Contention in Multicore Hardware Shared Resources: Understanding of the State of the Art

Gabriel Fernandez[1,2], Jaume Abella[2], Eduardo Quiñones[2], Christine Rochange[4], Tullio Vardanega[5], and Francisco J. Cazorla[2,3]

1 **Universitat Politècnica de Catalunya**
2 **Barcelona Supercomputing Center**
3 **Spanish National Research Council (IIIA-CSIC)**
4 **IRIT, University of Toulouse**
5 **University of Padova**

Abstract

The real-time systems community has over the years devoted considerable attention to the impact on execution timing that arises from contention on access to hardware shared resources. The relevance of this problem has been accentuated with the arrival of multicore processors. From the state of the art on the subject, there appears to be considerable diversity in the understanding of the problem and in the "approach" to solve it. This sparseness makes it difficult for any reader to form a coherent picture of the problem and solution space. This paper draws a tentative taxonomy in which each known approach to the problem can be categorised based on its specific goals and assumptions.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Contention, Multicores, WCET Analysis

Digital Object Identifier 10.4230/OASIcs.WCET.2014.31

1 Introduction

At a conceptual level, the intent of timing analysis is to provide, at low-enough cost, a WCET bound for programs running on a given processor. Ideally, the transition to multicore processors should allow industrial users to achieve higher levels of guaranteed utilization, together with attractive reduction in energy, design complexity, and procurement costs. Unfortunately however, the architecture of multicore processors poses hard challenges on (worst-case) timing analysis. The interference effects arising from contention on access to processor-level resources in fact need far greater attention than in the singlecore case, as much greater is the arbitration delay and state perturbation that resource sharing may cause, and consequently much greater is the "padding" factor that needs to be captured in the computed bounds to compensate for the relevant effects.

From a bottom-up perspective of the system architecture, the utilization that individual tasks make of many of those shared resources at the processor level is too low to justify a dedicated use of them. Efficient use of those resources, which is needed for decent average performance, requires sharing. However, resource sharing shatters tightness in timing analysis and endangers the trustworthiness of the bounds computed with it.

Different approaches (angles) have been considered to address the timing effects of contention for shared resources. However, there is evident lack of common understanding of the problem space, in terms of processor features, and of the assumptions made to solve it.

Contribution. This paper aims to contribute to the community understanding of how bounds can be computed for the execution time of software programs running on processors with shared hardware resources. Special attention is given to capturing the assumptions made by each considered technique, as a way to gage the applicability of its results. The authors' intent is to capture the principal angles of attack to the problem, as they emerge from the state of the art, flag-shipped by specifically representative strands of work, without necessarily achieving exhaustive coverage of the literature. In the authors' opinion, whereas the number of works on multicore resource contention is vast and growing, the community lacks a common understanding of the big picture, which would be needed to determine which new techniques are needed to best address the problem from the perspective of benefiting application.

2 A tentative taxonomy of state-of-the-art techniques to analyse the timing impact of resource contention

Under the umbrella term of resource contention, we capture the various forms of timing interference that software programs suffer owing to access to shared hardware resources. Notably, our analysis does not cover the contention on access to software resources. Furthermore, contentions arising from parallel execution of a software program fall outside of our analysis and are recognized in Section 4 as an important emerging ramification of the problem.

The challenge of contention in multicore processors has been addressed with various approaches. This paper classifies them in four broad categories, dependent on where they seem to direct their focus: (1) on system considerations, which address the contention problem top down, from the software perspective; (2) on WCET considerations, which take the opposite view, studying how contention phenomena affect the timing behaviour of the software; (3) on architecture considerations, which devise processor features and arbitration policies that help achieve composable timing behaviour; and on (4) Commercial Off-The-Shelf (COTS) considerations, which propose processor-specific ways to deal with processor-specific contention and arbitration features. We discuss the approaches in each category in isolation and we break them down into subgroups where appropriate.

2.1 System-centric techniques

System-centric techniques take a top-down approach to the problem. Those techniques take an off-chip, hence coarse-grained, perspective. Off-chip resources have longer latency and also a higher degree of visibility from the software standpoint than on-chip resources. For instance, at software level one can easily tell where a set of addresses is mapped to memory, but it is (much) harder to determine which data item is (or is not) in cache at a given time. In fact, the cache impact has characteristics that can be captured, from different angles and with different precision, with techniques that we classify in different categories of our taxonomy (system-centric and WCET-centric). Besides this cross-boundary overlap, the techniques in this category predominantly focus on off-chip resources.

