by our approach, Periodic Server, and TDMA. In order for an instance to be schedulable, all tasks in all applications (*i.e.*, partitions and servers) on every core must be schedulable. In Periodic Server approach, each application is assigned its own server. The period of the server is the greatest common divisor of the periods of tasks that belong to the server, which is essentially a best-case assumption for Periodic Server. The utilization of the server is equal to the application budget. This allows us to compute server parameters. The analysis in [18] is used to test the schedulability. This analysis requires execution time information. For fairness, the execution times and all other data (if applicable) used here by Periodic Server are also applied in our approach and TDMA as well. When TDMA is used, each application is assigned a TDMA partition. The utilization of the partition is equal to the application budget. Then the greatest common divisor of the periods is used as a major cycle for a TDMA schedule. The schedulability of TDMA was analyzed by the method presented in [19].

From Fig. 8, we can see that our approach schedules more task sets than the TDMA or Periodic Server can. This is because our approach avoids priority inversions by scheduling tasks irrespective of their assignments to applications. All the three approaches show also that task sets with higher core count are less schedulable. Our result should not be read as saying Periodic Server is inferior to TDMA, because in some instances Periodic Server can successfully schedule them while TDMA does not and vice versa.

7 Related Work

I/O is one of the most serious interference sources on multicore processors. However, it has not been extensively researched in real-time computing literature, whereas bus, cache and memory related issues have been extensively addressed in [3], [4], [5], [6], [7]. In this paper, we tackle the I/O issue by a *conflict-free I/O* model, which is motivated by concepts such as zero partition or device management partition [12], [13], [14], [15], [16] that are used in IMA (Integrated Modular Avionics) systems. The strictly periodic and non-preemptive model of I/O sections makes implementation and analysis easier [8]. There has been a stream of work on scheduling optimization for non-preemptive and strictly periodic tasks on multiprocessor systems. In [17], [20], Korst *et al.* addressed the problem of scheduling strictly periodic and non-preemptive tasks on a multiprocessor with the minimum number of processors. The authors showed that the problem is NP-complete in the strong sense even for a uniprocessor case. Al Sheikh *et al.* [21] addressed a similar problem by proposing a Game Theory based algorithm that not only finds a feasible schedule of tasks but also maximizes the relative distances between them. Kermia *et al.* [22] presented a greedy heuristic approach to find a non-preemptive multiprocessor schedule of strictly periodic tasks with precedence constraints. However, exclusiveness in resource sharing across processors is not modeled in these work. In [16], IMA-partition level exclusive I/O model was addressed, while in this paper each task has its own I/O section.

Perhaps the earliest work in a single-core system for utilization bound to account for unknown execution times was a generalized utilization bound for fixed-priority scheduling by Park *et al.* [23] that takes into account task periods, deadlines, and arbitrary fixed priorities. The underlying optimization problem was later simplified, leading to a solution that was not tight [24]. Chen *et al.* proposed other approximations [25], together with a tight utilization bound computed in exponential time for a periodic task model with known periods and implicit deadlines, under rate monotonic scheduling. A more recent approach to solve the problem offered another linear programming formulation [26]. In [11], Bini and Buttazzo proposed a method to reduce the number of constraints that should be considered.

Some work has addressed temporal modularity for resource partitioning. In [19], the author presented a scheduling bound when only information of higher-level partitions is given. The work was based on TDMA and assumes that periods are the same as deadlines. Shin and Lee [27] proposed the periodic resource model for hierarchical scheduling. They propose schedulability analysis of tasks mapped to a periodic resource supply (server). The authors presented the exact schedulability analysis under RM and EDF scheduling and derived the corresponding utilization bounds. Davis *et al.* [18], [28] presented the exact worst-case response time analysis under the periodic server, sporadic server, and deferrable server. Differently from their work, [2] assigns only CPU utilization for each application (a group of tasks) and then globally schedules tasks (i.e., ignores application-level isolation). We employed this model in this paper.

