

Mixed Criticality on Multicore/Manycore Platforms

Edited by

Sanjoy K. Baruah¹, Liliana Cucu-Grosjean², Robert I. Davis^{3,2},
and Claire Maiza⁴

1 University of North Carolina at Chapel Hill, US, baruah@cs.unc.edu

2 INRIA Roquencourt – Le Chesnay, FR, liliana.cucu@inria.fr

3 University of York, GB, rob.davis@york.ac.uk

4 VERIMAG – Gières, FR, claire.maiza@imag.fr

Abstract

This report provides an overview of the discussions, the program and the outcomes of the first Dagstuhl Seminar on Mixed Criticality on Multicore/Manycore Platforms. The seminar brought together researchers working on challenges related to executing mixed criticality real-time applications on multicore and manycore architectures with the main purpose of promoting a closer interaction between the sub-communities involved in real-time scheduling, real-time operating systems / runtime environments, and timing analysis as well as interaction with specialists in hardware architectures.

Seminar March 16–20, 2015 – <http://www.dagstuhl.de/15121>

1998 ACM Subject Classification C.3 Special-purpose and Application-based Systems – Real-time and embedded systems

Keywords and phrases Mixed-Criticality, Real-time systems, Multicore/Manycore Platforms, fixed priority; probabilistic scheduling, varying-speed processors, model combination

Digital Object Identifier 10.4230/DagRep.5.3.84

Edited in cooperation with Adriana Gogonel

1 Executive Summary

Liliana Cucu-Grosjean

Robert I. Davis

Claire Maiza

Sanjoy K. Baruah

License © Creative Commons BY 3.0 Unported license

© Liliana Cucu-Grosjean, Robert I. Davis, Claire Maiza, and Sanjoy K. Baruah

Real-time systems are characterised not only by the need for functional correctness, but also the need for timing correctness. Today, real-time embedded systems are found in many diverse application areas including; automotive electronics, avionics, and space systems. In these areas, technological progress is resulting in rapid increases in both software complexity and processing demands. To address the demand for increased processor performance, silicon vendors no longer concentrate on increasing processor clock speeds, as this approach has led to problems with high power consumption and excessive heat dissipation. Instead, technological development has shifted to multicore processors, with multiple CPUs integrated onto a single chip. The broad technology trend is towards much larger numbers of cores, referred to as manycore, requiring network-on-chip rather than bus interconnects.



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Mixed Criticality on Multicore/Manycore Platforms, *Dagstuhl Reports*, Vol. 5, Issue 3, pp. 84–142

Editors: Sanjoy K. Baruah, Liliana Cucu-Grosjean, Robert I. Davis, and Claire Maiza



Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Requirements on Size Weight and Power consumption, as well as unremitting cost pressures, are pushing developments in avionics and automotive electronics towards the adoption of powerful embedded multicore processors, with a longer term vision of migrating to manycore. With the adoption of such technology comes the opportunity to combine different applications on the same platform, potentially dramatically reducing assembly and production costs, while also improving reliability through a reduction in harnessing. Different applications may have different criticality levels (e.g. safety-critical, mission-critical, non-critical) designating the level of assurance needed against failure. For example, in automotive electronics, cruise control is a low criticality application, whereas electric steering assistance is of high criticality. In an aerospace context, flight control and surveillance applications in Unmanned Aerial Vehicles are of high and low criticality respectively. The very low acceptable failure rates (e.g. 10^{-9} failures per hour) for high criticality applications imply the need for significantly more rigorous and costly development and verification processes than required by low criticality applications.

Combining high and low criticality applications on the same hardware platform raises issues of time separation and composition; it must be possible to prevent the timing behaviour of high criticality applications from being disturbed by low criticality ones, otherwise both need to be engineered to the same rigorous and expensive standards. Simple methods of achieving this separation, such as time partitioning or allocation to different cores can however be wasteful of processing resources. They may require more expensive hardware than necessary, increasing production costs, which is something industry is strongly motivated to avoid. Time composability is needed so that the timing behaviour of applications, determined in isolation, remains valid when they are composed during system integration. Without time composability integration of complex applications would become infeasibly expensive. The transformation of real-time embedded systems into mixed criticality multicore and manycore systems is recognised as a strategically important research area in Europe and the USA.

The seminar focused on the two key conflicting requirements of Mixed Criticality Systems: separation between criticality levels for assurance and sharing for resource efficiency, along with the related requirement of time composability. The key research questions addressed were:

- How to provide effective guarantees of real-time performance to applications of different criticality levels via intelligent sharing of resources while respecting the requirements for asymmetric separation / isolation between criticality levels?
- How to provide asymmetric time separation between applications with different levels of criticality so that the impact of lower criticality applications on those of higher criticality can be tightly bounded independent of the behaviour or misbehaviour of the former, without significantly compromising guaranteed real-time performance?
- How to provide time composability for applications of different criticality levels, so that the timing behaviour of applications determined in isolation remains valid when they are composed during system integration?

The sessions of the seminar were structured around a set of themes. Particular attention was given to the interfaces between themes, as these are the areas that can benefit most from improved understanding and collaboration. The discussion groups were organized around the following themes that correspond to research challenges in mixed criticality systems (MCS):

- Platforms and Experimental Evaluation (see Section 5.1);
- Worst-Case Execution Time (see Section 5.2);
- Criticality (see Section 5.3);
- Probabilistic (see Section 5.4).