We single out three angles worthy of specific discussion: timing analysis frameworks; access scheduling and allocation; and works on COTS architectures.

2.1.1 Timing analysis frameworks

In general, these techniques assume that on-chip shared resources (e.g. core-to-cache bus, caches, etc) are replicated or partitioned across cores, so that software programs allocated

to a core suffer no contention on access to on-chip resources. These techniques also assume that analysis frameworks model off-chip shared resources in isolation and provide worst-case access timing bounds for them. The impact of contention is only considered for the off-chip resources and it is captured compositionally[1], when the WCET of the software program, determined assuming no contention, is increased by delay factors that consider the sources of off-chip contention in the presence of co-runners.

The shared resources considered in this class of approaches are assumed to process one request at a time. It is also assumed that the corresponding services cannot be pre-empted (or split or segmented). It is further assumed that the requests are synchronous so that the requesting task is stalled while the access request is served. In reality, some requests are asynchronous not blocking the calling task execution. The analysis focuses on individual tasks, whose program units are logically divided into blocks for which maximum and minimum access bounds and execution time bounds are derived.

For approaches in this class, the access to the shared resource is assumed to be arbitrated by either a TDMA bus [9] or a dynamic arbitration bus [29] or else an adaptive bus arbiter [10]. For TDMA buses, focus is on determining the worst-case alignment of the requests in the TDMA schedule. As the bus schedule is static, co-running tasks do not affect one another's execution time, which makes their execution time composable with respect to the bus. The fact that service is assumed run-to-completion and that requests do not overlap simplifies the problem.

For dynamic arbiters, the workload that a task places on the shared resource affects the access time of the co-running tasks, which breaks time composability. This type of arbiters has generated a research line of their own. Several authors [4, 29] propose how to derive bounds for the number of accesses per task in a given period of time. The timing analysis for an individual task therefore depends on the request workload generated by the co-runners in that time duration. Interestingly, while the number of accesses that a task generates to the resource can be considered independent of the co-runners as long as caches are partitioned, the task's frequency of access depends on how often the co-runners delay the task's requests. The cited models capture that dependence.

Adaptive arbiters (as in, e.g., FlexRay) exhibit a window with per-requester slot scheduling combined with a window in which requests are dynamically arbitrated; this trait makes them show characteristics that we have seen above as distinctive for static and dynamic arbiters.

The authors of [30] provide a useful survey of how time-deterministic approaches to bus arbitration and scheduling for multicore processors can be captured, compositionally, by timing analysis techniques. The cited work also presents benchmark-based empirical evidence of the degradation that TDMA arbitration causes to average-case performance in comparison to other techniques with acceptable characteristics in terms of time determinism.

2.1.2 Task scheduling and allocation

The state-of-the-art approaches to multicore scheduling and schedulability analysis that match the techniques which fall in this category can be grouped in two sets: those that ignore contention issues at their level and leave it to WCET analysis; and those that consider

[1] We use the term compositional to signify that some property of an individual part of the system can only be determined on (assumed) knowledge of the constituents of the system. This is in contrast with the term composable, which regards those properties of an individual part that can be determined considering that part in isolation and hold true on composition into the system [28].

it as a factor of influence to task allocation, which is adjusted to attain increased schedulable utilization.

There essentially exist three classes of scheduling algorithms, which differ in the way they assign tasks to cores [5]. Partitioned and global scheduling fall on the respective extremes of the spectrum: the former statically maps tasks to cores, so that a task can only be scheduled on the core it has been assigned to; the latter allows tasks to migrate jobs from one core to another and does not leave any core idle if there is some work to be done (work conserving), at the expense of possibly costly task (job) migrations. The former risks considerable under-utilization of the processing resources, especially for tasks with medium to high loads. The middle of the spectrum is occupied by clustered or semi-partitioned algorithms, which – with various techniques and for different goals – only allow or cause statically-set groups of tasks to migrate within statically or dynamically determined subsets of cores.

