
Article:
Maiza, Claire, Rihani, Hamza, Rivas, Juan et al. (3 more authors) (2019) A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. ACM Comput. Surv.. ISSN 0360-0300

https://doi.org/10.1145/3323212

A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems

CLAIRE MAIZA, Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, France
HAMZA RIHANI, Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, France
JUAN M. RIVAS, Université libre de Bruxelles, PARTS Research Centre, Belgium
JOËL GOOSSENS, Université libre de Bruxelles, PARTS Research Centre, Belgium
SEBASTIAN ALTMEYER, University of Amsterdam, Netherlands
ROBERT I. DAVIS, University of York, UK

This survey provides an overview of the scientific literature on timing verification techniques for multi-core real-time systems. It reviews the key results in the field from its origins around 2006 to the latest research published up to the end of 2018. The survey highlights the key issues involved in providing guarantees of timing correctness for multi-core systems. A detailed review is provided covering four main categories: full integration, temporal isolation, integrating interference effects into schedulability analysis, and mapping and allocation. The survey concludes with a discussion of the advantages and disadvantages of these different approaches, identifying open issues, key challenges, and possible directions for future research.

CCS Concepts: • **Computer systems organization → Real-time system architecture**; • **Software and its engineering →** *Real-time schedulability*; *Automated static analysis*;

Additional Key Words and Phrases: real-time systems, architecture, multi-core, timing analysis, schedulability analysis, WCET, co-runner interference

ACM Reference Format:
Claire Maiza, Hamza Rihani, Juan M. Rivas, Joël Goossens, Sebastian Altmeyer, and Robert I. Davis. 2019. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Comput. Surv.* 1, 1, Article 01 (February 2019), 46 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Multi-core processors offer significantly higher computing performance, as well as advantages in meeting SWaP (Size, Weight, and Power consumption) requirements compared to conventional single-core platforms. Such architectures have the potential to provide a solution to the increasing demands for low-cost, low-power consumption, high-performance real-time hardware platforms that are posed by the development of autonomous systems, including self-driving cars, Unmanned Aerial Vehicles (UAVs) or drones, and autonomous robots. While providing a number of significant advantages, the shift to multi-core hardware raises substantial challenges in relation to guaranteeing real-time performance through time determinism, time predictability, and composability.

Authors' addresses: Claire Maiza, Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, Grenoble, France, claire.maiza@univ-grenoble-alpes.fr; Hamza Rihani, Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, France, ; Juan M. Rivas, Université libre de Bruxelles, PARTS Research Centre, Belgium, ; Joël Goossens, Université libre de Bruxelles, PARTS Research Centre, Brussels, Belgium, joel.goossens@ulb.ac.be; Sebastian Altmeyer, University of Amsterdam, Amsterdam, Netherlands, altmeyer@uva.nl; Robert I. Davis, University of York, York, UK, rob.davis@cs.york.ac.uk.

0360-0300/2019/2-ART01 $15.00
https://doi.org/0000001.0000001

Real-time systems are characterised by the need to meet both functional requirements and timing constraints, typically expressed in terms of deadlines. Many design and verification techniques have been proposed which aim to guarantee that all timing constraints are met. In traditional single-core systems, timing behaviour is typically verified via a two-step process. In the first *timing analysis* step, the Worst-Case Execution Time (WCET) of each task is determined. The WCET is an upper bound on the execution time of the task running *in full isolation* on the platform, i.e. without preemption, interruption, or indeed any co-runners on other processing cores. This WCET bound is then used in the second step, *schedulability analysis*. Schedulability analysis involves considering the worst-case pattern of task execution, either via the mapping of tasks to time slots in a table-driven schedule, or via the use of a scheduling policy such as Fixed Priority Preemptive Scheduling (FPPS) or Earliest Deadline First (EDF) employed by a Real-Time Operating System (RTOS). Schedulability analysis determines the Worst-Case Response Time (WCRT) of each task in its execution context. In practice, this analysis needs to take account of preemptions by higher priority tasks and interrupt handlers, scheduling overheads caused by the operation of the RTOS, and delays in accessing shared software resources and communications. The combination of timing analysis and schedulability analysis to verify the timing correctness of a real-time system is referred to in this survey as *timing verification*[1].

The clear separation between the two steps diminishes in multi-core systems where the interference on shared resources can also depend heavily on the behaviour of *co-runners*, i.e. tasks executing in parallel on other cores. In multi-core platforms, a large set of hardware features may be shared, such as the interconnect, bus[2] or NoC (Network on Chip), the Last Level Cache (LLC), and the main memory. When a task is executed in full isolation on a multi-core platform, the behaviour of the system in terms of timing (and other non-functional properties such as power consumption) is defined by that task alone, the same as on a single-core platform. However, when two or more tasks are executed in parallel on different cores, the interplay between the tasks on shared hardware resources may lead to unforeseen and unpredictable delays [32, 123, 124]. For this reason, using the WCET of tasks executing in *isolation* on a multi-core platform without regard for co-runner interference can potentially lead to WCRT values which are *optimistic* (i.e. incorrect) by a large margin when considered in the context of the complete system. As an example, consider a cache which is shared between two or more cores. Useful cache blocks that were loaded by one task may be evicted by another task that is running in parallel on a different core. Similarly, a request to the memory bus may be blocked by an ongoing bus request from another core. We refer to all events that may prolong a task's execution due to the execution of other tasks, including additional blocking time, increased bus access times, and cache reloads, under the generic term *interference*. The potential for co-runner interference greatly exacerbates both the difficulty and the complexity of providing accurate timing verification for multi-core systems. To illustrate this, consider the simple approach of accounting for all possible co-runner interference in the timing analysis step. The difficulty here is that the execution context, with respect to co-runner behaviour, is unknown. Thus worst-case assumptions need to be made to provide a WCET that is *context independent*. This is possible, however, the results can be so pessimistic that they are not useful. For example assuming the maximum possible delay for every bus access request may result in a WCET on an m core platform that is m or more times the WCET for the task running in isolation. Thus the guaranteed real-time performance of the multi-core platform is effectively degraded to the same or worse than that of one of its cores.

[1]We note that in some of the literature the term *timing analysis* is used to describe the overall process of determining whether timing constraints are met.

[2]In this paper, as in many of the surveyed papers, we use the term *bus* or *memory bus* to mean a local interconnect from cores to the shared memory through an arbiter.

At the other extreme, attempting to account for all possible patterns and interleaving of co-runner execution in a fully integrated approach to timing verification (combining both execution and scheduling considerations into a single analysis step) quickly becomes intractable due to combinatorial explosion in the number of possible states that need to be considered. Hence, timing verification for multi-core systems typically faces the trade-off between analysing all possible interplay between tasks on different cores to provide a tightly bounded result, and making simplifying assumptions to obtain an analysis that is tractable in term of its computational complexity.

A number of approaches have been proposed to deal with this problem. Approaches based on *temporal isolation* seek to bound the delays experienced by tasks executing on one core, independent of the behaviour of the tasks on the other cores. For example using Time Division Multiple Access (TDMA) bus arbitration leads to the same delays for bus access requests irrespective of whether or not there are any tasks running on the other cores. While this is advantageous in terms of making it simpler to compute worst-case delays and interference, there is a trade-off. Mechanisms that provide temporal isolation have the disadvantage that they typically do not permit efficient resource utilisation.

Techniques that ensure temporal isolation between cores provide *timing composability*, meaning that the timing behaviour of the tasks on one core may be verified without knowledge of the behaviour of the tasks on the other cores. An alternative approach is *interference analysis*, characterising the additional delays due to contention over shared resources. These additional delays can then be incorporated into timing analysis or schedulability analysis. Examples of interference analysis in single-core systems include the derivation of cache-related preemption delays and write-back delays, which can be incorporated into schedulability analysis.

Another challenge that timing verification for multi-core systems has to tackle is the problem of *timing compositionality* [53] (note this is different from timing composability). Timing compositionality depends on the decomposition of the system into components for analysis purposes. For example, the WCET of a task might be decomposed into two components, memory access time and processing time. If the local worst-cases for each component of the decomposition can be summed to obtain an upper bound on the overall worst-case timing behaviour, then the decomposition is said to be *timing compositional*. However, in multi-core systems, this is not always the case. Timing anomalies, such as the interaction between caches and pipelines, can mean that a local worst-case behaviour e.g. a cache miss may not always lead to the overall worst-case. Further, domino effects [53] can amplify and reinforce the effects of interference. For example preemption effects on a First-In First-Out (FIFO) cache can cause additional delays that are effectively unbounded with respect to the original perturbation.

Research on timing verification for multi-core systems has received increasing attention over the past ten years, with a number of often orthogonal and sometimes incompatible approaches introduced. In this survey we provide an overview of the main approaches proposed and a review of the specific methods published in the literature.

1.1 Organisation

Research into timing verification of multi-core real-time systems can be classified into four main categories. This classification forms the basis for the four main sections of this survey. For ease of reference we have numbered these categories below to match the section numbering, starting at 2.

2 *Full Integration*: Research in this category considers full information about the behaviour of the various tasks executing on the different cores, including when they can execute and their individual resource demands. The detailed interference due to specific co-runners executing within specific time intervals is thus integrated into the analysis for every task. The timing

analyses in this category are therefore context-dependent (i.e. the results for each task depend on the set of co-runners).

3 *Temporal Isolation*: Research in this category considers either partial or full temporal isolation achieved via software means such as phased task execution, memory bus regulation, or static offline scheduling; or via hardware means such as a TDMA bus arbitration and memory partitioning. The aim being to either upper bound or eliminate the interference from co-runners that needs to be considered in the analysis. The timing analyses reviewed in this category are therefore context-independent (i.e. the results for each task are independent of the actual set of co-runners).

4 *Integrating Interference Effects into Schedulability Analysis*: Research in this category integrates the effects of co-runner interference and the use of shared hardware resources into schedulability analysis. In this case, the analysis may take into account the behaviour of the actual co-runners, or it may rely on worst-case assumptions that are valid for any co-runner. Thus both context-dependent and context-independent analyses are possible. For example both are possible when considering a bus with Round-Robin arbitration.

5 *Mapping and Scheduling*: Research in this category considers the mapping of tasks to cores and their feasibility in terms of both timing behaviour and memory requirements. Here, the processor and memory load on each core, as well as the impact of interference delays, influence the optimal placement of tasks.

Each of the main review Sections 2 to 5 starts with an overview and ends with a critique and perspectives on the research in that specific area. A few papers overlap this broad categorisation. In these cases, we provide a review in one section and a cross-reference to it in the other. Section 6 discusses the main advantages and disadvantages of techniques in the four main categories. It concludes with an identification of key open questions and important directions for future research. Supplementary material in Appendix A provides a more detailed classification of the literature according to various characteristics and properties relating to: (i) the type of timing verification performed; (ii) the timing model used for the analysis; (iii) the hardware properties considered; (iv) the software properties considered.

Note, due to space limitations, we do not provide an overview of embedded multi-core architectures and hardware components; rather we assume that the reader has a basic working knowledge of these topics (see [111] for a state-of-the-art in the context of time-predictable embedded multi-core architectures, and [115] for a discussion of more advanced Common-Off-The-Shelf (COTS) multi-cores).

It is interesting to note how research in the different categories has progressed over time. Figure 1 illustrates the number of publications in each of the four main categories in each two year interval from 2006/7 to 2016/17. (This figure is best viewed online in colour). A number of observations can be drawn from Figure 1. Firstly, the research topic is little more than a decade old. Of the 119 papers reviewed[3], the oldest papers [7, 120] that we classified were published in 2006. Since 2006, the research intensity in this area has increased rapidly, with around 60% of the papers published in the last 4 years. We can also observe trends with respect to the approach taken to timing verification. Although the number of publications aiming for a fully integrated timing and schedulability analysis (see Section 2) remained largely constant over the past 10 years, all other categories have received increasing attention. In particular, the vast majority of the papers on temporal isolation (Section 3) and on mapping and scheduling (Section 5) have been published in the last 6 years.

[3]Note not all of the bibliography is classified papers, some are related work.

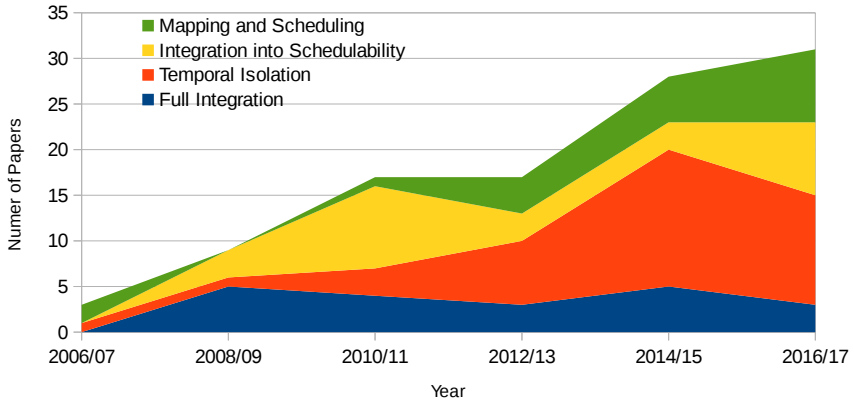

Fig. 1. Distribution of classified papers over the past 10 years

We note that the literature covers a large number of separate research threads (within which papers build on the same model and assumptions); however, these various threads of research are highly divergent with the different hardware and task models used meaning that sophisticated comparisons between different analysis results are out of reach.

1.2 Related Areas of Research and Restrictions on Scope

The scope of this survey is restricted to *timing verification techniques for multi-core and many-core platforms that specifically consider the impact of shared hardware resources*. The following areas of related research are outside of the scope of this survey: (i) works that introduce isolation mechanisms for multi-core systems, but rely on existing analysis for timing verification (examples include the work on the MERASA [94] and T-CREST projects [111], as well as work on shared cache management techniques (surveyed in [46]) and cache coherence [47, 56]; (ii) works that introduce mechanisms and analyses of the worst-case latencies for a particular component, for example predictable DDR-DRAM memory controllers[4] (comparative studies in [49, 57]), but rely on these latencies being incorporated into existing analyses for timing verification; (iii) scheduling and schedulability analyses for multiprocessor systems that consider only a simple abstract model of task execution times (surveyed in [38]); (iv) multiprocessor software resource sharing protocols; (v) timing verification techniques for many-core systems with a Network-on-Chip (NoC) (surveyed in [59, 70]) which consider only the scheduling of the NoC, or consider that tasks on each core execute out of local memory with the only interaction with packet flows being through a consideration of release jitter; (vi) measurement-based and measurement-based probabilistic timing analysis methods; (vii) research that focuses on timing verification of single-core systems. Further, the survey does not cover specific research into multi-cores with GPGPUs or re-configurable hardware.

2 FULL INTEGRATION

Research in this category aims at a fully integrated analysis utilising information about both low-level timing behaviour and high level scheduling. Thus the works discussed in this section assume almost *full knowledge* of the task under analysis and those executed in parallel, often including the exact release times of all tasks. Consequently, the results are typically only valid for the *exact scenario* which is analysed. Two approaches are dominant: WCET analyses based on *model-checking*

[4]We note that Reduced Latency DRAM (RL-DRAM) is arguably better suited for use in real-time systems than DDR-DRAM due to reduced variability in access latencies, as shown by Hassan [55] in 2018.

and analyses based on an *abstract processor state*. The worst-case path (i.e. the path with the longest execution time) is often derived via Implicit-Path Enumeration Techniques (IPET) with an Integer Linear Programming (ILP) solver. We note that all of the works assume partitioned non-preemptive scheduling. Further, the schedule is often assumed to be fixed with synchronous task releases and limited or no release jitter. Below, we subdivide the literature into analyses that focus on: (i) shared memory (i.e. cache), (i) the bus or interconnect, (iii) both the interconnect and shared memory.

2.1 Analysis of Shared Memory Only

Papers in this sub-category focus solely on the shared cache, while neglecting any interference on the common interconnect. Cache analyses either employ a complete model of the cache state, and thus use model checking to derive a cache hit/miss classification, or they employ an abstract cache state with a variant of abstract interpretation.

In 2008, Yan and Zhang [128] presented a cache analysis assuming private L1 and shared L2 caches. The analysis starts with a cache analysis assuming no interference, and then accounts for the additional L2 cache misses due to accesses from other tasks to the L2 cache. The analysis is implemented within CHRONOS and evaluated for direct-mapped instruction caches only. In 2009 Hardy et al. [54] extended this initial work to set-associative caches, multiple levels of caches, and an arbitrary number of competing cores. A further extension aims to reduce the cache interference via compiler optimisations which classify memory as reused and therefore cached, or not-reused, in which case accesses bypass the cache. Zhang and Yan [137] also presented an interference analysis for shared L2 instruction cache. They use an extended ILP and cache conflict graphs to model all possible interleaving of pairs of tasks with regard to cache accesses. These interleavings are more precisely defined than in [128], but this results in a scalability issue.

A WCET analysis based on *model checking* was presented by Gustavsson et al. [51] in 2010. The modelled architecture again features private L1, shared L2 caches, and a set of tasks, but no explicitly modelled interconnect. Instead, the authors assume dedicated memory ports per core. The core architecture is assumed to be trivial with fixed execution times per instruction and without any timing anomalies. UPPAAL is used as the model checker and despite the strong simplifications, the state-space explosion limits the applicability of the approach, even without modelling the program semantics.

Also in 2010, Lv et al. [82] combined abstract interpretation and model checking to provide a fully integrated analysis. Abstract interpretation is used to give a basic block timing model. Timed automata are then used to model the whole program and the shared bus. The authors consider a multi-core architecture with a local L1 cache and no code sharing between concurrent programs. Each L1 cache miss is considered as a shared bus access. The analysed shared bus arbiters are First-Come First-Served (FCFS) and TDMA, with the approach scaling better in the case of TDMA due to the timing isolation provided. UPPAAL is the model checker used in the experiments.