Others have made efforts to find good parameters for a higher-level resource (e.g., server, partition, or budget) in hierarchical scheduling. That is to calculate the minimal amount for the allocated partitioned resource so the system is schedulable. Almeida *et al*. [29] analyzed a periodic server model by introducing the server availability function. They also developed a heuristic algorithm for server (resource) parameter optimization achieving the minimum system utilization. Lipari *et al*. [30] also considered the server parameter optimization problem in a hierarchical scheduling system with a different approach of schedulability analysis. Yoon *et al*. [31] considered multiple resources for parameter optimization achieving the minimum system utilization. In that work, the authors solved the problem with Geometric Programming. In [32], Saewong *et al*. presented a response time analysis for real-time guarantees for deferrable and sporadic servers. [18], [28] also presented a greedy algorithm to select multiple server parameters and addressed optimal selection as a holistic problem.

As a hierarchical scheduling framework for safety-critical systems, specifically such as avionics, the IMA architecture has been widely adopted in industry. In [33], [34], Lee *et al.* considered an IMA system in which multiple processors are connected over an Avionics Full-Duplex Switched Ethernet. The authors presented a method to find a cyclic schedule for both IMA partitions and bus channels that guarantees the timing requirements of tasks and messages. Tămaş–Selicean *et al.* [35], [36] considered an optimization problem of scheduling mixed-criticality partitioned resources in a distributed architecture. The presented algorithm finds an assignment of tasks and partitions as well as their schedules that satisfy application schedulability while minimizing the design cost. Rufino *et al.* in [37] addressed a Temporal and Spatial Partitioning system that includes a partitioning concept that is relevant for IMA. Unlike ours, the IMA solutions are based on TDMA, which we show is inferior in terms of schedulability.

8 Conclusion

In this paper, we presented a schedulability test and task execution model to support reducing the cost of migrating software from single-core to multicore systems. Our schedulability test improves upon classical schedulability bounds by taking advantage of known information on task periods and deadlines. By synchronizing I/O sections globally so as to be conflict-free across cores, I/O-level isolation was achieved between cores. Once the existence of a global I/O schedule was assured, periodic tasks were analyzed for schedulability on each core subject to application budgets that provide temporal modularity. Together, our schedulability test and task execution model enable us to obtain a utilization bound that places an allowable limit on total task execution times. We compared our approach to other resource partitioning approaches and showed a considerable advantage in schedulability.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[2] J.-E. Kim, T. Abdelzaher, and L. Sha, "Budgeted generalized rate monotonic analysis for the partitioned, yet globally scheduled uniprocessor model," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2015), Seattle, WA, USA*. IEEE, April 2015.

[3] J. Rosén, A. Andrei, P. Eles, and Z. Peng, "Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip," in *Proc. of IEEE Int'l Real-Time Systems Symposium*, 2007, pp. 49–60.

[4] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware Support for WCET Analysis of Hard Real-Time Multicore Systems," in *Proc. of IEEE/ACM Int'l Symp. on Comp. Arch.*, 2009.

[5] M.-K. Yoon, J.-E. Kim, and L. Sha, "Optimizing Tunable WCET with Shared Resource Allocation and Arbitration in Hard Real-Time Multicore Systems," in *Proceedings of the 32nd IEEE Int'l Real-Time Systems Symposium*, 2011, pp. 227–238.

[6] B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, "Making shared caches more predictable on multicore platforms," in *Proceedings of the 25th Euromicro Conference on Real-Time Systems*, 2013.

[7] M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson, "Cache sharing and isolation tradeoffs in multicore mixed-criticality systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, 2015.

[8] A. K.-L. Mok, "Fundamental design problems of distributed systems for the hard–real–time environment," Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1983, ph.D. Thesis.

[9] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Real Time Systems Symposium, 1989., Proceedings.*, Dec 1989, pp. 166–171.

[10] D.-W. Park, S. Natarajan, A. Kanevsky, and M. J. Kim, "A generalized utilization bound test for fixed-priority real-time scheduling," in *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, 1995.

[11] E. Bini and G. C. Buttazzo, "Schedulability analysis of periodic fixed priority systems," *IEEE Trans. Comput.*, vol. 53, no. 11, Nov. 2004.

[12] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," *NASA Langley Technical Report*, Mar. 1999.

[13] J. Krodel, "Commercial off-the-shelf real-time operating system and architectural considerations," *Federal Aviation Administration*, Feb. 2004.

[14] P. Parkinson and L. Kinnan, "Safety-critical software development for integrated modular avionics," *White Paper, Wind River Systems*, 2007.

[15] J.-E. Kim, M.-K. Yoon, S. Im, R. Bradford, and L. Sha, "Optimized scheduling of multi-IMA partitions with exclusive region for synchronized real-time multi-core systems," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2013, pp. 970–975.