Contention oblivious. The principal works on scheduling algorithms and associated schedulability analysis for multicore processors assume that the WCET of all tasks is given in input. Thus, they assume that the WCET bounds may be determined before decisions are made on task mapping to cores and on scheduling at run time. This is tantamount to postulating that the WCET bounds are composable, that is to say, free from variations determined by the presence of contenders in the system. Ironically, the only plausible way in which WCET bounds can actually be made to be composable for use in schedulability analysis for multicore processors is by increasing them compositionally, by a factor determined by given patterns of conflicts that upper bound the actual contention delays suffered at run time. In essence, the approaches of this kind escape the intrinsic (and painful) circularity between the dependence of WCET analysis on knowledge of the contenders and the dependence of schedulability on knowledge of the WCET of the tasks in the system, by inflating the WCET budgets so that they can always be trusted to upper bound the actual costs.

Contention aware. Techniques such as [32, 12] focus on the shared last-level cache as one of the main resources in which contention occurs. The cited works benefit from hardware proposals that split the cache into different ways or allocate program data into different pages (colors) so that each task is limited to use a subset of the sets in cache, thereby reducing conflicts.

These works often assume partitioned scheduling for software programs, so that conflicts can be determined in a less pessimistic way, and focus their attention on devising cache-aware allocation algorithms that consider the mapping of tasks to cores determined by partitioning. Some of the works focus on how to assign colors (i.e. set partitions) to the tasks. It is also the case that works in this class do not address the contention occurring in other shared resources like the memory.

Other works [23] build on hardware proposals that control the interaction in several hardware resources (e.g. on-chip bus, cache and memory controller) in addition to the cache. These proposals also consider task allocation and scheduling.

2.2 WCET-centric techniques

WCET-centric techniques determine the impact of contention in the access to shared resources as part of WCET analysis. For multicore architectures, shared resources include cache memories, buses and memory controllers, but some approaches have also been designed to support intra-core resource sharing (e.g., pipeline and functional units in multithreaded

cores [3]). The objective of WCET-centric techniques is to derive safe stall times that can be accounted for at instruction-level timing analysis. We distinguish between the approaches that consider all the competing threads/tasks together to exhibit the possible interleavings of their respective accesses to the shared resource, from those that exploit a static allocation of slots among cores. In the latter category, some contributions include WCET-based strategies to optimize the mapping/scheduling of threads/tasks to cores to optimize the global WCET and/or to improve schedulability.

2.2.1 Joint analysis of concurrent tasks/threads

One way to identify how contention may impact the WCET of a task is to combine the analyses of concurrent tasks to identify where they can interfere. Two kinds of interference are considered here: spatial (tasks share storage, e.g. a cache) and temporal (tasks share bandwidth, e.g. a bus). Both incur additional delays.

Techniques that address spatial contention start by perform individual tasks analyses, then determine how contention affects their results. More precisely, they determine which cache lines used by one task might be replaced by another task in a shared L2 instruction [14][33] or data [11] cache. The analysis of contention does not account for the exact respective timings of tasks (then could be valid for any schedule, provided all possible concurrent tasks are known at analysis time). However, [33] improves the accuracy of the analysis by considering constraints on task scheduling (non-preemptive, priority-based, with task inter-dependencies), which allows bounding tasks lifetimes and limits the opportunities for contention.

To account for temporal conflicts and derive instruction timings, possible interleavings of (statically-scheduled) threads must be explored. Several approaches use timed automata to represent both the tasks and the state-based behaviour of hardware components. All these automata are combined and model checking techniques are used to determine the WCET though a binary search process. [6] focuses on the shared L2 cache with fixed cache miss latency. A shared bus with First-Come-First-Served (FCFS) or TDMA arbitration is analysed in [19]. The weakness of these approaches is in the huge number of states to be handled.

2.2.2 Independent analysis of tasks/threads

Some techniques leverage the deterministic guarantees offered by the underlying hardware on access to a shared resource. Thanks to such guarantees, they can analyse the WCET of one task/thread independently of the concurrent workload.

The impact of arbitration delays on a TDMA bus with uniform slot size is explored in [31]. The cited work presents an approach to evaluate the misalignment of accesses with TDMA slots (TDMA offsets). A TDMA-composable system is assumed: arbitration delays neither impact instructions that do not access the bus nor the bus access time (except for the arbitration delay).