In 2016, Nagar et al. [86] presented a shared cache analysis with private L1 and shared L2 caches. The analysis determines the global worst-case arrival pattern for all accesses to the shared cache including the worst-case execution path. The analysis is therefore independent from the precise task release times, but still requires precise knowledge about all tasks executing on the other cores. The authors show that finding the worst-case interference pattern is NP-hard and thus as well as the exact analysis an approximation with linear complexity in the cache size is proposed. The analysis is implemented within the CHRONOS timing analyser.

2.2 Analysis of the Interconnect Only

Papers in this sub-category focus solely on the common interconnect. Interference due to shared memory is either ignored or full isolation (spatial via cache partitioning, or temporal via TDMA) is

assumed. In all papers, the delay due to the interconnect is precisely estimated and integrated into the worst-case execution time.

In 2010, Andersson et al. [8] presented an analytical method to calculate the WCET of tasks in a multi-core including contention on the shared bus. They assume no inter-task evictions (i.e. the shared cache is partitioned) and non-preemptive scheduling. Further, the memory arbitration is assumed to be work-conserving. The analysis formulation is based on an upper bound on the maximum number of bus requests from the tasks.

In 2015, Rihani et al. [103] used SAT Modulo Theory (SMT) to encode the whole execution of a task on a platform with TDMA bus arbitration. The novelty here is the encoding of the program semantics to gain precision in the WCET bound, taking infeasible execution paths into account. The bus access is modelled with an offset. The SMT-solver provides an execution path with correct semantics and WCET analysis.

In 2015, Jacobs et al. [62] proposed deriving WCET bounds for multi-core systems by integrating the bus delay into the pipeline analysis. The authors assume a shared bus policy with bounded blocking time (e.g. Round-Robin or TDMA). For a memory access over the shared bus, the entire range of possible blocking delays is considered, i.e. direct access to the bus, blocked for 1 cycle, 2 cycles, etc. In the so-called co-runner insensitive analysis, this scheme is applied to all bus accesses made by the analysed task, and assuming the entire range of blocking cycles. In the co-runner sensitive analysis, information about potential accesses from other task is considered to reduce the over-approximation. This assumes precise knowledge about when memory accesses of co-running tasks may occur.

Subsequently, in 2016, Jacobs et al. [63] presented a framework for the derivation of WCET analyses for multi-core systems. They point out that computing the WCET taking into account all possible inter-leavings of resource accesses is computational intractable. They make the key point that the vast majority of work assumes timing compositionality and hence first compute a basic bound without interference effects and then add an upper bound on direct interference effects. However, for real architectures that exhibit timing anomalies and domino effects, this is not sufficient. The key idea in the paper is to lift system properties to an abstract level, so that they can be applied to the abstract model of traces to identify sets of infeasible traces, thus making the analysis less pessimistic. The framework is instantiated and used to provide bounds for the Malardalen benchmarks for 2, 4, and 8-core platforms, with either local scratchpad or cache, and in-order or out-of-order execution.

Research that overlaps with this sub-category but is reviewed elsewhere in this survey includes the work of Dasari et al. [35, 36] and Kelter et al. [66–68] (see subsection 3.4).

2.3 Analysis of Shared Memory and Interconnect

Paper in this sub-category consider both the common interconnect and shared memory.

In 2008, Andrei et al. [10] presented a WCET analysis technique in conjunction with an algorithm to build a static (offline) schedule for the tasks and the TDMA bus in multi-core systems. The authors target a platform model where the cores have a private cache, and then through the shared bus can access another private memory, and a global shared memory. Only the private memories are cached. Inter-core communications are handled via the global shared memory. Additional tasks are added to handle the copy and writes to the global shared memory (i.e. explicit communications). A task can also produce implicit communications when a cache miss occurs, and the private memory must be fetched (with an access to the bus). The proposed technique iteratively constructs the schedule of tasks and incrementally defines the TDMA slot configuration, taking into account task execution time inflation due to bus accesses.

An approach for worst-case response time analysis of a multi-core system with private L1 cache and shared L2 cache was developed by Li et al. [78] in 2009. The aim is to compute the worst-case L2 cache interference between tasks running on the different cores accounting for their lifetimes, i.e. the time intervals in which the tasks can be active. The model used is a synchronous one, with a Message Sequence Chart describing the interaction and precedence relations between tasks. Tasks are assumed to be partitioned to cores and run non-preemptively. The WCET includes the effects of interference on the L2 cache. An iterative process is used to calculate WCRTs, task lifetimes, and the amount of interference from overlapping execution due to tasks on the other cores.

In 2012, Dasari et al. [34] presented a method to estimate tight WCETs of tasks running on a multi-core system in which the bus arbitration is Round-Robin, and tasks are scheduled non preemptively under an unspecified non-work conserving policy. The work is divided into two parts. The first part presents an algorithm to compute an upper bound of the number of memory request by a core in any time window, considering the interference from every task at the same time (instead of analysing tasks independently and then adding their contributions). The actual schedule of jobs is unknown which adds pessimism to the computed worst-case ordering of jobs. A second contribution of the paper is to use the provided upper bound on the number of requests to enhance the classical WCET formula for systems with Round-Robin buses.

A full analysis from WCET to mapping with interference analysis was presented by Boniol et al. [17] in 2012. This work focuses on the implementation of synchronous applications on a multi-core platform. The architecture consists of cores with local L1 and L2 cache, a shared main memory with shared bus and DRAM controller. The timing analysis consists of a core and memory mapping/scheduling with slicing of the timing window. It takes into account the interference on the shared bus and memory controller. The interference analysis is based on timed automata. This paper also takes a holistic view on WCET and schedulability analysis, but differs from all other papers in this category in that it focuses on the schedulability analysis directly.

In two papers in 2012 [24] and 2014 [23], Chattopadhyay et al. presented a WCET analysis that includes interference on a shared TDMA-based Round-Robin bus and a shared L2 cache. The global analysis is based on an ILP formulation (execution graph) of each delay from the pipeline to the bus and cache interferences. The delays are expressed as an interval and combined through solving the ILP. The bus analysis uses an offset model introduced by Chattopadhyay et al. [26] together with the cache model. The earlier work [24] assumes a timing-anomaly free architecture, whereas the later one [23] supports architectures with timing anomalies and more sophisticated cache architectures.

In 2014, Kelter et al. [69] analysed the WCET of *parallel* tasks. They estimate the execution time with a known synchronisation of the task releases. From this release cycle, an execution graph is derived at a cycle level and the access to shared resource is evaluated depending on which task is requesting access at that time. It is an IPET based analysis (ILP system of constraints). There are three arbitration schemes modelled as part of the bus analysis: TDMA, Round-Robin and priority based. The platform under analysis is a multi-core based on an Arm7 pipeline with local cache or scratchpad, a shared bus to access shared memory. The authors mention that analysis of 8 cores was not possible due to scalability issues.

In 2016, Perret et al. [101] studied various sources of interference on COTS many-core platforms (similar to the Kalray MPPA-256). A realistic analysis is presented regarding the sources of interference between applications on their memory access path. The analysis focuses on local Static Random Access Memory (SRAM) banked memory, NoC, and global Double Data Rate Synchronous DRAM (DDR-SDRAM). The authors show and quantify the potential pessimism implied by a context-independent "worst-case-everywhere" calculation (i.e. with no assumptions about the competitors for each shared resource). They then propose recommendations for building viable

execution models (complying to certain rules) in order to ease tight timing analysis (an extended version of these recommendations are presented by the same authors in [102]).

2.4 Critique and Perspectives

Only 20 papers fall into the category of full integration analysis, and interestingly, these papers were nearly all written by researchers from groups that focus on static timing analysis and have developed their own tools (e.g. OTAWA, Heptane, CHRONOS, aiT, WCC) in the past. The scheduling aspect on multi-core systems is largely ignored by these papers and instead the task schedule is assumed to be known, fixed, and non-preemptive. How the schedule can be determined without initially knowing the WCETs is not explained in any of these works. This leads to an implicit restriction: the exact schedule including the release time at which tasks start executing must be known or at least approximated a priori. Many approaches in this category make an even stronger assumption in that they assume precise release times [23, 51, 63, 69, 128]. Any variation in release times may therefore invalidate the results. Consequently, these approaches can only be used if either: (i) the operating system enforces the scenarios as analysed, or (ii) the analysis is repeated for all potential execution scenarios. In the first case, the system utilisation may be heavily reduced and it may even be infeasible to enforce the precise scenarios analysed, especially on large multi-core system with asynchronous clocks. In the second case, the number of scenarios that must be analysed is prohibitively large, even for small task sets. This problem adds to the fundamental drawback of these approaches: The fully integrated approaches suffer from high complexity and thus low scalability. This is particularly the case for approaches based on model-checking, but also affects those based on implicit path enumeration. The problem exists considering strongly simplified processor models that only model memory or the interconnect; modelling both types of components, let alone realistic multicore architectures further exacerbates the computational complexity. It is debatable whether any of these approaches can be used to analyse realistically sized task sets on non-trivial (i.e. realistic) multi-core architectures.

The need for full system knowledge also forecloses, or at least greatly complicates, common and desirable design practices such as integration of 3rd party software and incremental verification.

3 TEMPORAL ISOLATION

Research in this category simplifies the problem of multi-core timing verification by shaping access to the main sources of inter-core interference, mainly the shared interconnect or memory bus. The basic idea is that by controlling access to the memory bus, contention can be avoided or bounded, and thus becomes more predictable and easier to integrate into a timing analysis formulation. The result is analyses that are not only easier to define, but are also less computationally demanding and thus more scalable, particularly when compared to the full integration approaches reviewed in the previous section. The research work in this area identifies the *memory bus as the main source of inter-core interference* that must be controlled, and relies on additional complementary techniques to manage shared cache and shared DRAM subsystems. Temporal isolation is sometimes combined with spatial isolation to improve resource sharing and utilisation while reducing the interference. Below, we subdivide the literature depending on how the access to the memory bus is controlled via: (i) phased execution model, (ii) memory bandwidth regulators, (iii) offline scheduling, (iv) hardware isolation, for example a TDMA bus.

3.1 Phased Execution Model

Research papers in this sub-category re-factor the tasks' code to create a new equivalent task with two types of disjoint phases: memory bound phases and computation bound phases. Each task typically starts by executing a memory phase in which all the data and instructions needed by the

task are read from main memory and stored in local private memory (e.g. cache or scratchpad). The subsequent computation phase can then perform the actual computation of the task using data and instructions that are readily available in the private memory, *without any need to access the interconnect or shared main memory.* Optionally, some models add a further memory phase at the end of each task to write back the updated data into the main memory. The phased execution of tasks is typically used in conjunction with a scheduling algorithm that exploits the fact that access to the shared main memory and bus are only going to occur during the memory phases of the tasks. Therefore the scheduler can avoid contention in the main memory by not overlapping memory phases in different cores. Such a schedule can be achieved for example by defining a global scheduler with a queue just for memory phases.

An initial relevant implementation of the phased execution concept can be found in the work of Pellizzoni et al. [97] in 2008. This paper presents a framework to schedule tasks taking into account contention in the shared memory induced by concurrent master I/O peripheral Direct Memory Access (DMA) transfers. The basic idea is to control when the peripherals can operate using a special hardware called a *peripheral gate.* A *co-scheduling* algorithm is proposed that tracks actual execution times of the tasks to determine their slack (difference to their execution budgets). These dynamic slack times are used to open the peripheral gates, making sure that the slack times are always non negative. Although the paper targets single core systems, its concepts can be applied to multi-core systems.

In 2011, Pellizzoni et al. [96] expanded the above ideas to define the PRedictable Execution Model (PREM). Under PREM, periodic tasks are decomposed into so-called predictable intervals and compatible intervals. Compatible intervals are reserved for those sections of code that cannot be re-factored into memory/computation phases, mainly due to the inability at compile time to determine which memory regions are accessed. From a scheduling stand-point, compatible intervals are treated as memory phases, since they can incur accesses to the bus. Experiments on a prototype test-bed validate the theory. In particular, execution is highly predictable and the WCET of memory bound tasks is significantly reduced.

Later works extend the ideas of PREM. In 2012 Yao et al. [129] proposed Memory Centric scheduling where *non-preemptive* memory phases are executed in defined time slots using a high level coarse grained TDMA schedule. The main idea behind the Memory Centric approach is to promote the priority of tasks that are in memory phases above those that are in execution phases. Fixed priority partitioned scheduling of tasks is assumed, and is extended to include priority promotion for memory phases. Cache partitioning is assumed to avoid any intra-core evictions. The TDMA memory scheduling provides isolation and thus single core equivalence. Hence the response time analysis provided only needs to consider the tasks executing on the same core as the task under analysis. The approach is effective when tasks have a high memory demand, but not when the memory demand of the tasks is low.

A comparison of different algorithms to schedule tasks that follow PREM was presented by Bak et al. [14] in 2012. The comparison is based on simulations performed on synthetic task sets, with different combinations of TDMA and priority based schedulers for the memory and computation phases. The authors conclude from the simulated scenarios that the best policy is to schedule the computation phases with EDF, and the memory phases in a least-laxity-first non-preemptive manner. The simulations confirm the benefits of the priority promotion of the memory phases over the computation phases, as was also shown analytically by Yao et al. [129].

In 2014, Alhammad et al. [4] proposed an offline static scheduling approach leveraging PREM. This work assumes a multi-core system with private LLC, or a partitioned shared cache. A static schedule of the memory and computation phases is constructed such that concurrent accesses to the shared memory are avoided. This eliminates delays due to concurrent accesses reducing

WCETs. Later in 2014 Alhammad et al. [3] extended PREM to global non-preemptive scheduling (called gPREM). Under gPREM, a task can only run when no other task is executing its memory phase and at least one processor is idle. This scheduling algorithm ensures that only one memory phase is executed system-wide, and contrary to the memory-centric approach of Yao et al. [129], no priority promotion of memory phases is implemented.

Also in 2014, Wasly et al. [125] proposed a mechanism to partially hide the overheads induced by the memory phases by splitting the local double-ported memory into two partitions. The scheduling algorithm then ensures that, while a task is executing out of one local partition, the code and data for the next task to execute is placed in the *other* partition. A variation on fixed priority non-preeemptive scheduling is assumed. The policy divides time into intervals, with scheduling decisions only made at the start of an interval. At the start of an interval, the processor executes the ready task that was loaded into one of the partitions before the end of the previous interval. The interval ends once both task execution and DMA operations have completed. An upper bound worst-case response time analysis is presented for the scheduling policy. The evaluation is performed using a custom scratchpad controller that implements the required latency-hiding capabilities. The paper assumes isolation at the bus level (e.g. TDMA arbitration in the memory bus). An extension of this work for global scheduling, that does not assume isolation through TDMA was presented by Alhammad et al. [5] in 2015. In 2016, Tabish et al. [122] also reused this work and combined it with another set of isolation mechanisms (core private scratchpads, dedicated DMA channel using a TDMA schedule) to obtain predictable execution of an OS on a multi-core platform.

In 2014, Kim et al. [74] presented a method to construct offline (static) schedules of Integrated Modular Avionics (IMA) partitions for a multi-core system, with the objective of avoiding conflicts in I//O transactions. Each application is divided into three sequential non-preemptive phases: (i) device input, in which the input data of the device is copied into the main memory, (ii) processing of the data, and (iii) device output, in which the generated data is copied from the main memory to the output device. Each of these steps is executed as an IMA partition, and the I/O conflicts are avoided by executing the I/O partitions in a dedicated I/O core, which avoids overlapping I/O partitions. The paper presents a scalable heuristic to calculate the static schedule to meet the deadline and precedence constraints of the partitions. A previous work was presented in 2013 by the same authors [75] in which the I/O phases (called solo partitions in this work) can be mapped to any core, so the concept of defining an I/O core is not used. The authors find this generalized problem to be NP-complete, and propose two solutions. The first is solved with Constraint Programming, and has the restriction that partitions mapped to the same core in the initial single-core system must be mapped to the same core in the final multi-core system. The second solution relaxes this restriction, but restricts the I/O phases to have a unitary length.

In 2015 Melani et al. [85] analytically characterised the improvements that can be achieved by a phased execution scheme for global fixed task priority scheduling. The paper proposes the first *exact* schedulability test for this model, and concludes that EDF is not optimal in this case. This work also provides and proves correct the critical instant for phased execution and defines an exact worst-case response time analysis for fixed priority scheduling.

The Memory Centric scheduling approach [129] was extended to global scheduling in 2016 by Yao et al. [130] without relying on TDMA to schedule the memory phases. This work uses the concepts of virtual memory cores, and virtual execution cores. The idea is to limit the number of cores that can access memory at the same time, ensuring that the memory bandwidth is not saturated and the interference due to memory contention remains very small. A global fixed priority scheduler is used. The priority of memory phases is promoted over that of execution phases (of all tasks). Only the m_{memo} highest priority ready tasks in memory phases are permitted to execute at any given time. Other ready tasks in memory phases are blocked. Schedulability analysis is provided,

which effectively assumes that the memory phases run on m_{memo} cores and the execution phases on the remaining cores. Evaluation shows that the approach outperforms a baseline (considering contention) when the slowdown of the latter due to memory contention is large.

In 2018, Pagetti et al. [93] proposed a compilation framework that extends the WCET-aware compiler (WCC) to also generate a mapping and a partitioned non-preemptive schedule. The execution model considered aims at eliminating interference by splitting tasks into sub-tasks representing data acquisition, execution, and replication phases. Moreover, the platform considered has a TDMA bus. Considering constraints from applications and platform descriptions, the framework uses an ILP solver to generate an off-line schedule of tasks mapped to each core.

3.2 Memory Bandwidth Regulators

Research papers in this sub-category consider systems where access to the memory bus is restricted. Since memory bus accesses (e.g. LLC misses) can occur at any point in task execution, control over access to the memory bus is achieved by implementing a *memory bandwidth regulator*, which operates in a similar way to a periodic server: each core is assigned a portion of the total memory bus bandwidth in terms of a budget, which is replenished periodically. When a core exhausts it memory bus bandwidth, then it is idled until the start of the next period. The effect of implementing a memory bandwidth regulator is that the interference from other competing cores can be easily bounded. Additionally, by assigning budgets that avoid overloading the memory bus, each task runs with a guaranteed memory bandwidth that effectively ensures that its execution is *independent* of the behaviour of tasks running on the other cores. Thus timing analysis for single-core systems can be adapted to the case of multi-core systems with memory bandwidth regulation.

