A Schedulability Test for Software Migration on Multicore System

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 consists of a schedulability test and execution model. Its 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.**

I. 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: (i) Utilization bound analysis using task period information (ii) Conflict-free I/O scheduling, and (iii) 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 (*i.e.*, ratio of two task periods is square root of 2); actual periods are unlikely to resemble the worst case.

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 showed that this architecture significantly improves schedulability over TDMA and Periodic Server (PS), while maintaining isolation properties.

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 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.

Acknowledgement: This work was accomplished under ONR N000141410717, N000141612151 and NSF CNS 1302563, 1219064, 1329886 and 1618627, and US ARL W911NF0920053. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of sponsors.

II. Software Migration to Multicore Systems

This paper proposes a task execution model and a schedulability analysis test, motivated by the need to transition safety-critical software certified on single-core systems to multicore. Toward that end, we make three assumptions motivated by 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 each core. Our model attempts to remove timing dependencies across applications for their portability. We provide a solution to the schedulability problem. More details of this paper can be found in [2].

A. Task Execution Model

Schedulability Analysis with Task Period Data: In this paper, we assume that an allocation of applications to cores has already taken place. We are given M cores. In each core, m, we consider a set, $S_{(m)}$, of periodic tasks, indexed such that a lower number implies a higher priority in a core. 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 WCET of the task, denoted by $C_{m,i}$, may *not* be known. Once development is complete, the factors that affect WCETs, including timing interference and delay due to shared resources (*e.g.*, bus, cache, memory), are assumed to be abstracted (by methods such as [3]–[6]) and incorporated in the final WCETs. However, the utilization bound in our analysis allows for unknown WCETs and still obtains a bound on allowable application utilization.

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 [7]. I/O scheduling thus reduces to choosing phasing for the I/O transactions. I/O transactions are modeled as periodic tasks with period $T_{m,i}$ and execution time $IO_{m,i}$. They are analyzed as having the *highest priority*, and being globally scheduled in a conflict-free manner, such that only a single section executes at a time *across all cores*. Hence no I/O on any core is blocked, preempted, or otherwise delayed by I/O in 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. We use *I/O section* to refer to such an object, having total duration, $IO_{m,i}$, for task $\tau_{m,i}$. 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. The duration can be affected by interference on shared resources such as bus and cache. We assume such interferences can be bounded by methods such as [3]–[6] 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.

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, 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 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, $\sum_{\forall \tau_{m,i} s.t. \alpha(\tau_{m,i})=\alpha_z} u_{m,i} \leq B_z$. Observe that, the budget, B_z, of α_z is *a design-time constraint*, not a run-time 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 for individual tasks will be enforced, thereby indirectly enforcing application budget compliance. WCET-enforced tasks will be scheduled using fixed priority scheduling. This mechanism indirectly allows enforcement of resource budgets, without employing resource partitioning mechanisms at run-time. [8] The mechanism avoids inefficiencies of resource-partitioned systems such as the PS 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). Tasks' schedulability is analyzed in a fixed-priority fashion no matter which application they belong to.

[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 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 breaking an application across cores. To do otherwise would invite additional complications with no 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.

B. An Equivalent Independent Task Model

We note that the task model can be transformed to one of scheduling independent tasks on a core. We assume I/O to be non-preemptible, and the following precedence constraints involving execution and I/O tasks to be satisfied for every invocation of a task: (i) the processing does not begin until after the sub-task of input is complete, (ii) the sub-task of output does not begin until after the processing is completed, and (iii) the sub-task of input for the next period's invocation of the task does not begin until after the sub-task of output from the current period's invocation is completed. (The very first invocation of the task in the global schedule does not require a predecessor.) Theorem 1 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.*

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. ■

Hence, we eliminate cross-core interference due to I/O and obtain the utilization bound for an allowable limit on total task execution times including interference effects. Then, our schedulability problem is distilled into two subproblems: (i) Ensure that I/O sections are scheduled strictly periodically and conflict-free. (ii) Analyze task schedulability on each core separately.

III. Schedulability Analysis with Budget Constraints

A. 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*[3] task set, $\sum_{i=1}^{n} u_{m,i}$, over all possible values of computation times $C_{m,i}$ for $1 \leq i \leq n$.

Consider the critical time zone[4] of task $\tau_{m,n}$, of α_z, where the task arrives at time $t{=}0$ together with its all higher priority tasks. Suppose that the invocations in this interval are a critically schedulable task set. Since scheduling is work-conserving, the interval $0 \leq t < D_{m,n}$ is continuously busy. Also, budget constraints limit the utilization of all tasks in α_z up to B_z. However, the two constraints conflict 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 remove the budget constraint for the application $\alpha(\tau_{m,n})$. Let the resultant bound be $U^n_{m,released}$. 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,

$$U^n_{m,released} \leq U^n_{m,bound}. \tag{1}$$

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, $\forall \alpha_z \in \mathcal{A}^n_{(m)}, \alpha(\tau_{m,i}) \neq \alpha_z, \sum_{i=1}^{n-1} u_{m,i} \leq B_z$. 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 (1), $B^n_{m,total} \leq U^n_{m,released} \leq U^n_{m,bound}$, so $B^n_{m,total} \leq U^n_{m,bound}$. It means that $\tau_{m,n}$ is schedulable by the definition of schedulability bound. ■

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.

B. The Utilization Bound

The utilization bound, $U^n_{m,released}$, 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. It is formulated as a linear programming (LP) problem.