Some of the hardware solutions to enforce access guarantees do not offer equal opportunities to all threads. Cache partitioning techniques may allocate partitions with different sizes [23]. Bus arbiter may grant a different number of slots to each core, as in [18] or [26]. Those techniques use these mechanisms to increase the performance achievable by combined task-to-core allocation and scheduling decisions, especially in the case of unbalanced workloads (with variable demand levels to the shared resource). Performance benefits are obtained as a result of reducing the WCET bound predictions for the affected tasks. As we

noted in Section 2.1, our taxonomy is not clear-cut enough to place some of these techniques uniquely in one class, as they might arguably also belong to the system-centric group.

2.3 Architecture-centric techniques

Several hardware design paradigms have been proposed to deal with the inter-task interference caused by contention for shared hardware resources. Four topical approaches can be singled out in this group: the time-triggered architecture [36]; PRET [37]; CompSOC [8]; and MERASA [38].

One of the differentiating elements for these approaches is whether they achieve composability at the level of the WCET bounds that they allow computing or at higher levels of abstraction. The objective of the former solution is to support determining WCET bounds for individual tasks in isolation, independently of the activity of their co-runners. When that is guaranteed, the execution time of a task may well suffer variations caused by contention effects caused by some of its co-runners, but its WCET estimate stays valid. With the latter type of solutions, composability is achieved by regulatory mechanisms operating at run time, and thus with effect on the task execution time. Those regulatory mechanisms ensure that the activity of the co-runners cannot affect the response time of the hardware shared resources. This form of composability may place more requirements on the processor hardware than the former approach. In general it requires that the access time to a hardware shared resource stays always the same irrespective of the actual load of the system. To that end, a resource that might respond ahead of time is stalled until the agreed latency for the request is reached.

From another angle, it is worth noting that a trade off arises as a consequence of the observation that the pursuit of time composability always comes at the cost of some (over-provisioning) pessimism. The effect of this (static) over-provisioning allows tightening the WCET bounds, because they eradicate sources of variations, but at the cost of renouncing the true meaning of time composability (as independence from the presence of contenders), which is central to the incremental verification needs of integrated architectures such as Integrated Modular Avionics (IMA).

Somewhat orthogonal to the discussion above, the focus of several proposals is to upper bound the access time to hardware shared resources, either indirectly, by guaranteeing pre-determined bandwidth on access to the resource, or directly by ensuring bounds on the access time (comprised of the wait time preceding access upon request, and the actual service time).

The techniques of interest from this angle vary for *stateless* and *stateful* resources. Stateless resources have an access time that is not or only very modestly dependent on execution history. A single-cycle latency bus is a typifying example of resources of this kind. If the bus had a two-cycle latency, then the service time of a request might depend on whether the preceding request was sent the cycle before the current one gets ready. Caches are a difficult exemplar of stateful resources. This is because the state-dependent effect builds up with history of execution, which causes analysis to have to keep track of the full history of access. Truncated information requires conservative assumptions to be made. This difficulty explains why the typical solution proposed for caches consists in splitting its space in small areas assigned to individual tasks, so that history becomes much smaller (and free of conflicts with co-runners) and thus easier to trace. This can be done dividing the cache into different banks or different ways [22].

The most prominent stateless resources on which the real-time community has focused are network-on-chip (NoC) and memory controllers. For the interconnection network, proposals

exist which range from from simple buses [35] or rings [20] to more complex solutions such as those described in [41]. All share the goal to provide some type of bound to the longest time a request has to wait to get access to the resource. For the memory controller, proposals with the same goal exist [1, 21], though the actual solutions are more complex since the state retention is higher.

2.4 COTS-based techniques

The goal of several works focusing on real hardware is to analyse how amenable a given multicore design is for real-time analysis. To that end authors analyse different shared resources as well as their impact on execution time. For many resources the manuals of the processor under analysis rarely provide all the required information to analytically derive those bounds. As a result, the way in which the authors derive bounds is different from previous approaches and it is based on experimentation on the specific architectures under analysis [25, 39, 17]. These works include analysis of the FreeScale P4080 and some FPGA versions of the Aeroflex Gaisler LEON4.

Another set of works is carried out at an analysis level providing understanding of the timing behaviour of hardware shared resources and the challenges they bring to timing analysability [13, 27, 42]. Finally, some of the works on software-cache partitioning (page colouring) have been done for processors like the ARM Cortex A9 [12].