[16] J.-E. Kim, M.-K. Yoon, , R. Bradford, and L. Sha, "Integrated modular avionics (IMA) partition scheduling with conflict-free I/O for multicore avionics systems," in *Proceedings of the 38th IEEE Computer Software and Applications Conference*, 2014.

[17] J. H. M. Korst, E. H. L. Aarts, and J. K. Lenstra, "Scheduling periodic tasks," *INFORMS Journal on Computing*, vol. 8, no. 4, 1996.

[18] R. I. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proc. of the 24th IEEE RTSS*, 2005.

[19] L. Sha, "Real-time virtual machines for avionics software porting and development." in *Real-Time and Embedded Computing Systems and Applications, 9th International Conference, RTCSA 2003, February 18-20, 2003.*, J. Chen and S. Hong, Eds., 2003, pp. 123–135.

[20] J. Korst, E. Aarts, J. K. Lenstra, and J. Wessels, "Periodic assignment and graph colouring," *Discrete Appl. Math.*, vol. 51, no. 3, pp. 291–305, Jul. 1994.

[21] A. Al Sheikh, O. Brun, P.-E. Hladik, and B. J. Prabhu, "A best-response algorithm for multiprocessor periodic scheduling," in *Proc. of the 23rd Euromicro Conference on Real-Time Systems*, 2011, pp. 228–237.

[22] O. Kermia and Y. Sorel, "A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor." in *Proc. of ISCA 20th International Conference on Parallel and Distributed Computing Systems, PDCS'07*, 2007, pp. 1–6.

[23] D.-W. Park, S. Natarajan, A. Kanevsky, and M. J. Kim, "A generalized utilization bound test for fixed-priority real-time scheduling," in *Proceedings of the 2Nd International Workshop on Real-Time Computing Systems and Applications*, ser. RTCSA '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 73–.

[24] D.-W. Park, S. Natarajan, and A. Kanevsky, "Fixed-priority scheduling of real-time systems using utilization bounds," *Journal of Systems and Software*, vol. 33, no. 1, pp. 57 – 63, 1996.

[25] D. Chen, A. K. Mok, and T.-W. Kuo, "Utilization bound revisited," *IEEE Trans. Comput.*, vol. 52, no. 3, pp. 351–361, Mar. 2003.

[26] C.-G. Lee, L. Sha, and A. Peddi, "Enhanced utilization bounds for qos management," *IEEE Trans. Comput.*, vol. 53, no. 2, pp. 187–200, Feb. 2004.

[27] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, ser. RTSS '03, 2003, pp. 2–13.

[28] R. I. Davis and A. Burns, "An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems," in *Proceedings of Real-Time and Network Systems, RTNS*, 2008.

[29] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proceedings of the 4th ACM international conference on Embedded software*, 2004, pp. 95–103.

[30] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, 2003, pp. 151–158.

[31] M.-K. Yoon, J.-E. Kim, R. Bradford, and L. Sha, "Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling," in *Proceedings of the 16th ACM/IEEE Design, Automation, and Test in Europe (DATE 2013)*, 2013, pp. 1313 – 1318.

[32] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein, "Analysis of hierarchical fixed-priority scheduling," in *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, 2002, pp. 152–160.

[33] Y.-H. Lee, D. Kim, M. Younis, J. Zhou, and J. Mcelroy, "Resource scheduling in dependable integrated modular avionics," in *Proc. of the International Conference on Dependable Systems and Networks (FTCS-30 and DCCA-8)*, 2000, pp. 14–23.

[34] Y.-H. Lee, D. Kim, M. Younis, and J. Zhou, "Scheduling tool and algorithm for integrated modular avionics systems," in *Proc. of the 19th Digital Avionics Systems Conference (DASC)*, 2000.

[35] D. Tămaş-Selicean and P. Pop, "Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures," in *Proc. of the 32nd IEEE Real-Time Systems Symposium*, 2011, pp. 24–33.

[36] ——, "Optimization of time-partitions for mixed-criticality real-time distributed embedded systems," in *Proc. of the 2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2011, pp. 1–10.

[37] J. Rufino, J. Craveiro, and P. Verissimo, "Architecting robustness and timeliness in a new generation of aerospace systems," *Architecting dependable systems VII, LNCS*, vol. 6420, pp. 146–170, 2010.