In 2012, Yun et al. [135] (reviewed in Section 4.1) proposed memory regulation similar to the behaviour of a periodic server. Subsequently, in 2013, they introduced MemGuard [136], which provides synchronous (same period) regulation of all cores via performance counter overflow interrupts.

In 2013, Behnam et al. [16] presented the Multi-Resource Server (MRS). These servers regulate both the processing time and memory bandwidth. When any budget in the server is depleted, the server is stalled until its next replenishment period. The proposed analysis is based on a compositional framework. It performs a local analysis of the tasks within a server, and also validates the set of servers globally. The analysis has the important simplifying assumption that memory access delays are considered *constant*. This limitation was later lifted by Inam et al. [61] in 2015. Here, a global upper bound for DRAM latencies with First-Ready FCFS (FR-FCFS) arbitration is calculated and used, with different delays for private and shared memory banks taken into account. The analysis considers the worst case by assuming that tasks deplete the memory budget by issuing sequential memory requests at the beginning of the server period.

In 2014, Nowotsch et al. [90] introduced an interference-aware WCET analysis, coupled with run-time monitoring and enforcement. The schedule is assumed to be offline (static) and time-triggered (ARINC 653 standard). The WCET is composed of two parts: the core-local execution time assuming privileged access to shared resources, and an upper bound on the shared resource interference delay. This upper bound is guaranteed by the Operating System, which monitors resource usage and prevents execution if the resource budget is going to be exceeded. This way a limited form of temporal isolation is guaranteed. The authors describe under what circumstances a simple summation of the interference delay and core-local execution time results in a safe upper bound. Nowotsch et al. [89] extended their prior work [90][5] by implementing a budget (i.e. capacity) re-claiming method. When a task exhausts its assigned capacity, the capacity usage of all the tasks

[5]Note that although [90] was published after [89] it contains earlier work and is cited in [89].

is re-evaluated to determine if the suspended task can be granted some additional capacity to resume execution. As with the budget reclamation mechanisms implemented in Memguard [136], this approach aims at improving the average case while still providing a worst-case guarantee.

In 2015, Yao et al. [131] presented a WCET and schedulability analysis for multi-core systems with memory bandwidth regulators. Here, each core has a memory bandwidth budget (i.e. memory access time allowed per period) and a replenishment period. Cache is considered private or partitioned for each core. This work assumes a known WCET for each task running alone, and that this WCET can be divided into computation and memory access times. The analyses presented are based on the calculation of the stall time for the task under analysis, which can result from bandwidth regulation or from contention from other cores. One memory access can suffer interference at most once from each core. The stall time of the task under analysis is thus shown to be independent of the characteristics of the other tasks, thus achieving isolation via software means. For the WCET analysis proposed (which considers a task running alone in a core, but suffering interference from other cores), the authors show that tighter results can be obtained if the memory bandwidth replenishment periods of the cores are synchronised. The schedulability analysis is constructed by creating a synthetic task that includes the effects of all task instances in the relevant response-time analysis window. As a consequence, the WCET analysis can be applied to this synthetic task to obtain schedulability results. We note that this work considers a single constant memory access time. Subsequent work by Awan et al. [12] in 2018 extended the analysis from one to two memory controllers.

In 2015, Mancuso et al. [83] presented the Single Core Equivalence (SCE) framework which provides a means of *isolation* effectively exporting a set of equivalent single-core machines to the application. SCE methods are applied to the LLC, DRAM memory bandwidth, and shared DRAM banks. Profiling is used to determine the pages of each task's code that have most impact on the WCET and these pages are locked in the cache, thus avoiding any cross-core contention. The *Memguard* [136] mechanism is used to regulate the memory bandwidth allocation for each core in a given period. Thus once the capacity is reached in terms of the number of accesses, the core is idled until the next server period. Finally, the *PALLOC* tool [133] is used to assign disjoint sets of DRAM banks to applications running on different cores to exploit parallel accesses. The additional latency due to memory regulation is integrated into response time analysis assuming that memory bandwidth is equally distributed among the cores. Note, in contrast to the work of Yao et al. [131], this work considers minimum and maximum memory access latencies.

Also in 2016, Pellizzoni et al. [99] presented a schedulability analysis for memory bandwidth regulated systems with FR-FCFS memory arbitration. The analysis exploits knowledge of the restrictions on memory requests from competing cores to produce lower delay bounds. First, it performs a core-local analysis to determine the bandwidth requirements that a core needs to schedule its tasks. These requirements are called the schedulability envelope, and are defined as the memory budget required by the core, and the maximum memory budget that can be assigned to other cores. Second, a system analysis is performed to determine if the demands of all cores can be satisfied. The benefit is that each core can be developed independently, with no knowledge of the tasks running on the other cores. The authors define separate analyses for systems with read only regulation, and for systems with read/write regulation.

In 2017 Mancuso et al. [84] extended the SCE approach based on Memguard [136] by allowing different memory bandwidth budgets for the different cores. This work shows how to derive task WCETs as a function of the assigned memory budget, the execution time (not including memory requests) and the number of memory requests. Schedulability analysis is then given which utilises these WCET functions. The evaluation shows around 30% improvement in WCET compared to assigning equal budgets to all cores.

In 2017, Agrawal et al. [1] tackled the problem of scheduling ARINC 653 partitions in a multi-core system taking into account memory contention. The computation and memory access budgets are given at the slot level instead of the partition level. Memory budgets for each slot are selected from a list of pre-defined budgets that depend on how many cores are active in the slot (when more cores are active then each core is given a lower memory budget). The offline (static) schedule and slot budget assignments are performed using an upper bound on the number of memory accesses for each partition and a WCET for execution in isolation. The worst-case memory access pattern is constructed by moving the memory accesses to the slots with lower memory budgets.

In 2018, Agrawal et al. [2] considered the case where the core memory bandwidth budgets can change arbitrarily over time. The analysis is constructed as an optimisation problem that aims to find the worst-case memory request distribution for each budget configuration. The evaluation is performed on an ARINC 653 type system. It shows that dynamic budget assignments dominate static budget assignments for the task sets studied. A simple heuristic is also provided to calculate dynamic and static budget distributions.

In 2018, Awan et al. [13] extended the SCE approach considering uneven memory regulation (assuming upper and lower bounds on the access time of a single memory transaction). The idea being to provide inter-server isolation even for servers on the same core. The budget assignment is obtained by solving an ILP system of constraints. The results show an improvement in terms of schedulability over previous approaches.

In 2018, Freitag et al. [41] presented an isolation technique using performance counters. The isolation proposed here differs from other works on memory bandwidth regulation in that one core is given priority over all others. Real-time guarantees can therefore only be provided for this dedicated real-time core, whereas all other cores are effectively executing tasks on a best-effort basis. Performance counters are used to track the progress of the real-time application on the dedicated core. As soon as the performance falls behind the statically predicted necessary progress to finish on time, other cores are either slowed down, or halted entirely to reduce the memory contention.

3.3 Offline Scheduling

Research papers in this sub-category use offline static scheduling techniques whereby a table is generated that contains all the scheduling decisions for use at run-time such that concurrent accesses to shared resources (e.g. the memory bus) are controlled.

In a series of papers [43–45] from 2013 to 2016, Giannopoulou et al. build on top of non-preemptive Flexible Time-Triggered Scheduling (FTTS) with dynamic barriers to achieve partial isolation in a mixed-criticality system. The initial work [43] in 2013 considers offline scheduling and task mapping to cores, accounting for processing time and synchronous memory accesses, while ensuring coordination among cores such that tasks of different criticality levels are not executed at the same time. Simulated annealing algorithms are proposed for the task mapping and construction of the schedule. Worst-case response time analysis accounting for synchronous memory accesses is done via the Timed Automata and model checking methods described in [42]. Later work [44] in 2014 further develops the system design optimisation under FTTS for memory bank sharing. A simulated annealing framework is used to map tasks to cores, and data and communication buffers to shared memory banks. Finally a further generalisation in 2016 [45] considers scheduling and task/memory mapping for a *clustered many-core* taking into account delays due to memory banks local to the cluster and due to remote data copied from external memory. Task dependencies are supported, and tasks are also assumed to have a minimum degraded execution requirement that must be satisfied in all system criticality levels. Network calculus is used to bound the delays over the NoC, while

task mapping to cores and shared memory mapping to local memory banks is optimised using a simulated annealing algorithm.

In 2016, Perret et al. [102] introduced an execution model that restricts the application behaviour to ensure temporal and spatial isolation between applications and to bound the WCET. The execution model consists of a set of rules that include spatial partitioning of the tasks, time-driven access to the NoC, and rules to share the DDRx-SDRAM banks.

In 2015, Carle et al. [19] proposed an automatic procedure for the design and implementation of data-flow programs on multiprocessors. Here, a task allocation algorithm is used to optimise the end-to-end response time. Temporal isolation is enforced between tasks allocated to different time frames, while a conservative upper-bound on the WCETs accounts for all delays due to accesses to shared resources by tasks in the same time frame.

In 2018, Carle and Casse [18] presented a method to derive a static schedule of unmodified binaries that minimise the execution time inflation due to inter-core interference. The standard IPET WCET technique is used to identify which instructions in the binary code are not guaranteed to always be hits and thus could suffer from interference. These instructions are called TIPs (Time Interest Points). Tasks are represented by a sequence of intervals, each with a constant execution time, separated by TIPs. A partial isolation approach is proposed. This involves building a static schedule using ILP that minimises interference, considering a constant cost per interference. The authors conclude that with this approach ILP is only applicable for relatively short tasks.

Research that makes use of offline (static) scheduling for the purposes of temporal isolation but is reviewed elsewhere in this survey includes the work of Alhammad et al. [4] (see Section 3.1) and Becker et al. [15] (see Section 5.1).

3.4 Hardware Isolation

Research papers in this sub-category focus on WCET analysis in the context of hardware isolation mechanisms where interference either does not depend on the co-runners, or can be bounded independent of the behaviour of the co-runners. A TDMA or Round-Robin bus arbiter is the most popular way to ensure such hardware isolation, other approaches use memory partitioning / optimisation.

In 2007, Rosen et al. [105] proposed an approach that integrates system level static cyclic scheduling and WCET analysis. The main focus is on optimising the periodic schedule table of TDMA slots that give each core access to the bus, with the aim of minimising the latest termination time of a set of tasks represented by a task graph. A system level offline (static) schedule of tasks (executing on all cores) is constructed using a list scheduling approach. At each step, corresponding to the start of a task, the next segment of the bus schedule (up to the next completion of a task) is optimised according to a cost function which considers the WCETs of the currently active tasks and an estimate based on the conflict-free WCET of their successors. (Note the WCETs of the active tasks are dependent on the bus schedule and are recomputed for the different bus schedules examined as part of this optimisation step). Each WCET computation uses an ILP system that encodes accesses and the availability of the bus.

In a paper in 2011 [66] and the extension to it in 2014 [67], Kelter et al. presented WCET analysis in the context of multi-core systems with local instruction and data scratchpads connected via a shared TDMA bus to a global shared memory. The analysis takes the precise time offset between task execution and TDMA slots into account to bound the delay each memory access may suffer while waiting for the next TDMA slot. The analysis operates at the level of basic-blocks and formulates an offset graph that captures the dependencies between different memory accesses (i.e. the impact that delays to an earlier access can have on the timing of subsequent accesses). The offset graph is

used to encode the timing dependencies into an ILP system of constraints, and thus determine the WCET.

In 2011, Yoon et al. [132] introduced a WCET estimation depending on cache partitioning and bus configuration. The solving of a Mixed Integer Linear Programming (MILP) system of constraints leads to a WCET estimation and cache and bus configuration. The WCET estimation is separated into a local WCET estimation that does not depend on other cores or tasks, and an interference delay. For the bus, a configurable Round-Robin scheme is used where each core has an upper bound on the delay and different "rounds" are defined. For the cache, the notion of banks and columns are used to partition.

In 2013, Kelter et al. [68] derived formulae for the bounds on the worst-case bus access delay for four bus arbitration policies: Round-Robin, TDMA, priority division, and priority-based arbitration. They also compared the worst-case and average-case performance of these schemes. Building on this earlier work, in 2014 Kelter et al. [65] investigated two techniques to reduce WCETs in partitioned multi-core systems. First, they use a genetic algorithm to optimise the bus schedule (i.e. the number of slots and their length, assigned to each core) for a set of bus arbitration policies, including Round-Robin, TDMA, and priority-based. Second, they explore reordering instructions within basic blocks to better match the bus schedule. (The WCET bounds for each bus arbitration policy are estimated using the formulae from [68]). Overall, an improvement of up to 33% is observed. This technique also allows a trade-off between average-case and worst-case execution time.

A framework for memory contention analysis was introduced by Dasari et al. [35] in 2015. The focus of this work is on the shared resource access delay, with an upper bound on the number of accesses assumed to be provided for each task. (In the paper, this upper-bound is obtained from measurements, with regions of task execution used for a more fine-grained analysis). The shared bus arbiter is modelled by considering the earliest and latest available communication slots for the task under analysis. Mathematical models of the most widely used bus arbiters are provided. This work was extended by Dasari et al. [36] in 2016, considering cache partitioning to provide better isolation and hence obtain more precise timing bounds. The aim of the work is then to compute the WCETs of the tasks (executing concurrently) assuming specific bus policies, both work-conserving and non-work-conserving (e.g. TDMA), and fixed priority partitioned scheduling.

In 2016, Valsan et al. [123, 124] showed that partitioning the shared cache in COTS multi-core processors that use non-blocking caches is not enough to ensure predicable execution time behaviour. (Non-blocking caches allow hits to still be served from the cache, while waiting for a miss to be served from main memory). The authors showed for three different COTS multi-core architectures that even when the global (LLC) cache is partitioned between cores, co-runners can cause up to a 21-fold increase in execution times compared with running alone. This is due to contention over Miss Status Holding Registers (MSHR). A combined hardware and software solution is presented which controls the memory level parallelism of each core and eliminates contention over MSHRs.

In 2017 Oehlert et al. [92] proposed an ILP formulation for WCET estimation and a Genetic Algorithm for selecting code to place in local scratchpad memory so as to reduce the WCET. This work is in the context of TDMA bus arbitration. In particular, the stall delays due to the TDMA cycle influence the optimal scratchpad allocation. The bus-aware approach that the authors propose is evaluated against a similar bus-unaware approach. For a dual-core, the new ILP method gives 23% lower WCET estimates on average, while the genetic algorithm resulted in higher WCETs. These results were more pronounced for more cores dues to the longer bus stall times.

Research that overlaps with this sub-category but is reviewed elsewhere in this survey includes the work of Lv et al. [82] (see Section 2.1), Rihani et al. [103] (see Section 2.2), Andrei et al. [10] (see

Section 2.3), Suhendra et al. [121] and Chattopadhyay et al. [25] (see Section 4.2), and Schranzhofer et al. [112] (see Section 4.1).

3.5 Critique and Perspectives

Research in this category relies on mechanisms that shape access to shared hardware resources, particularly the interconnect/memory bus, reducing or preferably eliminating contention. This shaping makes the system more predictable and greatly simplifies the analysis needed for timing verification. There are however a number of trade offs.

Research in the first sub-category makes use of the *phased execution model* which only allows a limited number of memory phases at the same time, usually only one, thus providing temporal isolation. Here the trade-offs include: (i) the code overheads added by the memory phases, (ii) the delays until the scheduler grants each memory phase access to the bus.

Research in the second sub-category makes use of *memory bandwidth regulators*. The main trade-off of this approach is that regulation effectively *slows down* task execution. With each task running as if the memory bus was always just below its saturation point, even when other cores are idle. This leads, in many cases, to not fully utilising the computing resources of the platform.

Research in the third sub-category makes use of *offline scheduling* to shape accesses and hence provide temporal isolation. As with single core systems, a lack of flexibility is the main trade-off for the high degree of predictability afforded by offline schedules. The system configuration may need to be completely re-computed for every change in the tasks.

Research in the fourth sub-category makes use of *hardware isolation* to shape accesses. The difficulty here is that hardware isolation techniques necessarily require specific hardware designs to be employed. Often hardware isolation, for example via a TDMA bus, comes at a cost in terms of using non-work conserving protocols that compromise the effective bandwidth that can be used. As a result such techniques are rarely employed in COTS multi-core systems. Partial isolation using arbitration policies such as Round-Robin means that interference can be bounded independent of the co-runners; however, typically the analysis is more pessimistic since a worst-case delay is assumed on every access.

In conclusion, the research described in this section shows that while techniques based on the concept of temporal isolation can lead to more predictable access to shared hardware resources and consequently simpler analysis for timing verification, the main trade-off is in not fully taking advantage of the system computing resources (i.e compromising throughput and average case performance). An open question is whether temporal isolation techniques and their analyses can be improved to allow better utilisation of the available hardware resources. The work of Hebbache et al. [58] in 2018 on dynamic bus arbitration schemes based on TDMA is an example of research in this direction. The approach proposed provides utilisation similar to that of work-conserving policies, while preserving the guarantees of TDMA in the worst-case.

4 INTEGRATING INTERFERENCE EFFECTS INTO SCHEDULABILITY ANALYSIS

Research in this category aims to account for additional delays that impact task execution due to the arbitration policies used to access shared resources and the contention caused by co-running tasks on other cores. It does so by integrating these delays into *schedulability analysis*. Typically, the WCET for each task running in isolation (i.e. without contention for resources) is used as a baseline, with the effects of contention included as additional terms in the *response time analysis* or other schedulability test formulation. Many works consider a single shared resource such as the interconnect or memory bus, shared cache, or DRAM, while other papers integrate the effects of contention across multiple shared resources. Below we subdivide the literature depending on the

interference considered: (i) interference contending for the memory bus, (ii) interference contending for memory (cache, scratchpads, or DRAM), (iii) interference contending for multiple resources.