3 Critique

This section reviews the techniques captured in the taxonomy presented in Section 2 against multiple criteria including: (1) the presence of overlaps between them; (2) the presence of gaps among them; (3) the realism of their assumptions; (4) the challenges in taking that technique to industrial use; and (5) the relation between the confidence on the bounds determined by timing analysis and the assurance guarantees suitable for the application domain.

Much like the proposed taxonomy, the review discussed here is not meant to be exhaustive. It therefore does not cover all criteria for all techniques. Instead, it only aims at singling out specific problematics that we consider to need particular attention by the prospective user and further study by the research community.

System-centric techniques. The principal limitation with this class of techniques stems from their resting on two strong assumptions: that programs can be statically subdivided into (super)blocks for which bounds on resource usage can be derived; and that only one shared resource needs attention, which also does not support split transactions. The former restricts applicability to programs that can be divided into blocks for which maximum and minimum access bounds and execution time bounds can be derived. Further, working at block level may increase pessimism because every block is given a single worst-case cost value, which may be higher than the actual cost in the worst-case traversal of that block as taken by the program. The latter assumption reduces the applicability of the solution against increasingly common hardware.

For dynamic arbiters, the critical factor is in the dependence of their timing analysis on the request workload generated by the co-runners of the program of interest in a given time duration. On the one hand this trait reduces pessimism since the durations in which conflicts on access may occur can be better determined. On the other hand, it breaks time composability and resorts to compositionality. The latter defect may be a serious impediment

to incremental verification, which is a prerequisite to high-criticality domains (e.g., avionics). Budgeting in advance for the co-runners is obviously one countermeasure to that, but at the direct cost of over-provisioning.

WCET-centric techniques. The main challenge for this class of techniques arises from having to find tractable ways to analyse increasingly complex hardware. The abstract interpretation approach on which those techniques rely is inherently exposed to the state explosion problem, which is dramatically worsened by the way in which the architecture of modern processors cause the timing behaviour of several resources to exhibit possibly large jitter, extremely sensitive to the history of execution [42]. This dependence obviously cumulates bottom-up and manifests in very complex ways at software level.

As an example we consider a TDMA bus, whose timing behaviour is easy to model with three main parameters: window size, number of contenders, and slot size per contender. Interestingly, the state space for even such a simple model is already not negligible: when the exact time of an access request cannot be determined in fact, a conservative assumption must be made on when access will be granted (which inflates pessimism) or multiple candidate access times are considered, which causes multiple states to be contemplated upward in the analysis. As more complex NoC architectures are adopted by modern multiprocessors, more parameters will be needed to model the sources of contention, with inordinate increase in the complexity and cost of the analysis tools.

Architecture-centric techniques. A recurrent question on the viability of the techniques in this class is whether the hardware design that they propose in the intent to favour time analysability, will ever reach the market. This is a question of economics that equally applies to all research domains that propose new hardware architectures. However, it is especially important to the real-time systems domain, which holds a tiny niche of the market size, in comparison to consumer products, without sufficient critical mass to swing the prevailing design criteria from optimized for the average case to well-behaved in the worst case.

This is a long-known challenge for the real-time systems community. Fortunately, perseverance and authority have shown able to win some battles, so that some of the proposed designs (e.g., cache partitioning) are indeed retained in real processors. Our view here is that the changes proposed for the bus and the memory controller are simple enough so that they can be implemented in production with moderate effort and cost, for tangible benefits on timing analysability. Whether or not that will actually happen remains to be seen.

In general all hardware approaches assume processor designs without timing anomalies. It is interesting wondering, whether processor can be made simple enough to assure freedom from timing anomalies, without this causing detriment to the attainable performance. Architectural solutions will have to be devised that combine those two objectives harmoniously, which is not the case yet with the dominant approaches to multi- and manycore processor architectures.

COTS-based techniques. The techniques that belong in this class face the challenge that the architectural properties needed to provide full time isolation or time predictable interaction among processor cores cannot be had owing to the lack or inaccuracy of specification information or IP restrictions. Various approaches have been proposed to live with the consequent uncertainty, which all require building confidence arguments that accord with the requirements and practices of the application domain. The work in [34] makes an interesting review of how safety assurance guarantees relate to stipulating bounds on execution time.