[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 from the critical instant to the deadline. As all higher priority tasks and $\tau_{m,n}$ release at time 0, collectively their maximum possible 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]: $\forall 1 \leq k \leq n, \forall 1 \leq l \leq \lfloor \frac{D_{m,n}}{T_{m,k}} \rfloor$,

$$C_{m,n} + \sum_{i=1}^{n-1} \lceil \frac{l \cdot T_{m,k}}{T_{m,i}} \rceil C_{m,i} + \sum_{i=1}^{n} \lceil \frac{l \cdot T_{m,k}}{T_{m,i}} \rceil IO_{m,i} \geq l \cdot T_{m,k}. \tag{2}$$

Then, $\tau_{m,n}$, $\sum_{k=1}^{n} \lfloor \frac{D_{m,n}}{T_{m,k}} \rfloor$ constraints are generated. The number can be reduced by using $\mathcal{P}_i(t)$ presented in (6) in [11]. Hence, not all arrivals at $l \cdot T_k$ ($\forall 1 \leq k \leq n, \forall 1 \leq l \leq \lfloor \frac{D_{m,n}}{T_{m,k}} \rfloor$) but only the subset of them, $t \in \mathcal{P}_{n-1}(D_{m,n})$, are considered. We formulate the LP problem of finding $U^n_{m,released}$ for a given task $\tau_{m,n}$:

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

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

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.

IV. Conflict-Free I/O

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

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

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:

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

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. III).

V. Evaluation

A. I/O schedulability

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

[3]A task set is critically schedulable if any increase in execution time of any task would make the set unschedulable [1].

[4]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].

Fig. 1 shows task set counts 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 achieve much higher utilization; nevertheless, due to their non-preemptivity, the I/O sections cannot always be schedulable even when utilization is 100%. Besides that, 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%.

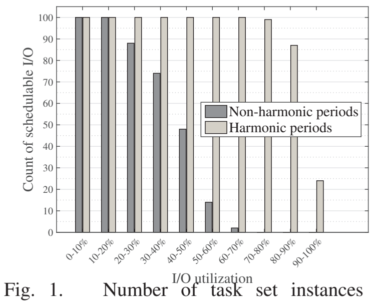

Fig. 1. Number of task set instances that have schedulable I/O when task periods are non-harmonic vs. harmonic.

B. 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 Fig. 2, 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. V-A above, but kept I/O utilization in 1%-5%. We generated 1,500 task sets with evenly-distributed numbers of tasks per core, (100 instances per task count). For comparison, the L&L utilization bound is plotted as well (line plot), when deadlines are equal to periods. As seen from the result, the bounds by our approach are above the L&L bound, since the analysis takes advantage of task period information.

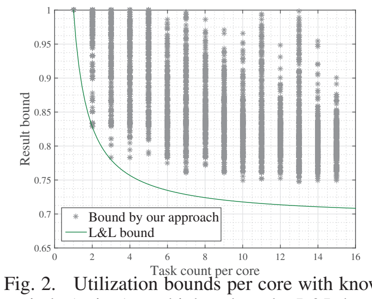

Fig. 2. Utilization bounds per core with known periods (points) are higher than the L&L bound (line) (when deadline is equal to period).

C. Our Approach vs. Other Resource Partitioning Mechanisms

Parameter values are the same as in the previous cases except that we vary core count from 2 to 8 – 300 instances for each core count. Fig. 3 shows the number of schedulable task set instances. For an instance to be schedulable, all tasks in all applications (*i.e.*, partitions and servers) on every core must be schedulable. In PS approach, each application is assigned its own server. Server period is the gcd of the periods of tasks that belong to the server, which is essentially a best-case assumption for PS. Server utilization 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 used here by PS are also applied in our approach and TDMA as well. When TDMA is used, each application is assigned a TDMA partition. Partition utilization is equal to the application budget. Then gcd 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]. In Fig. 3, see that our approach schedules more task sets than TDMA or PS 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 PS is inferior to TDMA, since in some instances PS can successfully schedule them while TDMA does not and vice versa.

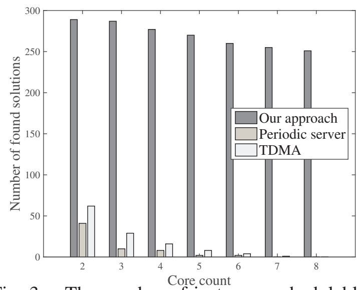

Fig. 3. The number of instances schedulable by our approach, PS, and TDMA.

REFERENCES

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. of the ACM*, vol. 20, no. 1, Jan. 1973.
[2] J.-E. Kim, R. Bradford, T. Abdelzaher, and L. Sha, "Schedulability analysis for certification-friendly multicore systems," Dept. of Computer Science, University of Illinois at Urbana-Champagin, technical report, 2016, http://hdl.handle.net/2142/94736.
[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 RTSS*, 2007, pp. 49–60.
[4] 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 RTSS*, 2011, pp. 227–238.
[5] 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.
[6] 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.
[7] 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, MIT, 1983, ph.D. Thesis.
[8] J.-E. Kim, T. Abdelzaher, and L. Sha, "Budgeted generalized rate monotonic analysis for the partitioned, yet globally scheduled uniprocessor model," in *Proc. of IEEE RTAS*, Apr. 2015.
[9] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proc. of Real Time Systems Symposium*, Dec. 1989.
[10] D.-W. Park, S. Natarajan, A. Kanevsky, and M. J. Kim, "A generalized utilization bound test for fixed-priority real-time scheduling," in *Proc. of the Int'l 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 *Proc. of DATE*, 2013.
[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 *Proc. of the 38th IEEE COMPSAC*, 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 *RTCSA*, Feb. 2003.