4.1 Interference Contending for the Memory Bus

Research papers in this sub-category focus mainly on analysing contention on the memory bus and how the effects of this can be integrated into schedulability analysis.

In a series of papers [107–110] from 2008 to 2011, Schliecker et al. considered the effect of resource accesses delays, due to co-runner contention, on the response times of tasks. The state-of-the-art for this thread of research is captured in the final paper [107], which generalises the approach to arbitrary shared resources with work-conserving arbitration policies. Schliecker et al. [107] use arrival curves to model the number of resource accesses from each task, using a minimum separation between accesses and a maximum number of accesses per job. This is extended to multiple tasks running on a single core. For each task, its WCRT is computed based on the task's baseline WCET (assuming no resource access delays), preemptions due to higher priority tasks on the same core, and delays due to accesses by the task itself, higher priority tasks on the same core, and also accesses due to tasks on other cores in the same time window. The overall framework uses an outer fixed point iteration to handle the dependencies between the WCRTs of tasks on different cores. The evaluation examines the performance of different types of interconnect, including a cross-bar, a FCFS bus, and a FCFS bus that supports two accesses in parallel. It shows that the proposed approach is much less pessimistic than assuming the worst-case delay for every bus access that a task makes.

Schranzhofer et al. published a series of three papers [112–114] during 2010 and 2011 based on the *superblock* model. In this model each task is represented by a sequence of superblocks, which may have branches and loops within them. The task set is assumed to be periodic and all execution on a core is assumed to be described by a sequence of superblocks (i.e. an offline static cyclic schedule). The first paper [112] considers TDMA bus arbitration. The second paper [113] provides worst-case completion time analysis for the superblocks and hence a schedulability test. It covers three models: (i) arbitrary memory accesses anywhere in the superblocks, (ii) dedicated phases where accesses only occur at the start and end of each superblock, and (iii) a hybrid where most accesses are in the acquisition and replication phases, but some accesses can take place in the intervening execution phase. The conclusion is that the dedicated model improves schedulability due to less contention. The third paper [114] extends the approach to an adaptive bus arbiter with both static and dynamic segments. The method uses a dynamic programming technique to compute the minimum amount of execution of a superblock which must have taken place to reach a particular frame in the arbitration schedule, considering the interference due to requests from tasks executing on other cores. The task WCRTs are then computed from this information.

In 2010, Pellizzoni et al. [98] also considered the superblock model. They assume that the maximum number of memory accesses made by an individual task (or by a set of tasks) in a given time interval can be upper bounded by an arrival curve. They derive arrival curves for both un-buffered accesses from a task which stalls waiting on the request and buffered accesses from DMA. A dynamic programming technique is then used to compute the maximum interference on the memory requests of a given task, and hence the worst-case delays due to memory bus interference under Round-Robin, fixed priority, or TDMA bus arbitration.

In 2012, Giannopoulou et al. [42] proposed a state-based response time analysis based on Timed Automata for multi-core systems using the superblock model. They model superblocks, static schedulers for the cores, and resource arbiters as Timed Automata with analysis provided via model checking. To improve scalability, each core is analysed separately, with interference from other cores abstracted via arrival curves.

Lampka et al. [77] published another paper using the superblock model in 2014. The approach in this paper relies on model checking and abstract interpretation combined with Real-Time Calculus. The aim here is to provide an upper-bound on the worst-case response time of concurrent tasks considering contention on shared memory. The authors first propose a method based on timed automata to precisely model the applications, the scheduler, shared resource accesses, and the arbitration policy used in the shared memory bus. Arbitration policies considered are Round-Robin, FCFS, and TDMA. The scalability of WCRT analysis is enhanced by abstracting the concurrent accesses to the shared resource with arrival curves from Real-Time Calculus.

In 2011, Dasari et al. [33] presented a response time analysis for partitioned tasks scheduled using fixed priority non-preemptive scheduling on a multi-core with a shared memory bus, which utilises a work-conserving arbitration policy. The bus contention from each task is modelled via a request function that returns the maximum number of bus requests that can be made by the task in a given time interval. The request function is computed by examining all paths through the code of the task and determining the pattern of accesses. Alternatively, it can be estimated via measurements using performance counters. The request function is integrated into schedulability analysis as an additional term in the WCRT equations. Since the request function for each task depends on the task's WCRT, an outer fixed point iteration is made over all tasks, until all response times converge to fixed values or a deadline is exceeded.

In 2012, Yun et al. [135] proposed a software method to control resource contention on a shared memory bus on COTS multi-cores. This work assumes a dedicated core which executes all critical real-time tasks. Contention due to tasks running on other cores is then limited by memory throttling. The idea is to set a budget of Q bus requests in any period P. If this budget is exceeded, as checked via performance counters, then the tasks on the non-critical core(s) are suspended until the period ends. The budget may be applied statically to individual cores or dynamically to a group of non-critical cores. The authors show how to incorporate an extra interference term into standard response time analysis to account for the interference from other cores. This term is limited via the parameters of the memory throttling. The evaluation shows that the dynamic approach is more effective and provides the non-critical cores with better performance, as they are throttled for less time.

4.2 Interference Contending for Memory

Research papers in this sub-category focus mainly on analysing contention over shared cache, while other works consider scratchpad memories and DRAM.

An allocation algorithm for placing instructions into local scratchpads and shared memory to reduce the WCET in multi-core systems was proposed by Suhendra et al. [121] in 2010. This algorithm also calculates scratchpad re-loading points to leverage the memory requirements of tasks with disjoint lifetimes. Each task can only access its own local scratchpad, and the latency of any access to memory (scratchpad or main memory) is considered bounded by a constant. This work also includes a response time analysis that takes into account the scratchpad allocation.

In 2011, Chattopadhyay et al. [25] presented an algorithm to allocate (at compile time) data variables into scratchpad and global shared memory in order to minimize the WCRT of the tasks. They provide a response time analysis that considers a scratchpad allocation, assuming an architecture where each core has a local scratchpad, and all scratchpads can be joined to form a virtual scratchpad space accessible by any core. The proposed allocation takes into account the delays to access the shared memory via a TDMA bus.

In 2009, Guan et al. [48] presented a cache-aware global fixed priority non-preemptive scheduling policy for multi-core systems with a shared L2 cache which can be partitioned at the level of cache blocks. When a core is idle, the policy selects the highest priority ready task to execute; however, the task only runs if sufficient L2 cache blocks are available, hence the policy is not work-conserving.

The authors note that allowing lower priority tasks requiring fewer blocks to run ahead of higher priority tasks (i.e. making the policy work-conserving) could result in unbounded blocking. They present two schedulability tests for the policy, one which uses a Linear Programming formulation, and a second which uses a closed form approximation. Evaluation shows that the second test has good performance, similar to the first.

In 2012, Liang et al. [79] presented a worst-case response time analysis for concurrent programs running on a multi-core system with a shared instruction cache. They assume that communication between the tasks is based on message passing, with the structure of concurrent programs captured via graphs of message sequence charts. The analysis derived obtains lower WCRTs than previous shared-cache analysis methods. The authors also show the benefits of cache locking, which improves the task WCRTs through a reduction in the number of conflicts.

In 2017, Zhang et al. [138] integrated analysis of cache-related preemption delays into schedulability analysis for global EDF scheduling. The main extension over prior work (for single processor systems) is recognising that in an m-core system, the m tasks with the shortest deadlines will not be preempted. The evaluation shows that taking this into account results in a modest improvement in schedulability. We note that a single shared cache is assumed for all cores; however, there is no consideration of the cache interference that would inevitably occur due to tasks executing in parallel on different cores and sharing the same cache.

In 2017, Xiao et al. [126] considered interference on shared caches in a multi-core system with local L1 caches, and a shared L2 cache. The task model assumes non-preemptive fixed priority scheduling on each core and a static partitioning of tasks to cores. This results in only inter-core interference on the shared cache (the processor bus is ignored). The authors propose an ILP formulation to bound the interference per task within its execution window. This interference is then included in the WCRT analysis, with the bound iteratively refined until a fixed-point is reached.

In a paper in 2014 [71] and the extension to it in 2016 [72], Kim et al. provided detailed analysis of memory interference due to contention for banked DRAM in COTS multi-core systems. The analysis takes account of the scheduling of DRAM access, different latencies due to closed/open rows (row hit, row conflict), and the FR-FCFS policy used by the DRAM controller. The model used assumes DRAM bank partitioning, with tasks allocated to the same core sharing the same bank partition. The analysis considers both inter-bank and intra-bank interference. Two approaches are taken to bounding the memory interference delay: *request-driven*, which uses only the parameters of the DRAM and processor, and *job-driven*, which uses only the parameters of the contending jobs, i.e. the number of memory requests. The interference delays are integrated into response time analysis for fixed priority preemptive scheduling. A task allocation algorithm is presented that aims to group memory intensive tasks together on the same core using the same DRAM bank partition, which substantially reduces contention. The evaluation shows that in the case of memory-intensive tasks there are large increases in response times if banks are shared. The proposed allocation algorithm is thus shown to be far more effective than prior works that perform allocation without considering memory requests.

In 2015, Yun et al. [134] identified that the work of Kim et al. [71] could be optimistic for modern COTS multi-core processors due to not considering that the same core can generate multiple outstanding memory requests. The authors propose a new interference analysis that takes into account multiple requests. The analysis also considers separate buffers for read and write requests. It is assumed that both the LLC and DRAM memory are partitioned. The experiments show that the analysis produces safe upper-bounds on the interference; however, they also highlight how COTS processors can induce pathological arrival patterns that greatly affect the worst-case delays.

Research that overlaps with this category but is reviewed elsewhere in this survey includes the work of Kafshdooz et al. [64] (see Section 5.1).

4.3 Interference Contending for Multiple Resources

Research papers in this sub-category provide analysis which covers interference due to contention for a number of different resources.

In 2015, Altmeyer et al. [6] introduced a Multi-core Response Time Analysis (MRTA) framework for explicit interference analysis. This framework combines the *resource demands* that tasks place on difference types of resources, e.g. the CPU, cache, memory bus, DRAM etc. and how these demands are satisfied by the *resource supply* provided by those hardware components. The work was extended by Davis et al. [37] in 2017. The framework is instantiated for cold / warmed-up caches, static and dynamic scratchpads, a variety of memory bus arbitration policies, including: Round-Robin, FIFO, Fixed-Priority, Processor-Priority, and TDMA, and accounts for DRAM refreshes. The sporadic task model is used as a basis, extended to include RTOS context switch behaviour and interrupt handlers, and shared software resources. The analysis assumes partitioned fixed priority preemptive scheduling. The evaluation covers different types of local memory, and a comparison between predictable, isolation, and reference architectures. This shows that a predictable architecture with scratchpads and a fixed priority bus, can substantially outperform an equivalent isolation architecture with partitioned caches and a TDMA bus, in terms of guaranteed real-time performance. The analysis results are also compared to those obtained via simulation. The evaluation indicates that the memory bus is typically the main bottleneck, and that fixed priority arbitration on the bus is more effective than other policies.

Rihani et al. [104] built upon the MRTA framework [6, 37] in 2016, extending the approach to consider mono-rate task graphs describing Synchronous Data-Flow applications, and taking account of the times at which particular tasks can run. They also tailor the analysis to the behaviour of the hardware resources of the Kalray MPPA-256 many-core processor, in particular the complex multi-level bus-arbitration policies used. The authors propose a method to compute an offline (static) schedule for the tasks that is feasible, accounting for the interference on shared resources. Starting from a given mapping of tasks to cores, and an execution order for each task that provides initial release dates, the analysis computes task response times taking into account the shared bus arbitration policy, and the interference from co-runners. The evaluation shows that the use of memory banks, and a consideration of the time at which other tasks can run are both important in achieving effective real-time performance guarantees.

In 2016, Huang et al. [60] presented an approach to schedulability analysis for partitioned multi-core systems using fixed priority preemptive scheduling, assuming contention for a shared resource. This analysis is based on recognising a symmetry between processing and resource access. A task can be viewed as executing and then suspending execution while it accesses the shared resource, or as accessing the resource and suspending access while it executes on the processor. The authors extend the standard sporadic task model with information about the amount of time for shared resource accesses and the maximum number of access segments. The analysis given compares well / improves upon the schedulability results obtained via the analysis of Altmeyer et al. [6]. It also provides performance only a little worse than the exact analysis for the phased-execution task model given by Melani et al. [85].

Cheng et al. [27], in 2017, built upon the analysis presented by Huang [60] extending it to systems with multiple memory banks which can be accessed independently and have a non-preemptive fixed priority arbitration policy. They assume that the total number of memory accesses and also the number of segments of consecutive accesses are known for each task. The authors observe that it is better to map all accesses from a single task to the same memory bank, since this reduces blocking,

and to also spread out the tasks among banks to reduce interference on accesses. The authors prove speedup and memory augmentation bounds for the task and bank allocation algorithms and the schedulability test that they propose. The evaluation considers a system similar to an island of the Kalray MPPA 256, with 8 cores and 8 memory banks. It shows that the proposed allocation algorithm significantly improves on the approach of Kim et al. [72] (see Section 4.2).

In 2016, Choi et al. [31] considered shared resource contention on a multi-core processor, with tasks that have dependencies (precedence relationships). Partitioned fixed priority preemptive or non-preemptive scheduling is assumed. The key idea is to model the maximum resource demand accounting for the separation between tasks and how they are clustered together and so execute in a particular order on specific cores. The resource demand function is then incorporated into response time analysis forming a nested iteration. Analysis of task clustering uses separation information based on the earliest and latest start and finish times for jobs and their precedence constraints to determine a fixed execution order for clusters of tasks. The evaluation shows that considering dependencies in this way has a useful impact in improving schedulability by reducing the resource demand in a given time window.

In 2018, Andersson et al. [9] presented a sufficient schedulability test with pseudo-polynomial complexity for partitioned fixed-priority preemptive scheduling on multi-core systems. Here, concurrent tasks are assumed to have co-runner-dependent execution times due to the effect of contention on shared resources. The system model proposed relies on a set of parameters, such as execution times, that can be obtained through static analysis or measurement-based methods. The evaluation of the approach is performed on a real-world multi-core by randomly generating tasksets and comparing the results of the schedulability test with measured execution times.

Also in 2018, Farshchi et al. [40] presented a memory design including RTOS-support for multi-core architectures to provide bounded memory access times. To this end, a memory management unit (MMU) is designed to ensure hardware-based isolation of the shared cache, and a DRAM controller to provide dedicated access to the DRAM banks. The bounded memory access times are then exploited in a response-time analysis, assuming fixed-priority preemptive scheduling. The starting point for the analysis is the tasks' WCETs in isolation, and the number of memory accesses per task. The main focus of this work is on the hardware design, but it also provides a response-time analysis to showcase how the hardware-based isolation can be exploited.

In 2018, Saidi and Syring [106] used arrival curves to model memory access interference at three levels: (i) local scratchpad memory, (ii) DRAM, and (iii) DMA, for a many-core architecture with local interconnect to banked memory and an external main memory. The set of models (memory access, DRAM and DMA latency bounds, bank and rank interference) are used to estimate a global response time. The main novelties are the model of the data locality captured by extended event streams, and the memory access granularity and interleaving. The authors show that their approach leads to a precise worst-case response time bound.

Research that integrates interference across a number of shared resources into schedulability analysis but is reviewed elsewhere in this survey includes the work of Boniol et al. [17] (see Section 2.3), Mancuso et al. [83, 84] and Yao et al. [131] (see Section 3.2), and Skalistis et al. [116, 117] (see Section 5.3).

4.4 Critique and Perspectives

The state-of-the-art is reflected in the papers on explicit interference analysis, which is a highly extensible approach that can be tailored to cater for different types of resources, different arbitration policies, and different ways of scheduling tasks. The main advantage of the approach stems from the fact that it models interference by considering the total demand on each resource over a time window which equates to the response time of the task under analysis. This leads to much less

pessimistic WCRTs than summing the worst-case resource delays over short time intervals, for example on a per access basis.

Many of the works which integrate the effects of contention over resources into schedulability analysis, suffer from an apparent circularity problem. The response times of tasks depend on the amount of resource contention, which in turn depends on the time interval considered (i.e. the response time of the task under analysis). This problem can however be solved via a nested fixed point iteration. Although this means that the analysis has pseudo-polynomial computational complexity, it is typically still feasible for practical sized systems. We note that explicit interference analysis is agnostic to the issues of timing composability. When used to analyse systems that provide some form of hardware or software isolation (for example a TDMA bus), then it does not require information about the resource demands of tasks on other cores. In contrast, with for example a Round-Robin bus, both time-composable and more precise non-time-composable analyses are possible.

Integrating the effects of contention into schedulability analysis has a significant advantage over integration into timing analysis; it can take into account the scheduling *context*. For example the set of potentially co-running tasks, the number of resource access requests that can occur within the response time of the task under analysis, and information about the pattern of requests can all potentially be integrated into the analysis, making it more precise.

Many of the research works in this area effectively consider serialised access to resources, and so do not take advantage of the overlap between processor and resource usage that often occurs in practice. For example, much of the literature assumes an atomic transaction bus, while many COTS multi-cores have a split-transaction bus. Parallelism (non-blocking caches, multi-bank DRAM, and memory controllers) is an important aspect of modern hardware, yet it is ignored by most of the analysis techniques surveyed. This current drawback represents a key opportunity to improve this form of analysis in the future.