4 Other aspects of interest

In this section we briefly touch upon two other aspects that, for different reasons, are tangent to questions addressed in this paper. One aspect, parallel programming, intrinsically enabled and called for by multicore processors, presents a novel, emerging challenge to bounding contention effects. The other aspect, with interesting potential and important ramifications, stems from shifting the angle of attack to the timing analysis problem, from finding a single value, the smallest possible computable upper bound, for all possible executions of a software program, to determining a probability function whose tail can be cut at the exceedance threshold of interest to the system.

Parallel applications. Communication of data in message-passing and synchronisations in shared-memory programming induce delays that must be accounted for in execution times. The focus is on deriving the WCET of the longest thread.

Two kinds of synchronization exist. (1) Mutual exclusion is very similar to accessing a hardware shared resource that can serve a single thread at a time. Computing the worst-case stall time of a thread at a critical section is analogous (when threads are served in a FIFO order) to computing the worst-case delay to a round-robin bus [15]. Stall times can then be integrated to instruction-level timing analysis. Another approach is to use timed automata and a model checker, as in [6]. In [7], a shared-memory parallel programming language is introduced and a fix-point analysis is able to identify all the possible thread interleavings at critical sections. (2)Progress synchronisation includes barriers as well as condition signalling and blocking message passing. Collective synchronisations (barriers), where all threads meet, are easier to consider since the goal is to compute the WCET of the longest thread, i.e. the last one to reach the barrier [15]. For point-to-point synchronisations (condition signalling or message passing) however, stall times depend on the respective progress of the threads. In [40], parallel applications where threads communicate through message passing are considered. A joint analysis is proposed, where the analysis of worst-case communication times is integrated into the analysis of the global WCET. The approach consists in merging the control flow graphs of parallel threads, then adding edges to model the synchronisations (dependencies) related to sending/receiving messages.

Some system-centric approaches have been extended to parallel fork-join applications and decide altogether the allocation of threads' memory in caches, the scheduling of threads' accesses to the shared bus and the scheduling of the threads themselves to the cores [2].

The probabilistic approach. Timing analysis techniques can be broken down into *deterministic*, which produce a single WCET estimate, and *probabilistic* that produce multiple WCET estimates with associated exceedance probabilities. It is noted that our discussion above has focused on standard (deterministic) timing analysis techniques. While both deterministic and probabilistic approaches try to reach time predictability, the former do so by advocating for hardware and software designs that are deterministic in their execution time, while the latter advocates for hardware and software designs that have a randomized timing behaviour, to produce WCET estimates that can be exceeded with a given *probability*.

The probabilistic approach deals with contention by means of time-randomised bus arbitration policies [16] as an alternative to deterministic policies such as round robin. Similarly, in [24] it is proposed a time-randomised shared cache for which impact of contention among co-running tasks can be determined. The main feature of this cache is that it does not split the cache, either into ways or sets, to prevent the interaction among co-running

tasks. Instead, it controls how often tasks evict data from cache as a way to bound the impact of contention on tasks' WCET estimates.

5 Conclusions

A wealth of relevant literature addresses the problem of finding a bound on the timing effect of contention on access to hardware shared resources in modern multicore processors. The industrial practitioner, and the researcher alike, who approach that body of knowledge without a preconceived solution in mind, may have serious difficulties in seeing the "big picture" of what options are possible and at what consequences. This paper sketches an initial taxonomy of the principal approaches that appear in the state of the art, and discusses gaps and overlaps among them.

Acknowledgements. The research leading to this work has received funding from: COST Action IC1202, Timing Analysis On Code-Level (TACLe); and the parMERASA and PROXIMA grant agreements (respectively no. 287519 and 611085 from the Seventh Framework Programme [FP7/2007-2013]). This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence.