5 MAPPING AND SCHEDULING

Research in this category concerns mapping and scheduling techniques. Mapping is the procedure to assign, at *design time*, tasks to processors (i.e. partition the task set) and often also *data* to a part of the memory (e.g. scratchpads). The mapping problem is as complex as the bin-packing problem which is an NP-Hard problem. Consequently if $P \neq NP$ then the problem cannot be solved exactly/optimally in polynomial time. Scheduling concerns the *run time* ordering and management of the assigned tasks. By construction the scheduling of each core can be considered to be independent thanks to the partitioning. Most of the papers reviewed define an *optimisation criterion* to express the efficiency of the solution. Different methods are considered to find an optimal solution, or to solve the problem sub-optimally using heuristics. Below we subdivide the literature depending on the *main* optimisation method used since papers often propose both exact and heuristic solutions: (i) ILP and MILP, (ii) Constraint Programming and Dynamic Programming, (iii) bin-packing, and (iv) Genetic Algorithms.

5.1 ILP and MILP

An integer linear programming (ILP) problem is a mathematical optimisation problem in which some or all of the variables are restricted to integers. The objective function (the function to optimise) and the constraints (other than the integer constraints) are *linear*.

In 2006, Suhendra et al. [120] presented an integrated algorithm to build an offline (static) schedule and map tasks in a theoretical multi-core system with scratch-pad memories. The solution is formulated as an ILP problem. It makes the simplifying assumption that access costs to global off-chip memory and remote scratchpad memories are constant. The approach calculates an optimal

partitioning and allocation of data to the scratchpads. The optimisation criterion is to minimise the initiation intervals of task graphs run under pipelined scheduling (i.e. a new instance of the task graph is initiated before the previous one has finished).

In 2012 [80] and a subsequent extension in 2015, Liu and Zhang [81] provided a WCET analysis/optimisation for multi-core systems with scratchpad memories, either as local memories, global memory or both. They focus solely on how to divide the scratchpad memory among the tasks in order to minimise the WCETs. This is done using an ILP-based partitioning algorithm, which assigns data objects to the shared scratchpad memories. Note the time to load/store data to other levels of memory is assumed to be statically bounded (i.e. the effect of the interconnect is ignored).

In 2013, Ding et al. [39] presented an ILP based optimization algorithm to perform task-to-core mappings aiming to minimize the WCRT of tasks by taking into account both the interference in the shared cache and a proper load balancing. They target applications modelled as a task graph and non preemptive scheduling.

In 2014, Kim et al. [76] described a scratchpad management technique for software managed multi-cores (SMM). SMMs have a scratchpad for each execution core. These cores can only directly access code from their scratchpad. Thus functions must be loaded into regions of the scratchpad before they can execute. The *mapping* of functions to regions of the scratchpad has an impact on the average-case and worst-case execution time of the program. The authors present two WCET-aware *mapping* techniques. The first technique can find the optimal mapping solution for WCET and is based on ILP. Because the number of function-to-region mapping choices grows exponentially with the number of functions in the program, this technique does not scale to large programs. Thus, they also present a polynomial-time heuristic that is scalable, but sub-optimal.

The trade-off between cache partitioning and cache sharing of the LLC for mixed-criticality systems was examined by Chisholm et al. [30] in 2015. The execution time bounds for the different criticality-levels are derived via measurements. The effect of cache partitioning/sharing are then added to these bounds depending on the size of the allocated partition. The paper presents a Linear Programming (LP) based optimisation of the LLC organisation to improve schedulability.

In 2015, Kafshdooz et al. [64] introduced a way to allow re-use of allocated Scratchpad Memory (SPM) space, whereby one task can release memory such that another task can allocate it. SPM allocation and task scheduling and mapping are formulated as an optimisation problem that is solved at design time. Two methods are proposed: (i) optimal solution with precise WCET estimation using ILP for small systems (2 cores) (ii) near-optimal solution using heuristics for larger systems (4 cores).

In 2016, Becker et al. [15] presented an approach to time-triggered scheduling for automotive "runnables" on a many-core platform (similar to the Kalyray MPPA 256). Bank privatisation is used to avoid contention between memory accesses from different cores (each core has a private bank). Runnables are assumed to have read-execute-write phases. The mapping executes each runnable non-preemptively, and ensures that accesses to the shared bank made in read and write phases do not overlap, thus avoiding contention on that shared memory. The mapping problem is formulated as an ILP. A Memory Centric Heuristic is also proposed and compared with a default Core Centric Heuristic, the former is significantly more effective, it is also much faster than the ILP for large systems representative of automotive applications.

If some decision variables are *not discrete* the problem is known as a mixed-integer linear programming problem (MILP) two recent contributions considered that method.

In 2016, Chisholm et al. [29] studied the MC^2 framework which uses hardware management to provide isolation between tasks of different critically levels running on different cores. In particular, it allocates regions of LLC using way-locking and page colouring, and also partitions the DRAM banks. This work builds upon that framework, adding a consideration of shared data. Shared data

is problematic, since it cuts across the isolation approach used. Three options are explored for lessening the impact of data sharing: (i) lock the shared buffer in the LLC, (ii) bypass the LLC so that the shared buffer is always accessed from memory, (iii) assign the tasks that share the buffer to the same core, so that accesses are effectively serialised and the buffer can then be cached. The MILP used in mapping tasks and regions of the LLC and DRAM in the MC^2 framework is extended to include sizing of the amount of shared buffers locked in the LLC (the most effective method).

An integrated task scheduling and cache locking/partitioning approach was presented by Zheng et al. [139] in 2017. The processor consists of m homogeneous cores with local L1 caches and a shared L2 cache. Effects of the shared bus are ignored. The aim of this work is to schedule sporadic tasks with given release times and deadlines to the m-cores while simultaneously partitioning the L1 and L2 cache taking into account the task interference. Partitioning here means that each task is given a share of the cache in which it locks its cache content. The cache locking and partitioning is therefore dynamic. The scheduling algorithm *aims to avoid preemption*; if a task is executed non-preemptively it can use the entire L1-cache. The optimisation criterion is the number of preemptions. The results show that the multi-core cache locking and scheduling algorithm can improve over a simple extension of a single-core based cache-locking approach in terms of task response times.

5.2 Constraint Programming and Dynamic Programming

Constraint programming is a programming paradigm based on relations between variables. The program deduces the solution to the problem based on these relations (the constraints). Dynamic programming a mathematical optimisation method, which can be used to reduce a complicated problem by breaking it down into simpler sub-problems in a recursive manner.

In 2016, Cheng et al. [28] considered the same model as Chang et al. [21] (see Section 5.3). They show that minimising the maximum core utilisation is NP-hard, and further that there is no *polynomial time approximation algorithm* (unless P=NP). They also show that minimising the maximum system utilisation (max of memory utilisation of a local memory and core utilisation) with no constraints on these values is also NP-hard. The authors propose a solution to the latter problem which achieves a lower bound approximation when the memory allocation is given. This solution is based on a method of optimal memory allocation to balance the memory and core utilisation of the tasks using dynamic programming. A greedy worst-fit algorithm is then used to allocate the tasks to cores. Evaluation shows that the proposed algorithm is effective and that the total local memory need only be 10–20% of the overall global memory requirement to achieve good performance.

The problem of mapping large applications on the Kalray MPPA platform so as to satisfy timing requirements while guaranteeing spatial and temporal isolation between applications was addressed by Perret et al. [100] in 2016. The properties to be guaranteed are: spatial partitioning inside compute clusters; isolated accesses to the NoC; offline pre-defined static buffers; and isolated accesses to the DDR-SRAM bank. The mapping problem is formulated using constraint programming (specifically the notion of conditional time-intervals), which is more scalable that existing ILP formulations of the problem. The aim of this work is to help application designers provide valid budgets for each application in terms of required processing, communication and memory resources. Note that this allocation approach is used in other work by Perret et al. [102] summarised in Section 3.3.

In two papers in 2016 [116] and 2017 [117] Skalistis et al. presented an interference based WCET analysis for data flow applications running on many-core platforms. The system is defined by a Unified System Model that includes a mapping of tasks to clusters, and adds communicating tasks that handle inter-cluster communications. From this model an initial WCET is obtained including an over-estimation of the inter-cluster and intra-cluster interference, which is used to perform an

initial mapping of tasks to resources, and a calculation of an offline (static) schedule using SMT solvers. An iterative method is then used that tightens the initial WCET estimation by eliminating impossible interference (i.e. from tasks that do not overlap in time and/or space). The schedule is recalculated making sure no new interference is added.

Research that overlaps with this category but is reviewed elsewhere in this survey includes the work of Kim et al. [74] (see Section 3.1).

5.3 Bin-Packing

The task partitioning problem is a form of *Bin-Packing* problem: k objects of different sizes must be packed into a finite number of bins of capacity B in a way that minimises the number of bins used. In our context, objects correspond to tasks and bins correspond to processor cores. The problem is known to be strongly NP-Complete. Many heuristics have been developed, e.g. next-fit, best-fit, worst-fit etc. which are fast, but often yield sub-optimal solutions.

In 2006, Anderson et al. [7] considered identical multiprocessors with 2 levels of cache (L1 local, L2 global). The aim being to reduce L2 cache misses. The main idea is to group the various tasks (named megatasks) such that the tasks of the same megatask can be co-scheduled, i.e. the data does not exceed the L2 cache size. The paper proposes a hierarchical Pfair-based scheduler for tasks and megatasks. The evaluation shows that L2 cache misses are significantly reduced.

In 2011, Paolieri et al. [95] introduced an algorithm to allocate tasks to shared memories and cores. The allocation is in two steps: first, the algorithm pre-allocates tasks to cores assuming a "non-interfering" WCET; Second, the execution time is updated with interference costs. The aim is to minimise the resources assigned to the hard real-time tasks. The interference cost is estimated and stored in a matrix that is used to determine the interference delay depending on which cores the tasks are allocated to. The interference analysis takes into account the shared bus (Round-Robin) and a memory controller (DDR2-SDRAM). The cache memories (L1 and L2) are partitioned through bankization.

In 2012, Chang et al. [22] considered the problem of mapping tasks to identical multiprocessors with heterogeneous shared memory of two types: a fast scratchpad-like memory, and a slow DRAM like memory. The authors propose a two step method. First, two algorithms (dynamic programming and a faster greedy approach) are used to satisfy the memory requirements of the tasks for each memory type such that the maximum task utilisation is minimised. The WCET of the tasks are calculated using the tool aiT. Second, the tasks are ordered according to their utilisations in a non increasing manner, and then sequentially assigned to the lowest utilisation processor.

Later in 2015, Chang et al. [21] studied the problem of task allocation on an island or cluster based multi-core system, with each cluster having fast local memory, which can be partitioned between tasks running on the cores of the cluster. The authors present algorithms aimed at solving two problems: (i) task allocation to cores when the number of local memory blocks each task uses is also part of the allocation problem, (ii) when the number of local memory blocks required by each task is fixed. A large scale case study shows that typically relatively small local memories (in comparison to the total memory usage of the tasks) are sufficient to substantially improve schedulability. Further, when the designer is free to allocate the number of memory blocks for each task much more efficient solutions can be found.

In 2013, Kim et al. [73] presented an approach to cache management for multi-core systems. Their approach consists of three components: (i) *cache reservation* that ensures exclusive use of a number of partitions of the shared cache for each core, thus eliminating inter-core cache interference; (ii) *cache sharing* that enables tasks on the same core to share cache partitions (if necessary); and (iii) *cache-aware task allocation* that utilises the other two components to find an efficient allocation that ensures task schedulability. The authors show how the cache warm-up delays and cache-related

preemption delays for their scheme can be integrated into response time analysis, assuming fixed priority preemptive scheduling of a subset of the tasks on each core. The cache-aware allocation algorithm sorts the tasks in order of decreasing utilisation, and employs a Best-Fit approach to allocation. The overall approach is software-based, using page colouring, thus it is applicable to COTS multi-cores.

In two papers in 2013 [88] and 2015 [87] Nikolic et al. study NoC-based many-core platforms. They consider the latencies of memory operations and propose two methods to obtain the upper-bound on the worst-case memory traffic delays of individual applications. The main contribution is a formulation of the problem of *mapping* on many-core platforms under the limited migrative model where tasks can only execute on a subset of cores. The authors propose a method which consists of three stages. In the initial phase all high-priority applications are mapped, and are not be subject to changes in later phases. During the feasibility phase, all low-priority applications are mapped, with the primary objective being to derive a feasible solution where all the applications are mapped. Lastly, during the optimisation phase the feasible solution is improved.

In 2016, Xu et al. [127] presented a cache-aware global fixed priority preemptive scheduling algorithm for multi-core systems called gFPca. Only shared cache is considered, and it is assumed to be partitioned. Under gFPca, tasks are scheduled according to their priorities and the number of cache partitions they require, with cache partitions assigned to jobs at run-time. A formal analysis is provided, including the delays produced due to reloading the cache contents after each preemption.

Research that overlaps with this category but is reviewed elsewhere in this survey includes the work of Andrei et al. [10] (see Section 2.3), Alhammad et al. [3, 4] (see Section 3.1), Agrawal et al. [1] (see Section 3.2), Giannopoulou et al. [43–45], Carle et al. [19] (see Section 3.3), and Cheng et al. [27] (see Section 4.3)

5.4 Genetic Algorithms

Genetic algorithms are a method of solving optimisation problems inspired by the process of natural selection.

In 2018, Still et al. [119] considered the problem of mapping tasks with their communication requirements to cores connected via a NoC. The authors note that all prior work on such mappings ignore the constraint that the tasks need to fit in the core's local memory. They explore three memory models for the requirements of tasks: (i) receiving only, (ii) sending and receiving, (iii) sending and receiving and code. A Multi-Objective Genetic Algorithm is used to perform the mapping. The authors note that convergence is somewhat slower with the additional memory constraint, but the method is still effective.

5.5 Critique and Perspectives

The mapping and scheduling techniques reviewed in this section effectively target a more complicated problem than the pure timing verification approaches reviewed in the other sections: They combine the problem of the timing verification with the search for an optimal or a feasible mapping or scheduling solution so that the timing requirements can be met. For the approaches described in the other sections, the mapping and scheduling is typically assumed to be fixed a priori. In other words, these approaches often leave the question open, where the mapping comes from, or if the mapping is chosen reasonably. The mapping and scheduling techniques, on the other hand, typically resort to a more abstract view of the multi-core system. Often, only a single shared component, like the memory or the communication bus is modelled, and the interference on all other components is assumed to be subsumed into the task's execution time bound. Other mapping approaches resort to a highly abstract model of interference. In these cases it is often questionable, how reliable or precise the estimates are, on which the mapping and scheduling decisions are made.

A notable exception is the work of Anderson et al. [7], which provides measurements to show that the computed mapping indeed improves the system performance; however, showing that no deadline misses can occur is not the aim of this work.

Despite the high level of abstraction employed in most of the surveyed techniques, scalability remains an issue. The mapping and scheduling problems formulated in this category are often NP-hard in the number of tasks and/or cores, so approximation techniques are necessary for many-core architectures and also in some cases for multi-core systems.

The mapping and scheduling techniques complement the timing verification techniques presented in the earlier sections, answering the question about how to map and schedule tasks on the cores. However, due to the abstract view taken of the system, they do not as yet provide a satisfying stand-alone solution to the overall allocation and timing verification problem. This is an area which merits further research.

6 OBSERVATIONS DISCUSSION AND CONCLUSIONS

In this section, we discuss the advantages and disadvantages of the main approaches used and their potential in providing a high level of guaranteed real-time performance. Finally, we conclude by identifying key open questions and important directions for future research.

6.1 Discussion

The present is a time of transition and transformation, with major industrial sectors such as automotive and aerospace moving cautiously forwards from using single-core processors to using multi-core processors in their deployed systems. These efforts are however being severely hampered by the difficulties in applying the techniques used in the analysis of past single-core systems to the rapidly evolving hardware platforms of the future. This problem is so serious that it has led to the practice of disabling all but one core severely undermining the advantages of using high performance multi-core platforms.

The problem is that the hardware platforms used for real-time systems are transforming. Firstly, there is no longer a single type of resource that needs to be considered in verifying worst-case timing behaviour, but rather multiple resource types (processors, co-processors and Graphical Processing Units (GPUs), memory hierarchies including data and instruction cache and scratchpads, inter-connects and networks, and I/O devices etc). Secondly, for valid commercial reasons of reduced cost, increased average-case performance and energy efficiency, the vast majority of COTS hardware platforms share resources between multiple cores. For example, the LLC, memory bus and DRAM are typically shared between a set of cores to realise Symmetric Multi-Processing (SMP), which is specifically targeted to optimise the average case. On the other hand, since different application tasks run in parallel on different cores, this gives rise to contention over the shared resources and hence the possibility for co-runner interference that can be difficult to predict. This greatly exacerbates both the difficulty and the complexity of providing accurate timing verification. In the subsections below, we discuss the potential for the main analysis approaches surveyed to solve this problem.

6.1.1 Full Isolation and the Traditional Two Step Approach. The traditional two-step approach to timing verification for single-core systems (i.e. timing analysis followed by schedulability analysis) can be used to a certain extent for multi-core systems. For example, timing analysis developed for single-core systems can be used to provide WCET information for tasks executing *in isolation* on a multi-core. It can also be used to bound WCETs when shared resource accesses are subject to delays which can be tightly bounded in a context-independent manner, as is the case with a TDMA bus. The problem here is that such *full isolation* limits the available bandwidth even

when other cores are not using the resource. Effectively the resource is no longer shared, it is divided. This is heavily detrimental to average-case performance and power consumption, which is why non-work-conserving protocols such as TDMA are rarely if ever used in COTS multi-core processors, even though TDMA is favoured in many research papers. While arbitration protocols such as Round-Robin can result in better average-case behaviour in practice, two-step analysis must rely on worst-case context-independent assumptions and can therefore only provide heavily degraded guarantees in this case.

In our view, the traditional two-step approach breaks down when applied to complex COTS multi-core systems, due to excessive pessimism. It may, however, remain applicable for some specialised high-integrity systems reliant on special purpose multi-core hardware that has been specifically designed to provide full temporal isolation. In such systems the trade-off in performance is worthwhile to ease predictability. Such temporal isolation also fully supports timing composition, as it facilitates the timing verification of an application on one core independent of the applications running on the other cores.

6.1.2 Full Integration. At the opposite extreme to the traditional two-step approach and full isolation are the approaches based on *full integration*. Such approaches attempt to account for all possible patterns of interleaving of co-runner execution by combining both timing analysis and schedulability analysis of all resources into a single step. Fully integrated approaches thus require detailed information about all co-runners and their precise release times, which must be enforced by the operating system. While full integration offers the promise of precise analysis, these approaches suffer from serious problems of complexity, due to an exponential explosion in the number of states examined. As a result, they are unlikely to scale to advanced real-time systems built on COTS multi-core hardware platforms. The need for full details of all co-runners is also at odds with the use of 3rd party software and design practices such as incremental verification. A fully integrated approach is also problematic when considering applications of different criticality, since the guaranteed performance of a high-criticality application on one core may be reliant on the behaviour of low-criticality applications on other cores. This dependency would require all applications to be developed to the standard required by the highest criticality application, which is both impractical and prohibitively expensive.

Over the past 10 years, there have been relatively few papers published on fully integrated analysis, with a downward trend in terms of the proportion of the overall research in the area surveyed over recent years. This could be an indication that the fully integration approach has entered the diminishing returns stage while the research community has identified other more practical and promising avenues of research.

6.1.3 Partial Isolation and Limited Interference. As indicated above, fully integrated analysis of multi-core systems where applications access multiple shared resources as well as the processing cores faces potentially insurmountable problems of complexity and scalability. As a consequence, many research works take a *divide and conquer* approach, breaking the analysis problem down into a number of simpler ones. This is achieved by either assuming context-independent upper bounds for delays on resource accesses, or by enforcing either complete or partial isolation from the effects of co-runner contention. When the hardware architecture does not provide the means for such isolation, then it can be obtained through software techniques by controlling the time or frequency at which shared resource accesses can be made from each core. This can be done via phased execution models (separating and scheduling memory access and processing phases), memory bandwidth regulation (limiting the number of co-runner resource accesses), and via offline (static) scheduling (separating when different tasks run on different cores). We note that more than half of the papers studied consider some form of partial isolation and limited interference.

Partial isolation techniques provide more predictable access to shared hardware resources and consequently simpler analysis for timing verification (compared to full integration approaches). The main trade-off is that the isolation mechanisms used still compromise average-case performance and energy consumption, with the software techniques also adding overheads. Worthwhile gains in guaranteed performance can however be obtained compared to full isolation approaches. Further, by controlling the bandwidth or timing of shared resource accesses, co-runner interference can be bounded *independently* of the actual behaviour of the co-runners, thus supporting time composability. An open question with temporal isolation techniques is whether the techniques and their analyses can be improved to allow better utilisation of the available computation resources.

6.1.4 Integrating Interference into Schedulability Analysis. Integrating the effects of co-runner interference into the timing analysis of a task has the disadvantage that the delays included typically have to assume the worst-case context-independent values, which can be highly pessimistic. In reality, the worst-cases delays may occur only rarely on some very specific accesses, but cannot possibly occur on every access. An alternative approach is to integrate the effects of co-runner contention over shared resources into schedulability analysis. This has two key advantages. First, interference from co-runners can be modelled by considering the total resource demand over the response time of each task, which typically leads to much less pessimistic analysis than assuming the worst-case delays for every individual resource access. Second, the analysis can take into account the scheduling context, i.e. the specific set of possible co-runners and the resource accesses that they can make. We note here that both time-composable and more precise non-time composable analyses are possible. An open problem identified in the literature is about how to take into account the overlaps in accesses to different resources (e.g. processing cores, interconnect, and memory). Integration of co-runner interference into schedulability analysis is a promising approach to reducing the pessimism in timing verification of multi-core systems. Research in this area is exhibiting an increasing trend with a large number of papers published over the past 4 years.

6.1.5 The Hypothesis of Timing Compositionality. The idea of timing compositionality depends on the decomposition of the system into components for analysis purposes. For example, the WCET of a task might be decomposed into the memory access time and the processing time. If the local worst-cases for each of these components can be summed to obtain an upper bound on the overall worst-case timing behaviour, then the the decomposition is said to be *timing compositional*. The papers reviewed in this survey (particularly those that integrate interference effects into schedulability analysis) typically assume, either explicitly or implicitly, a timing compositional architecture, free of any timing anomalies or domino effects [53]. This is a strong hypothesis, since as far as we know, there is currently no proof that *any* existing COTS multi-core hardware platform has this property. Research presented by Hahn et al. [52] in 2016 introduced a method to bound the effects of timing anomalies, opening the door to potentially using current COTS platforms as if they were timing compositional. Such techniques are necessary to validate the approaches that rely on the timing compositionability of the underlying hardware, since they would otherwise not be applicable to real-world systems.

6.2 Conclusions

In the future (2020 onward), real-time systems will involve multiple, concurrent, complex applications with a variety of different criticality levels, executing on advanced hardware platforms. These hardware platforms will consist of multiple cores, perhaps interconnected in small clusters or islands that are themselves connected to other clusters. The cores will be a mix of different types with different performance characteristics, able to execute at different speeds with different levels of energy consumption. The cores will also share many different types of hardware resources,

including memory hierarchies that are composed of multiple levels of cache or scratchpads, and main memory that is composed of multiple banks that can be accessed in parallel, as well as a variety of different I/O and communications devices. Such systems will need to be robust, resilient to faults, adaptable and flexible to changes in their environment and operating conditions. For instance, they will be able to reconfigure in the event of failure or damage to components, and in the more extreme situations, they will gracefully degrade.

In our opinion, we have reached a point where it is arguably no longer possible to deliver mass-market future-proof real-time systems relying only on static approaches (e.g. static resource allocation, bandwidth provision). Instead, the real-time requirements of such systems could be guaranteed via the use of a collection of dynamic run-time strategies that mitigate interference, underpinned by off-line analysis techniques that guarantee real-time performance. These strategies include dynamic resource allocation and bandwidth regulation, dynamic reconfiguration and re-scheduling; adaptive and anytime algorithms that tailor execution demands to the available resources; and control over core processing speeds. The key idea being that a supervisory system could closely monitor performance, and if it detects that an application could potentially violate its timing constraints then take appropriate steps, via the use of control mechanisms and policies, to ensure that such a timing violation cannot occur. During operation of the system, these dynamic mechanisms would be sufficient to ensure that no timing violation can occur with strong guarantees provided via offline analysis. Such a dynamic approach could also provide substantial improvements in average-case performance and energy consumption over approaches based on isolation while also supporting timing composition.

Before this future can be realised in a reliable, cost-effective and energy-efficient way, the following long term open questions must be addressed:

(1) How to characterise, in a precise way, application resource demands and platform resource supplies;
(2) How to detect that an application may be heading for a timing violation, and what policies and control mechanisms to apply as a guaranteed preventive step to ensure timing correctness;
(3) How to provide explicit, integrated interference analysis that delivers holistic timing verification for complex, adaptive real-time systems, accounting for the regulation and countermeasures applied by the supervisory system;
(4) How to ensure that the developed systems are robust to small changes in resources, and resilient to larger ones, thus providing graceful degradation;
(5) How to ensure extensibility for new applications, upgrades and modifications;
(6) How to optimise hardware platform configurations so that worst-case performance can be guaranteed, with minimal if any reduction in average-case performance.
(7) How to analyse systems with heterogeneous cores (e.g. FPGA, GPU) and speeds;
(8) How to leverage variable processing speeds to provide guaranteed timing behaviour while also minimising energy consumption and heat dissipation on realistic multi-core platforms;
(9) How to provide timing verification for multi-core systems where global or semi-partition scheduling and task migration is supported;
(10) How to provide timing verification for many-core platforms that do not rely on isolation techniques.

For high-integrity systems built on advanced multi-core or many-core architectures, compliance with standards, and/or certification is typically required. The approach taken in the guidance issued in the *Position Paper CAST32-A Multi-core Processors* [20], for the avionics domain, focuses on "Robust Resource and Time Partitioning" and hence approaches based on isolation or at least partial isolation. Similarly, in the automotive domain, the ISO26262 standard [91] focuses on "Freedom

From Interference" in the temporal domain. A final open question is how the relevant standards and certification requirements can adapt to enable the deployment of the complex multi- and many-core systems envisaged at the start of this section. Here, it is clear that a co-evolution of both techniques and certification standards / guidelines is needed.

To conclude, in this survey we recognise that there is intense interest in the real-time research community to embrace and exploit the potential benefits of multi-core systems. Although significant progress have been made during the last decade, there are still many important unanswered questions and open issues that need to be fully resolved.

ACKNOWLEDGMENTS

The authors would like to thank Sophie Quinton for the interesting discussions that were the origin of this paper and Leandro Indrusiak for interesting discussions on the state-of-the-art research on analysis of NoCs. The research that went into writing this paper was funded in part by the ESPRC grant MCCps (EP/P003664/1), the Innovate UK grant SECT-AIR (113099), and by the Wallonia Region (Belgium) BEWARE grant PARTITA (1610375). EPSRC Research Data Management: No new primary data was created during this study

REFERENCES

[1] A. Agrawal, G. Fohler, J. Freitag, J. Nowotsch, S. Uhrig, and M. Paulitsch. 2017. Contention-Aware Dynamic Memory Bandwidth Isolation With Predictability in COTS Multicores : An Avionics Case Study. *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)* 2 (2017), 1–2.

[2] A. Agrawal, R. Mancuso, R. Pellizzoni, and G. Fohler. 2018. Analysis of Dynamic Memory Bandwidth Regulation in Multi-core Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 230–241.

[3] A. Alhammad and R. Pellizzoni. 2014. Schedulability analysis of global memory-predictable scheduling. In *Proceedings of the IEEE & ACM International Conference on Embedded Software (EMSOFT)*. 1–10.

[4] A. Alhammad and R. Pellizzoni. 2014. Time-predictable execution of multithreaded applications on multicore systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 1–6.

[5] A. Alhammad, S. Wasly, and R. Pellizzoni. 2015. Memory efficient global scheduling of real-time tasks. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 285–296.

[6] S. Altmeyer, R. I Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. 2015. A Generic and Compositional Framework for Multicore Response Time Analysis. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. 129–138.

[7] J. H. Anderson, J. M. Calandrino, and U. C. Devi. 2006. Real-Time Scheduling on Multicore Platforms. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 179–190.

[8] B. Andersson, A. Easwaran, and J. Lee. 2010. Finding an Upper Bound on the Increase in Execution Time Due to Contention on the Memory Bus in COTS-based Multicore Systems. *SIGBED Rev.* 7, 1, Article 4 (Jan. 2010), 4 pages.

[9] B. Andersson, H. Kim, D. de Niz, M. Klein, R. Rajkumar, and J. Lehoczky. 2018. Schedulability Analysis of Tasks with Corunner-Dependent Execution Times. *ACM Transactions on Embedded Computing Systems* 17, 3 (2018), 71:1–71:29.

[10] A. Andrei, P. Eles, Z. Peng, and J. Rosen. 2008. Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In *21st International Conference on VLSI Design (VLSID 2008)*. 103–110.

[11] M.A. Awan, P.F. Souto, K. Bletsas, B. Akesson, and E. Tovar. 2018. Mixed-criticality Scheduling with Memory Bandwidth Regulation. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*.

[12] M.A. Awan, P.F. Souto, K. Bletsas, B. Akesson, and E. Tovar. 2018. Worst-case Stall Analysis for Multicore Architectures with Two Memory Controllers. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 2:1–2:22.

[13] M. A. Awan, P. F. Souto, B. Akesson, K. Bletsas, and E. Tovar. 2018. Uneven memory regulation for scheduling IMA applications on multi-core platforms. *Real-Time Systems* (16 Nov 2018).

[14] S. Bak, G. Yao, R. Pellizzoni, and M. Caccamo. 2012. Memory-Aware Scheduling of Multicore Task Sets for Real-Time Systems. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 300–309.

[15] M. Becker, D. Dasari, B. Nicolic, B. Åkesson, V. Nélis, and T. Nolte. 2016. Contention-Free Execution of Automotive Applications on a Clustered Many-Core Platform. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 14–24.

[16] M. Behnam, R. Inam, T. Nolte, and M. Sjödin. 2013. Multi-core Composability in the Face of Memory-bus Contention. *SIGBED Rev.* 10, 3 (Oct. 2013), 35–42.

[17] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti. 2012. Deterministic Execution Model on COTS Hardware. In *Architecture of Computing Systems – ARCS 2012*. 98–110.

[18] T. Carle and H. Cassé. 2018. Reducing Timing Interferences in Real-Time Applications Running on Multicore Architectures. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, Vol. 63. 3:1–3:12.

[19] T. Carle, D. Potop-Butucaru, Y. Sorel, and D. Lesens. 2015. From Dataflow Specification to Multiprocessor Partitioned Time-triggered Real-time Implementation. *Leibniz Transactions on Embedded Systems (LITES)* 2, 2 (2015), 01:1—-01:30.

[20] Certification Authorities Software Team (CAST). 2016. *Position Paper CAST-32A Multi-core Processors*. Technical Report.

[21] CW. Chang, JJ. Chen, TW. Kuo, and H. Falk. 2015. Real-Time Task Scheduling on Island-Based Multi-Core Platforms. *IEEE Transactions on Parallel and Distributed Systems* 26, 2 (Feb 2015), 538–550.

[22] C. W. Chang, J. J. Chen, W. Munawar, T. W. Kuo, and H. Falk. 2012. Partitioned scheduling for real-time tasks on multiprocessor embedded systems with programmable shared srams. In *Proceedings of the tenth ACM international conference on Embedded software*. ACM, 153–162.

[23] S. Chattopadhyay, L.K. Chong, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk. 2014. A unified WCET analysis framework for multicore platforms. *ACM Transactions on Embedded Computing Systems (TECS)* 13, 4s (2014), 124.

[24] S. Chattopadhyay, C. L. Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk. 2012. A Unified WCET Analysis Framework for Multi-core Platforms. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 99–108.

[25] S. Chattopadhyay and A. Roychoudhury. 2011. Static Bus Schedule Aware Scratchpad Allocation in Multiprocessors. *SIGPLAN Not.* 46, 5 (April 2011), 11–20.

[26] S. Chattopadhyay, A. Roychoudhury, and T. Mitra. 2010. Modeling Shared Cache and Bus in Multi-cores for Timing Analysis. In *Proceedings of the 13th International Workshop on Software and Compilers for Embedded Systems (SCOPES '10)*. ACM, 6:1—-6:10.

[27] SW. Cheng, JJ. Chen, J. Reineke, and TW. Kuo. 2017. Memory Bank Partitioning for Fixed-Priority Tasks in a Multi-core System. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 209–219.

[28] S. W. Cheng, C. W. Chang, J. J. Chen, T. W. Kuo, and P. C. Hsiu. 2016. Many-Core Real-Time Task Scheduling with Scratchpad Memory. *IEEE Transactions on Parallel and Distributed Systems* 27, 10 (Oct 2016), 2953–2966.

[29] M. Chisholm, N. Kim, B. C Ward, N. Otterness, J.H. Anderson, and F.D. Smith. 2016. Reconciling the tension between hardware isolation and data sharing in mixed-criticality, multicore systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 57–68.

[30] M. Chisholm, B. C. Ward, N. Kim, and J. H. Anderson. 2015. Cache Sharing and Isolation Tradeoffs in Multicore Mixed-Criticality Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 305–316.

[31] J. Choi, D. Kang, and S. Ha. 2016. Conservative modeling of shared resource contention for dependent tasks in partitioned multi-core systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 181–186.

[32] D. Dasari, B. Akesson, V. Nelis, M. A. Awan, and S. M. Petters. 2013. Identifying the sources of unpredictability in COTS-based multicore systems. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*. 39–48.

[33] D. Dasari, B. Andersson, V. Nelis, S. M. Petters, A. Easwaran, and J. Lee. 2011. Response Time Analysis of COTS-Based Multicores Considering the Contention on the Shared Memory Bus. In *International Conference on Trust, Security and Privacy in Computing and Communications*. 1068–1075.

[34] D. Dasari and V. Nelis. 2012. An Analysis of the Impact of Bus Contention on the WCET in Multicores. In *Proceedings of the IEEE International Conference on High Performance Computing and Communication*. IEEE Computer Society, Washington, DC, USA, 1450–1457.

[35] D. Dasari, V. Nelis, and B. Akesson. 2015. A framework for memory contention analysis in multi-core platforms. *Real-Time Systems* (2015), 1–51.

[36] D. Dasari, V. Nelis, and Benny Akesson. 2016. A framework for memory contention analysis in multi-core platforms. *Real-Time Systems* 52, 3 (01 May 2016), 272–322.

[37] R. I Davis, S. Altmeyer, L. S Indrusiak, C. Maiza, V. Nelis, and J. Reineke. 2017. An extensible framework for multicore response time analysis. *Real-Time Systems* (2017).

[38] R. I. Davis and A. Burns. 2011. A Survey of Hard Real-time Scheduling for Multiprocessor Systems. *ACM Comput. Surv.* 43, 4, Article 35 (Oct. 2011), 44 pages.

[39] H. Ding, Y. Liang, and T. Mitra. 2013. Shared cache aware task mapping for WCRT minimization. In *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 735–740.

[40] F. Farshchi, P. K. Valsan, R. Mancuso, and H. Yun. 2018. Deterministic Memory Abstraction and Supporting Multicore System Architecture. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 1:1–1:25.

[41] J. Freitag, S. Uhrig, and T. Ungerer. 2018. Virtual Timing Isolation for Mixed-Criticality Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, Vol. 106. 13:1–13:23.

[42] G. Giannopoulou, K. Lampka, N. Stoimenov, and L. Thiele. 2012. Timed model checking with abstractions: Towards worst-case response time analysis in resource-sharing manycore systems. In *Proceedings of the tenth ACM international conference on Embedded software*. ACM, 63–72.

[43] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. 2013. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *Proceedings of the IEEE & ACM International Conference on Embedded Software (EMSOFT)*. 1–15.