References

1 B. Akesson et al. Predator: a predictable SDRAM memory controller. In *CODES+ISSS*, 2007.
2 A. Alhammad and R. Pellizzoni. Time-predictable execution of multithreaded applications on multicore systems. In *DATE*, 2014.
3 P. Crowley and J.-L. Baer. Worst-case execution time estimation for hardware-assisted multithreaded processors. In *HPCA-9 Workshop on Network Processors*, 2003.
4 D. Dasari and V. Nelis. An analysis of the impact of bus contention on the WCET in multicores. In *HPCC-ICESS*, 2012.
5 R. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4), 2011.
6 A. Gustavsson et al. Towards WCET analysis of multicore architectures using UPPAAL. In *Workshop on WCET Analysis*, 2010.
7 A. Gustavsson et al. Toward static timing analysis of parallel software. In *Workshop on WCET Analysis*, 2012.
8 A. Hansson et al. Compsoc: A template for composable and predictable multi-processor system on chips. *TODAES*, 2009.
9 A. Schranzhofer et al. Timing analysis for TDMA arbitration in resource sharing systems. In *RTAS*, 2010.
10 A. Schranzhofer et al. Timing analysis for resource access interference on adaptive resource arbiters. In *RTAS*, 2011.
11 B. Lesage et al. Shared data caches conflicts reduction for wcet computation in multi-core architectures. In *RTNS*, 2010.
12 B. Ward et al. Making shared caches more predictable on multicore platforms. In *ECRTS*, 2013.
13 D. Dasari et al. Identifying the sources of unpredictability in COTS-based multicore systems. In *SIES*, 2013.
14 D. Hardy et al. Using bypass to tighten wcet estimates for multi-core processors with shared instruction caches. In *RTSS*, 2009.

15 H. Ozaktas et al. Automatic wcet analysis of real-time parallel applications. In *Workshop on WCET Analysis*, 2013.
16 J. Jalle et al. Bus designs for time-probabilistic multicore processors. In *DATE*, 2014.
17 M. Fernández et al. Assessing the suitability of the NGMP multi-core processor in the space domain. In *EMSOFT*, 2012.
18 M.-K. Yoon et al. Optimizing tunable WCET with shared resource allocation and arbitration in hard real-time multicore systems. *RTSS*, 2011.
19 M. Lv et al. Combining abstract interpretation with model checking for timing analysis of multicore software. In *RTSS 2010*, 2010.
20 M. Panić et al. On-chip ring network designs for hard-real time systems. In *RTNS*, 2013.
21 M. Paolieri et al. *An Analyzable Memory Controller for Hard Real-Time CMPs* . Embedded System Letters (ESL), 2009.
22 M. Paolieri et al. Hardware support for WCET analysis of hard real-time multicore systems. In *ISCA*, 2009.
23 M. Paolieri et al. IA3: An interference aware allocation algorithm for multicore hard real-time systems. In *RTAS '11*, 2011.
24 M. Slijepcevic et al. Time-analysable non-partitioned shared caches for real-time multicore systems. In *DAC*, 2014.
25 P. Radojković et al. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. *ACM TACO*, 2012.
26 R. Bourgade et al. Predictable two-level bus arbitration for heterogeneous task sets. In *ARCS*, 2013.
27 R. Wilhelm et al. Designing predictable multicore architectures for avionics and automotive systems. In *Workshop on Reconciling Performance with Predictability (RePP)*, 2009.
28 S. Hahn et al. Towards compositionality in execution time analysis–definition and challenges. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2013.
29 S. Schliecker et al. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *DATE*, 2010.
30 T. Kelter et al. Evaluation of resource arbitration methods for multi-core real-time systems. In *Workshop on WCET Analysis*, 2013.
31 T. Kelter et al. Static analysis of multi-core TDMA resource arbitration delays. *Real-Time Systems*, 2013.
32 X. Zhang et al. Towards practical page coloring-based multicore cache management. In *EuroSys*, 2009.
33 Yan Li et al. Timing analysis of concurrent programs running on shared cache multi-cores. In *RTSS*, 2009.
34 P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. In *Workshop on Mixed-Criticality Systems*, 2013.
35 J. Jalle et al. Deconstructing bus access control policies for real-time multicores. In *SIES*, 2013.
36 H. Kopetz and G. Bauer. The time-triggered architecture. *Proc. of the IEEE*, 91(1), Jan 2003.
37 I. Liu et al. A PRET architecture supporting concurrent programs with composable timing properties. In *44th ACSSC*, 2010.
38 MERASA. *EU-FP7 Project: www.merasa.org.*
39 J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. In *EDCC*, 2012.
40 D. Potop-Butucaru and I. Puaut. Integrated Worst-Case Execution Time Estimation of Multicore Applications. In *Workshop on WCET Analysis*, 2013.

41 M. Schoeberl et. al. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *NoCS*, 2012.
42 R. Wilhelm and J. Reineke. Embedded Systems: Many Cores – Many Problems. In *SIES*, 2012.