[44] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. 2014. Mapping mixed-criticality applications on multi-core architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 1–6.

[45] G. Giannopoulou, N. Stoimenov, P. Huang, L. Thiele, and B.D. de Dinechin. 2016. Mixed-criticality scheduling on cluster-based manycores with shared communication and storage resources. *Real-Time Systems* 52, 4 (2016), 399–449.

[46] G. Gracioli, A. Alhammad, R. Mancuso, A. A. Fröhlich, and R. Pellizzoni. 2015. A Survey on Cache Management Mechanisms for Real-Time Embedded Systems. *ACM Comput. Surv.* 48, 2, Article 32 (Nov. 2015), 36 pages.

[47] G. Gracioli and A. A. FrÃűhlich. 2014. On the Influence of Shared Memory Contention in Real-Time Multicore Applications. In *Brazilian Symposium on Computing Systems Engineering*. 25–30.

[48] N. Guan, M. Stigge, W. Yi, and G. Yu. 2009. Cache-aware Scheduling and Analysis for Multicores. In *Proceedings of the IEEE & ACM International Conference on Embedded Software (EMSOFT)*. 245–254.

[49] D. Guo, M. Hassan, R. Pellizzoni, and H. Patel. 2018. A Comparative Study of Predictable DRAM Controllers. *ACM Transactions on Embedded Computing Systems* 17, 2, Article 53 (Feb. 2018), 23 pages.

[50] Z. Guo, Y. Zhang, L. Wang, and Z. Zhang. 2017. Work-in-Progress: Cache-Aware Partitioned EDF Scheduling for Multi-core Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 384–386.

[51] A. Gustavsson, A. Ermedahl, B. Lisper, and P. Pettersson. 2010. Towards WCET Analysis of Multicore Architectures Using UPPAAL. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*. 101–112.

[52] S. Hahn, M. Jacobs, and J. Reineke. 2016. Enabling Compositionality for Multicore Timing Analysis. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. 299–308.

[53] S. Hahn, J. Reineke, and R. Wilhelm. 2013. Towards Compositionality in Execution Time Analysis – Definition and Challenges. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*.

[54] D. Hardy, T. Piquet, and I. Puaut. 2009. Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 68–77.

[55] M. Hassan. 2018. On the Off-Chip Memory Latency of Real-Time Systems: Is DDR DRAM Really the Best Option?. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 495–505.

[56] M. Hassan, A. M. Kaushik, and H. Patel. 2017. Predictable Cache Coherence for Multi-core Real-Time Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 235–246.

[57] M. Hassan and R. Pellizzoni. 2018. Bounding DRAM Interference in COTS Heterogeneous MPSoCs for Mixed Criticality Systems. In *Proceedings of the IEEE & ACM International Conference on Embedded Software (EMSOFT)*.

[58] F. Hebbache, M. Jan, F. Brandner, and L. Pautet. 2018. Shedding the Shackles of Time-Division Multiplexing. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 456–468.

[59] S. Hesham, J. Rettkowski, D. Goehringer, and M. A. Abd El Ghany. 2017. Survey on Real-Time Networks-on-Chip. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (May 2017), 1500–1517.

[60] W.H. Huang, J.J. Chen, and J. Reineke. 2016. MIRROR: Symmetric Timing Analysis for Real-Time Tasks on Multicore Platforms with Shared Resources. In *Proceedings of the Design Automation Conference (DAC)*. 1–6.

[61] R. Inam, M. Behnam, T. Nolte, and M. Sjödin. 2015. Compositional analysis for the Multi-Resource Server. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*. 1–10.

[62] M. Jacobs, S. Hahn, and S. Hack. 2015. WCET analysis for multi-core processors with shared buses and event-driven bus arbitration. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. 193–202.

[63] M. Jacobs, S. Hahn, and S. Hack. 2016. A framework for the derivation of WCET analyses for multi-core processors. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 141–151.

[64] M.M. Kafshdooz and A. Ejlali. 2015. Dynamic Shared SPM Reuse for Real-Time Multicore Embedded Systems. *ACM Trans. Archit. Code Optim.* 12, 2, Article 12 (May 2015), 25 pages.

[65] T. Kelter, H. Borghorst, and P. Marwedel. 2014. WCET-aware scheduling optimizations for multi-core real-time systems. In *Proceedings International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. 67–74.

[66] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. 2011. Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 3–12.

[67] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. 2014. Static Analysis of Multi-core TDMA Resource Arbitration Delays. *Real-Time Systems* 50, 2 (2014), 185–229.

[68] T. Kelter, T. Harde, P. Marwedel, and H. Falk. 2013. Evaluation of resource arbitration methods for multi-core real-time systems. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, Vol. 30. 1–10.

[69] T. Kelter and P. Marwedel. 2015. Parallelism Analysis: Precise WCET Values for Complex Multi-Core Systems. In *Formal Techniques for Safety-Critical Systems*. 142–158.

[70] A.E. Kiasari, A. Jantsch, and Z. Lu. 2013. Mathematical Formalisms for Performance Evaluation of Networks-on-chip. *ACM Comput. Surv.* 45, 3, Article 38 (July 2013), 41 pages.

[71] H. Kim, D. De Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. 2014. Bounding memory interference delay in COTS-based multi-core systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 145–154.

[72] H. Kim, D. De Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar. 2016. Bounding and Reducing Memory Interference in COTS-based Multi-core Systems. *Real-Time Systems* 52, 3 (May 2016), 356–395.

[73] H. Kim, A. Kandhalu, and R. Rajkumar. 2013. A Coordinated Approach for Practical OS-Level Cache Management in Multi-core Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 80–89.

[74] JE. Kim, MK. Yoon, R. Bradford, and L. Sha. 2014. Integrated Modular Avionics (IMA) partition scheduling with conflict-free I/O for multicore avionics systems. *Proceedings - International Computer Software and Applications Conference* (2014), 321–331.

[75] JE. Kim, MK. Yoon, S. Im, R. Bradford, and L. Sha. 2013. 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)*. 970–975.

[76] Y. Kim, D. Broman, J. Cai, and A. Shrivastaval. 2014. WCET-aware dynamic code management on scratchpads for Software-Managed Multicores. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 179–188.

[77] K. Lampka, G. Giannopoulou, R. Pellizzoni, Z. Wu, and N. Stoimenov. 2014. A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets. *Real-Time Systems* 50, 5 (2014), 736–773.

[78] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury. 2009. Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 57–67.

[79] Y. Liang, H. Ding, T. Mitra, A. Roychoudhury, Y. Li, and V. Suhendra. 2012. Timing analysis of concurrent programs running on shared cache multi-cores. *Real-Time Systems* 48, 6 (2012), 638–680.

[80] Y. Liu and W. Zhang. 2012. Exploiting multi-level scratchpad memories for time-predictable multicore computing. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*. 61–66.

[81] Y. Liu and W. Zhang. 2015. Scratchpad Memory Architectures and Allocation Algorithms for Hard Real-Time Multicore Processors. *JCSE* 9 (2015).

[82] M. Lv, W. Yi, N. Guan, and G. Yu. 2010. Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. IEEE Computer Society, 339–349.

[83] R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. 2015. WCET (m) estimation in multi-core systems using single core equivalence. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 174–183.

[84] R. Mancuso, R. Pellizzoni, N. Tokcan, and M. Caccamo. 2017. WCET Derivation Under Single Core Equivalence With Explicit Memory Budget Assignment. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*.

[85] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. Buttazzo. 2015. Memory-Processor Co-Scheduling in Fixed Priority Systems. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. 87–96.

[86] K. Nagar and Y. N. Srikant. 2016. Fast and Precise Worst-Case Interference Placement for Shared Cache Analysis. *ACM Transactions on Embedded Computing Systems* 15, 3 (mar 2016), 1–26.

[87] B. Nikolic and S.M. Petters. 2015. Real-time application mapping for many-cores using a limited migrative model. *Real-Time Systems* 51, 3 (01 Jun 2015), 314–357.

[88] B. Nikolic, P. M. Yomsi, and S. M. Petters. 2013. Worst-case memory traffic analysis for many-cores using a limited migrative model. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 42–51.

[89] J. Nowotsch and M. Paulitsch. 2013. Quality of Service Capabilities for Hard Real-time Applications on Multi-core Processors. *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)* (2013), 151–160.

[90] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt. 2014. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 109–118.

[91] nternational Organization for Standardization. 2011. *ISO 26262 Road Vehicles Functional Safety*. Technical Report.

[92] D. Oehlert, A. Luppold, and H. Falk. 2017. Bus-Aware Static Instruction SPM Allocation for Multicore Hard Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*.

[93] C. Pagetti, J. Forget, H. Falk, D. Oehlert, and A. Luppold. 2018. Automated generation of time-predictable executables on multicore. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. ACM, 104–113.

[94] M. Paolieri, J. Mische, S. Metzlaff, M. Gerdes, E. Quiñones, S. Uhrig, T. Ungerer, and F. J. Cazorla. 2013. A Hard Real-time Capable Multi-core SMT Processor. *ACM Trans. Embed. Comput. Syst.* 12, 3 (2013), 79:1—-79:26.

[95] M. Paolieri, E. QuiÃśones, F. J. Cazorla, R. I. Davis, and M. Valero. 2011. IA^3: An Interference Aware Allocation Algorithm for Multicore Hard Real-Time Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 280–290.

[96] R Pellizzoni, E Betti, S Bak, G Yao, J Criswell, M Caccamo, and R Kegley. 2011. A Predictable Execution Model for COTS-Based Embedded Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 269–279.

[97] R. Pellizzoni, B. D. Bui, M. Caccamo, and L. Sha. 2008. Coscheduling of CPU and I/O Transactions in COTS-Based Embedded Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 221–231.

[98] R. Pellizzoni, A. Schranzhofer, J-J. Chen, M. Caccamo, and L. Thiele. 2010. Worst case delay analysis for memory interference in multicore systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 741–746.

[99] R. Pellizzoni and H. Yun. 2016. Memory Servers for Multicore Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 1–12.

[100] Q. Perret, P. Maurère, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet. 2016. Mapping Hard Real-time Applications on Many-core Processors. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. 235–244.

[101] Q. Perret, P. Maurere, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet. 2016. Predictable composition of memory accesses on many-core processors. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*.

[102] Q. Perret, P. Maurere, E. Noulard, C. Pagetti, P. Sainrat, and B. Triquet. 2016. Temporal Isolation of Hard Real-Time Applications on Many-Core Processors. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 1–11.

[103] H. Rihani, M. Moy, C. Maiza, and S. Altmeyer. 2015. WCET analysis in shared resources real-time systems with TDMA buses. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. ACM, 183–192.

[104] H. Rihani, M. Moy, C. Maiza, R.I. Davis, and S. Altmeyer. 2016. Response Time Analysis of Synchronous Data Flow Programs on a Many-Core Processor. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*. 67–76.

[105] J. Rosèn, A. Andrei, P. Eles, and Z. Peng. 2007. Bus Access Optimization for Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*.

[106] S. Saidi and A. Syring. 2018. Exploiting Locality for the Performance Analysis of Shared Memory Systems in MPSoCs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 350–360.

[107] S. Schliecker and R. Ernst. 2011. Real-time Performance Analysis of Multiprocessor Systems with Shared Memory. *ACM Transactions on Embedded Computing Systems* 10, 2, Article 22 (Jan. 2011), 27 pages.

[108] S. Schliecker, M. Negrean, and R. Ernst. 2010. Bounding the Shared Resource Load for the Performance Analysis of Multiprocessor Systems. In *Proceedings of the Design Automation Conference (DAC)*. 759–764.

[109] S. Schliecker, M. Negrean, G. Nicolescu, P. Paulin, and R. Ernst. 2008. Reliable Performance Analysis of a Multicore Multithreaded System-on-chip. In *Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. 161–166.

[110] S. Schliecker, J. Rox, M. Negrean, K. Richter, M. Jersak, and R. Ernst. 2009. System Level Performance Analysis for Real-Time Automotive Multicore and Network Architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems* 28, 7 (2009), 979–992.

[111] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi. 2015. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture* 61, 9 (2015), 449–471.

[112] A. Schranzhofer, JJ. Chen, and L. Thiele. 2010. Timing Analysis for TDMA Arbitration in Resource Sharing Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 215–224.

[113] A. Schranzhofer, R. Pellizzoni, J. J. Chen, L. Thiele, and M. Caccamo. 2010. Worst-case response time analysis of resource access models in multi-core systems. In *Proceedings of the Design Automation Conference (DAC)*. 332–337.

[114] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo. 2011. Timing Analysis for Resource Access Interference on Adaptive Resource Arbiters. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 213–222.

[115] J. P. Shen and M. H. Lipasti. 2013. *Modern Processor Design: Fundamentals of Superscalar Processors*. Waveland Press.
[116] S. Skalistis and A. Simalatsar. 2016. Worst-Case Execution Time Analysis for Many-Core Architectures with NoC. In *International Conference on Formal Modeling and Analysis of Timed Systems*. 211–227.
[117] S. Skalistis and A. Simalatsar. 2017. Near-optimal deployment of dataflow applications on many-core platforms with real-time guarantees. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 752–757.
[118] P. Souto, P. B. Sousa, R. I. Davis, K. Bletsas, and E. Tovar. 2015. Overhead-Aware Schedulability Evaluation of Semi-Partitioned Real-Time Schedulers. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 110–121.
[119] L. R. Still and L. S. Indrusiak. 2018. Memory-Aware Genetic Algorithms for Task Mapping on Hard Real-Time Networks-on-Chip. In *Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*.
[120] V. Suhendra, C. Raghavan, and T. Mitra. 2006. Integrated Scratchpad Memory Optimization and Task Scheduling for MPSoC Architectures. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. ACM, New York, NY, USA, 401–410.
[121] V. Suhendra, A. Roychoudhury, and T. Mitra. 2010. Scratchpad Allocation for Concurrent Embedded Software. *ACM Trans. Program. Lang. Syst.* 32, 4, Article 13 (April 2010), 47 pages.
[122] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. S Phatak, and R. Pellizzoni. 2016. A Real-Time Scratchpad-centric OS for Multi-core Embedded Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.
[123] P. K. Valsan, H. Yun, and F. Farshchi. 2016. Taming Non-Blocking Caches to Improve Isolation in Multicore Real-Time Systems. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 1–12.
[124] P. K. Valsan, H. Yun, and F. Farshchi. 2017. Addressing isolation challenges of non-blocking caches for multicore real-time systems. *Real-Time Systems* 53, 5 (2017), 673–708.
[125] S. Wasly and R. Pellizzoni. 2014. Hiding Memory Latency Using Fixed Priority Scheduling. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 75–86.
[126] J. Xiao, S. Altmeyer, and A. Pimentel. 2017. Schedulability Analysis of Non-preemptive Real-Time Scheduling for Multicore Processors with Shared Caches. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 199–208.
[127] M. Xu, L.T.X. Phan, HY. Choi, and I. Lee. 2016. Analysis and implementation of global preemptive fixed-priority scheduling with dynamic cache allocation. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 1–12.
[128] J. Yan and W. Zhang. 2008. WCET Analysis for Multi-Core Processors with Shared L2 Instruction Caches. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 80–89.
[129] G. Yao, R. Pellizzoni, S. Bak, E. Betti, and M. Caccamo. 2012. Memory-centric scheduling for multicore hard real-time systems. *Real-Time Systems* 48, 6 (nov 2012), 681–715.
[130] G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo. 2016. Global Real-Time Memory-Centric Scheduling for Multicore Systems. *IEEE Transactions on Computers* 65, 9 (sep 2016), 2739–2751.
[131] G. Yao, H. Yun, Z.P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha. 2016. Schedulability Analysis for Memory Bandwidth Regulated Multicore Real-Time Systems. *IEEE Transactions on Computers* 65, 2 (feb 2016), 601–614.
[132] M. K. Yoon, J. E. Kim, and L. Sha. 2011. Optimizing Tunable WCET with Shared Resource Allocation and Arbitration in Hard Real-Time Multicore Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*. 227–238.
[133] H. Yun, R. Mancuso, Z. P. Wu, and R. Pellizzoni. 2014. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicore platforms. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 155–166.
[134] H. Yun, R. Pellizzoni, and P.K. Valsan. 2015. Parallelism-aware memory interference delay analysis for cots multicore systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, 184–195.
[135] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. 2012. Memory Access Control in Multiprocessor for Real-Time Systems with Mixed Criticality. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*. 299–308.
[136] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. 2013. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 55–64.
[137] W. Zhang and J. Yan. 2009. Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 455–463.
[138] Y. Zhang, Z. Guo, L. Wang, H. Xiong, and Z. Zhang. 2017. Integrating Cache-Related Preemption Delay into GEDF Analysis for Multiprocessor Scheduling with On-chip Cache. In *IEEE Trustcom/BigDataSE/ICESS*. 815–822.
[139] W. Zheng, H. Wu, and C. Nie. 2017. Integrating task scheduling and cache locking for multicore real-time embedded systems. In *Proceedings of the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. 71–80.

SUPPLEMENTARY MATERIAL

A DETAILED CLASSIFICATION

In this supplementary section, we present a detailed classification of existing approaches to timing verification for multi-core systems. The criteria used are grouped into four categories:

A.1 Type of timing verification performed;
A.2 Timing model used for the analysis.
A.3 Hardware properties (relevant to multi-core timing verification);
A.4 Software properties (that can make the analysis more efficient or more precise);

Typographical convention. In the descriptions below and in the classification table that follows, we assume that criteria, and consequently column names in the table, are written in a **bold** font, while values inside a column are underlined.

A.1 Type of Timing Verification Performed

We distinguish between papers that provide different forms of analysis, focusing on interference, WCET, scheduling, and mapping. Note, some papers cover more than one of these four types of analysis.

***I**-Interference Analysis.* Identifies papers which characterise the interference that a task may experience when it is not executed in isolation. For example, quantifying extra delays due to the use of a bus with an arbitration policy such as Round-Robin, TDMA or FIFO. Interference analysis depends strongly on the hardware characteristics described in Section A.3.

***W**-WCET Analysis.* Identifies papers which aim to provide WCET bounds for a set of tasks executing on the target platform. This analysis may consider the task running in isolation (i.e. software or hardware isolation), or it may take into account (partially or completely) different types of interference.

***S**-Scheduling and schedulability Analysis.* Identifies papers that study a scheduling problem, that is, given a task set and the platform characteristics, they study if the tasks meet their timing constraints when executed under a particular scheduling algorithm.

***M**-Mapping Analysis.* Identifies papers that define a strategy to allocate tasks to the various cores. The workload must be spread over the hardware platform feasibly so that timing constraints, memory constraints, and any affinity requirements are met. Note, we include in this category work on both static mapping, and dynamic re-configuration which may change the mapping at runtime from one allocation to another.

A.2 Timing Model Used for the Analysis

Different timing models can be used for the same type of analysis. More detailed models can yield more precise results, but are more difficult to handle. We identify the types of interference delays that are explicitly dealt with. There are two explanations for a type of delay that is not explicitly part of a timing model: (i) such delays are guaranteed to be zero due to some system property, for example task migrations may not be permitted and so there are no task migration delays; (ii) the delays are assumed to be included within the WCET time bound provided for the task.

***E**-Execution Time bound.* Unless they are computed by the analysis, the timing model always assumes that task WCET bounds are provided.

***γ**-Cache-related preemption delay.* Such delays are relevant in preemptive systems with cache memory.

***Mi**-Migration delay.* Migration delays exist whenever the execution of a task may be suspended on one core and resumed on another.

***ID**-Interference delay.* Covers other forms of interference that a task may suffer due to other tasks executing on the multi-core platform. This includes the blocking time due to accesses to shared resources, but not cache-related preemption or migration delays. This interference delay depends on the type of multi-core hardware considered (see Section A.3).

A.3 Hardware Properties

The timing verification techniques applied to a multi-core system must be compatible with key aspects of the hardware architecture.

Relevant properties, used in our classification, are detailed below.

A.3.1 Architecture. Multi-core architectures can be very diverse. Here, we identify platform properties and the number of cores as being key aspects relevant to timing verification.

***Pt**-Platform properties.* Identify aspects of the architecture which have a strong influence on timing verification. Platforms may be Predictable (P), Many-core (M), Heterogeneous (H), Common-Off-The-Shelf (C) as described below. Papers where the platform is unclassified in this category consider only specific aspects of some generic (possibly hypothetical) multi-core. (We note that the set of platform properties are not orthogonal, for example the Kalray MPPA-256 is a COTS many-core platform).

Predictable (P) platforms are designed to reduce non-determinism with respect to timing in cores (pipelines), local memories (for example cache replacement policies), interconnections (buses and arbitration policies) and shared memory (e.g. memory controllers). Such platforms avoid sources of timing anomalies and domino effects. All components are timing composable, meaning that the interaction between the cores and the shared memory remains predictable even during contention. This can be combined with hardware isolation employing mechanisms for spatial isolation (for example using partitioned memory) or temporal isolation (for example TDMA arbitration policies).

Many-core (M) platforms are programmed differently from platforms with a smaller number of cores and may offer more options for isolation. Typically, on such platforms, the interconnect is a *Network-on-Chip* (NoC) and the protocols used in the NoC are predictable such that communication delays can be bounded. Many-core platforms are becoming popular with the emergence of industrial platforms targeting real-time systems (such as the Kalray MPPA-256).

Heterogeneous (H) platforms are either composed of otherwise *identical* cores with *different* processing speeds (known as *uniform* platforms) or composed of *different* kinds of cores, where jobs have different execution times on the different cores with no specific mathematical relations. In this kind of platform, some jobs can have restrictions or affinities, meaning that they are allowed to run only on a subset of the available cores. The execution time of a task depends on which type of core it is assigned to. Analysis of such systems needs to take into account these different execution times.

Common-Off-The-Shelf (C) platforms typically, but not always, have unknown or unpredictable timing behaviour. The objective of this criterion is to give an indication of the applicability of existing techniques to currently available platforms.

#C-Number of cores. We only consider works dealing with systems that have multiple cores. Even so, design choices and analysis techniques that have been validated on a system with a small

number of cores (typically 2 or 4 cores) may not be applicable to systems with a larger number of cores. For this reason, we use the notation 2+ to identify papers that focus on systems with two cores and claim a generalisation to more cores. The notation 1+ applies to papers focuing on systems with *full isolation*, such that each core may be analysed independently from the rest of the system. Papers that provide analysis which is applicable to an arbitrary number of cores are denoted by N (note in this case, partial isolation may also be used).

A.3.2 Memory. Different types of memories with different access times may be used in a multi-core architecture. Examples include caches, scratchpads and main memory. Memory that may be shared between multiple cores is referred to as *global memory*, while memory that is private to a single core is referred to as *local memory*. For example, in a multi-core system each core may have access to a private L1 cache, a Last Level Cache (LLC) that is shared between all cores, and finally main memory that is also accessible to all cores. In this case, the L1 cache is local, while the LLC and main memory are forms of global memory. Local as well as global memories may be banked[6]. Banked memory is mainly used to reduce the interference from concurrent accesses by other cores; however, it can also be used for a local memory to reduce the interference between tasks. Each memory bank has its own arbiter. Thus, accesses to addresses in different banks do not interfere with one another. This hardware configuration offers a form of *spatial isolation* that can be exploited at the application level to reduce interference (i.e. to provide at least partial temporal isolation).

#***L****-Memory levels.* There may be more than one level of local memory and more than one level of global memory. A larger number of levels typically provides the potential for higher performance, but also increases the complexity of the analysis.

LM*-Local memory.* Accesses to a local memory do not suffer from delays due to concurrent accesses by other cores. The classification table indicates the type of local memory that is used, as follows: cache (C), scratchpad memory (S), or no local memory(N). Further, banked memory is indicated by (BM).

GM*-Global memory.* Shared memories are typically significantly larger than local memories; however, accesses to global memories may suffer from interference. Global memories may be of the types listed above for local memories (C, S, N, or BM) If no type is given, the global memory is *unbanked* main memory.

A.3.3 Interconnection. Indicates how the cores and the memories are linked together.

IC*-Interconnect.* The interconnect may be a bus (B), a Network-on-Chip (NoC), or unspecified.

A.4 Software Properties

Software properties can be used to limit the complexity involved in the timing verification of multi-core systems or to improve its precision. Such properties can be guaranteed by the software design, obtained through static analysis or enforced at run time by the operating system. Here, we identify the most relevant ones in our classification.

Pr*-Precedence graph or synchronous data-flow.* For some applications it is possible to know where the data comes from and thus which tasks may share data. This can be used to increase precision in interference analysis (using knowledge of where the data is and which task may read or write

[6] A memory may also be partitioned using software; this will then appear in the *software isolation* column of the classification table.

it), and in the schedulability analysis (a task may not start to execute before all its predecessors have completed).

***Ph**-Phased execution.* Another popular way of limiting the duration of memory interference is to use phased execution where tasks first read all their inputs, then compute and finally write all their outputs. Thus, if a local memory is used and is sufficiently large to contain all instructions and data, the execution phase may be considered as local. It therefore does not interfere with the execution on other cores, nor suffer interference from them. The read or write phases, in contrast, are subject to interference. The principle of phased execution comes from the code generated from design level tools for reactive systems (e.g. Scade/Lustre or Simulink). This phased execution model has been extended in a number of ways. For example, shared data may be stored locally such that only write phases interfere. Phased execution is a useful way to achieve isolation, for example by employing a scheduling policy that avoids any concurrent read or write phases.

***Is**-Software isolation.* Refers to any mechanism that is implemented at the software level in order to isolate tasks or cores from one another. Such mechanisms may be implemented in a number of ways, for example through scheduling, using a hypervisor, memory bandwidth regulators, or by fixing some execution time window (software time division multiplexing). The common goal is that the mechanism bounds or eliminates interference.

A.5 Classification Table

The classification table below lists all of the 119 research papers[7] on multi-core timing verification reviewed in this survey. For each one, it provides a detailed classification according to the criteria and properties described above.

[7]Note some of the papers in the bibliography are related works that are not classified in this table.

	Type of analysis				Timing model				HW properties						SW properties		
Total: 119 papers	I	W	S	M	E	γ	Mi	ID	Pt	#C	#L	LM	GM	IC	Pr	Ph	Is
Agrawal et al. [1]		X	X		X			X	C	2+	1	N		NoC			X
Agrawal et al. [2]		X			X			X	C	2+	1	N		B			X
Alhammad and Pellizzoni [4]			X		X				C	2+	2	C		B		X	X
Alhammad and Pellizzoni [3]			X		X				C	1+	2	C		B		X	X
Alhammad et al. [5]			X		X					N	2	C, S		B		X	X
Altmeyer et al. [6]	X		X		X	X		X	P	N	2	C, S		B			
Anderson et al. [7]			X		X				M	N	2	C	C				X
Andersson et al. [8]	X				X			X	C	N	2	C		B			
Andersson et al. [9]			X		X			X	C	N	2	C					
Andrei et al. [10]	X	X	X		X			X		N	3	C		B			
Awan et al. [11]			X		X				C	2+	2	C	C	B		X	X
Awan et al. [13]	X		X		X			X	C	N	2	C	C, BM	B			X
Bak et al. [14]			X		X				C	2+	2	C		B		X	X
Becker et al. [15]			X	X	X				M	N	1		BM	B, NoC	X	X	X
Behnam et al. [16]			X		X				C	N	3	C	C	B			X
Boniol et al. [17]	X		X	X	X			X	C	2+	3	C		B	X	X	X
Carle et al. [19]			X		X					N	1			B	X	X	X
Carle and Cassé [18]	X		X		X			X	C	N	2	C		B			X
Chang et al. [22]				X	X					N	2	C					
Chang et al. [21]				X	X					N	2	S (partitioned)					
Chattopadhyay et al. [26]	X	X							I, P	2+	3	C	C	B			
Chattopadhyay and Roychoudhury [25]	X		X		X			X	P	N	2	S		B	X		
Chattopadhyay et al. [24]	X	X							P	2+	3	C	C	B			
Chattopadhyay et al. [23]	X	X							P	2+	3	C	C	B			
Cheng et al. [28]				X	X					N	2	S (partitioned)		B			
Cheng et al. [27]	X			X	X	X			P	N	1		BM	B			
Chisholm et al. [30]	X				X			X	C	N	3	C	C				
Chisholm et al. [29]	X			X	X			X	C	N	3	C	C, BM		X	X	X
Choi et al. [31]	X		X		X			X		N	1				X		
Dasari et al. [33]	X		X		X			X	C	N	1			B			
Dasari and Nelis [34]	X	X			X			X	C	N	1			B			
Dasari et al. [35]	X				X			X		N	1			B			
Dasari et al. [36]	X				X			X		N	1	C (partitioned)	C (partitioned)	B			
Davis et al. [37]	X		X		X	X		X	P	N	2	C, S	C, S	B			
Ding et al. [39]				X	X			X	C	N	3	C	C	B	X		
Farshchi et al. [40]			X		X			x	P	2+	2	C	BM	B			
Freitag et al. [41]	X								C, P	2+	2	C		B			

	Type of analysis				Timing model				HW properties						SW properties		
Total: 119 papers	I	W	S	M	E	γ	Mi	ID	Pt	#C	#L	LM	GM	IC	Pr	Ph	Is
Giannopoulou et al. [42]	X		X		X			X		N	1			B		X	
Giannopoulou et al. [43]	X		X	X	X			X		N	1			B		X	X
Giannopoulou et al. [44]	X		X	X	X			X	P	N	1		BM	B			X
Giannopoulou et al. [45]	X		X	X	X			X	M, C, P	N	1		BM	B, NoC	X		X
Guan et al. [48]			X		X					N	2	C					X
Guo et al. [50]	X			X	X	X				N	2	C		B			
Gustavsson et al. [51]	X	X							P	2+	3	C	C				
Hardy et al. [54]										N	N	C	C				
Huang et al. [60]			X	X	X			X	P	N	1			B			
Inam et al. [61]	X		X		X			X	C	N	2	C		B			X
Jacobs et al. [62]	X	X								4	2	C, S		B			
Jacobs et al. [63]	X	X								N	2	C, S		B			
Kafshdooz and Ejlali [64]		X	X		X			X		2+	3	S		B			
Kelter et al. [66]	X	X						X	P*	N	2	C, S		B			
Kelter et al. [68]	X								P	N	2	S		B			
Kelter et al. [67]	X	X						X	P*	N	2	C, S		B			
Kelter et al. [65]	X	X			X			X	P	N	1			B			
Kelter and Marwedel [69]	X	X								N	3	C, S		B			
Kim et al. [75]			X	X	X				C	N	1			B			X
Kim et al. [73]	X		X	X	X	X		X	C	N	2	C	C				X
Kim et al. [71]	X		X		X			X	C	N	N	C	C, BM				
Kim et al. [76]		X			X					1+	2	S		B			X
Kim et al. [74]			X		X				C	N	1			B			X
Kim et al. [72]			X	X	X			X	P	N	2		BM	B			
Lampka et al. [77]	X							X		N	3	C		B		X	X
Li et al. [78]		X			X			X	P	N	3	C	C	B	X		
Liang et al. [79]	X	X	X		X	X		X		2+	3	C	C		X		
Liu and Zhang [81]	X	X							P	2+	2	S	S				
Lv et al. [82]	X	X									2	C		B			
Mancuso et al. [83]	X		X		X			X	C	N	2	C	C, BM	B			X
Mancuso et al. [84]	X		X		X			X	C	N	2		C, BM	B			X
Melani et al. [85]			X		X					1+	2	C		B		X	X
Nagar and Srikant [86]	X	X			X			X		N	3	C	C	B			
Nikolic et al. [88]	X							X	M	N	1	N		NoC			
Nikolic and Petters [87]	X			X					M	N	1	N		NoC			
Nowotsch and Paulitsch [89]	X	X			X			X	C	N	2			NoC			X
Nowotsch et al. [90]	X	X			X			X	C	N	2			NoC			X

	Type of analysis				Timing model				HW properties						SW properties		
Total: 119 papers	I	W	S	M	E	γ	Mi	ID	Pt	#C	#L	LM	GM	IC	Pr	Ph	Is
Oehlert et al. [92]	X	X								N	2	S		B			
Pagetti et al. [93]		X	X	X					P	N	2	S		B	X	X	X
Paolieri et al. [95]	X			X	X			X	P	N	3	C, BM	C, BM	B			
Paolieri et al. [94]	X	X							P		3	C, S	C, BM	B			
Pellizzoni et al. [97]	X	X	X		X			X	C	1+	2		C	B			X
Pellizzoni et al. [98]	X		X		X			X	C	N	2	C		B			
Pellizzoni et al. [96]			X		X				C	1+	2	C		B		X	X
Pellizzoni and Yun [99]	X		X		X			X	C	N	2	C		B			X
Perret et al. [100]				X	X				M	N	2		P	NoC	X		X
Perret et al. [101]	X	X							M, C, P	N	2	BM	BM	B, NoC			
Perret et al. [102]									M, C, P	N	2	C	BM	NoC			X
Rihani et al. [103]	X	X							P	1+	1			B			
Rihani et al. [104]	X		X		X			X	M, C, P		1		BM	B	X	X	
Rosèn et al. [105]	X	X	X					X	P	N	2	C		B	X	X	
Saidi and Syring [106]	X		X					X	M	N	3	S	BM	B, NoC			
Schliecker et al. [109]	X		X		X			X		N	2	C					
Schliecker et al. [110]	X		X		X			X		N				B			
Schliecker et al. [108]	X		X		X	X		X	P	2+	2	C		B			
Schliecker and Ernst [107]	X		X		X	X		X	P	N	2	C		B			
Schoeberl et al. [111]	X	X							P	N	2	C, S		B, NoC			
Schranzhofer et al. [112]	X									N	1			B		X	
Schranzhofer et al. [113]	X		X		X			X	P	N	1			B	X	X	
Schranzhofer et al. [114]	X		X					X	P	N	1			B		X	
Skalistis and Simalatsar [116]	X	X						X	M	N	1	N	BM	NoC	X		
Skalistis and Simalatsar [117]	X		X	X	X			X	M	N	1	N	BM	NoC	X		
Souto et al. [118]			X		X	X	X			N							
Still and Indrusiak [119]				X	X					N	1	S		NoC			
Suhendra et al. [120]			X	X	X				H, P	N	2	S		B	X		
Suhendra et al. [121]	X		X		X				P	N	2	S		B	X		
Tabish et al. [122]			X		X				P	N	2	S		B		X	X
Valsan et al. [123]	X				X				C	N	3	C	C	B			X
Valsan et al. [124]	X				X				C	N	3	C	C	B			X
Wasly and Pellizzoni [125]			X		X					1+	2	S		B			X
Xiao et al. [126]	X		X		X			X	H,C	N	2	C	C				
Xu et al. [127]	X	X	X		X	X	X		C	N	1		C				
Yan and Zhang [128]		X			X			X		2+	3	C	C	B			
Yao et al. [129]			X		X					N	2	C	C	B		X	X

	Type of analysis				Timing model				HW properties						SW properties		
Total: 119 papers	I	W	S	M	E	γ	Mi	ID	Pt	#C	#L	LM	GM	IC	Pr	Ph	Is
Yao et al. [130]			X		X					N	2	C	C	B		X	X
Yao et al. [131]			X		X				C	2+	2	C	C	B		X	X
Yoon et al. [132]	X	X	X		X			X	P	N	2		C, BM	B			
Yun et al. [135]	X		X		X			X	C	2+	2	C		B			
Yun et al. [134]	X	X							C	N	3	C	C, BM				
Zhang and Yan [137]	X	X							P	2+	3	C	C				
Zhang et al. [138]	X		X		X	X				N	2		C				
Zheng et al. [139]			X	X	X			X		N	2	N	C	B	X		

Received October 2018; revised February 2019; accepted February 2019