Organization of the Seminar

The seminar took place from 15th to 20th March 2015. The first day started with a keynote talk by Prof. Alan Burns (University of York), one of the most influential researchers in the Real-Time Systems field over the last 25 years. Alan reviewed advances in MCS research and underlined current open problems. An overview of his talk is provided in Section 3. The first day ended with presentations and feedback on real implementations (see Section 4) as well as identifying the main themes for group discussion.

The following three days started with presentations, which were followed by discussions either within the identified groups or in an open format.

The second day started with discussions about the motivation for mixed-criticality systems presented by three different participants (see Sections 4.4, 4.5 and 4.6). Different notations are used by different sub-communities and several presentations underlined these differences (see Sections 4.7, 4.8 and 4.9). An outline of the main ideas for probabilistic analysis of real-time systems provided the topics for the discussion group on probabilistic MCS (see Sections 4.10 and 4.11).

The morning of the third day commenced with discussions on the relation between time and MCS (see Section 4.11), which continued into the afternoon's hiking activity.

Starting from the fourth day a slot dedicated to anonymous mixed criticality supporters was added to the program allowing researchers new to the topic to identify open problems in MCS from the perspective of their different domains.

As detailed later in this report, the seminar enabled the real-time community to make important progress in articulating and reaching a common understanding on the key open problems in mixed criticality systems, as well as attracting new researchers to these open problems (see Section 6). The seminar also provided an ideal venue for commencing new collaborations, a number of which are progressing towards new research publications, see Section 7.

The seminar has helped define a research agenda for the coming years that could be supported by follow-up events, given the strong interest expressed by the participants of this seminar.

As organizers, we would like to thank Prof. Reinhard Wilhelm for encouraging us to submit the seminar proposal, Dagstuhl's Scientific Directorate for allowing us to run a seminar on mixed criticality systems, and to the staff at Schloss Dagstuhl for their superb support during the seminar itself. Finally, we would like to thank all of the participants for their strong interaction, presentations, group discussions, and work on open problems, sometimes into the early hours of the morning. We were very pleased to hear about the progress of new found collaborations, and to receive such positive feedback about the seminar itself. Thank you to everyone who participated for a most enjoyable and fruitful seminar.

2 Table of Contents

Executive Summary

Liliana Cucu-Grosjean, Robert I. Davis, Claire Maiza, and Sanjoy K. Baruah . . . 84

Keynote

Mixed Criticality – A Personal View

Alan Burns 89

Keynote addenda: An Augmented Model for Mixed Criticality

Alan Burns 92

Overview of Talks

Mixed Criticality in Multicore Automotive Embedded Systems

Sebastien Faucou 94

Efficiently Safe: Decoding the Dichotomy in Mixed-Criticality Systems

Arvind Easwaran 95

Adding Cache and Memory Management to the MC² (Mixed Criticality on Multicore) Framework

James H. Anderson 97

Mixed-criticality in Railway Systems: A Case Study on Signaling Application

A. Cohen, V. Perrelle, D. Potop-Butucaru, E. Soubiran, Z. Zhang 98

Confidence in Mixed-Criticality Multi-Core

Zoë Stephenson and Mark Pearce 103

Challenges in Mixed Criticality Systems Design – Integration Issues

Rolf Ernst 105

Real-time Performance Evaluation and VT Control mechanisms for the timing correct use of shared main memory

Kai Lampka 106

System-level, Inter-Criticality, Multi-Core Resource Sharing with Scalable Predictability

Gabriel Parmer 107

Mixed Criticality Support on Networks-on-Chip

Leandro Soares Indrusiak 110

Mapping criticalities to certification levels – a probabilistic attempt

Liliana Cucu-Grosjean and Adriana Gogonel 111

Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters

Dorin Maxim 113

Viewpoints on the Timing Aspect of Mixed Criticality Systems

David Broman 115

Mapping the landscape of mixed criticality systems research

Sanjoy K. Baruah 116

Some Open Problems in Mixed-Criticality Scheduling

Pontus Ekberg 118

Runtime monitoring of time-critical tasks in multi-core systems <i>Christine Rochange</i>	120
Timing Analysis for Multi/Many-core Platforms <i>Jan Reineke</i>	120
Analysis of pre-emptive systems with caches <i>Sebastian Altmeyer</i>	122
Using Mixed-Criticality to Reason about Temporal Correctness in Uncertain & Dynamic Environments <i>Nathan Fisher</i>	123
Augmenting Criticality-Monotonic Scheduling with Dynamic Processor Affinities <i>Bjoern B. Brandenburg</i>	124
Adaptive Uni-processor Fixed Priority Pre-emptive Probabilistic Mixed Criticality <i>Yasmina Abdedda</i>	126
MC Scheduling on Varying-Speed Processors <i>Zhishan Guo</i>	128
Speedup bounds for multiprocessor scheduling <i>Suzanne van der Ster</i>	130
Working Groups	
Report on Platforms and Experimental Evaluation <i>Robert I. Davis</i>	131
Report on WCET <i>Claire Maiza</i>	133
Report on Criticality <i>Sanjoy K. Baruah</i>	134
Report on Probabilistic Approaches <i>Liliana Cucu-Grosjean</i>	136
Open Problems	
Unification of mixed criticalities, WCET, and probabilistic execution time <i>Enrico Bini</i>	137
New collaborations	
Providing Weakly-Hard Guarantees for Mixed-Criticality Systems <i>Robert I. Davis and Sophie Quinton</i>	139
A Multicore Response Time Analysis Framework <i>S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke</i>	140
Mixed criticality support for automotive embedded systems <i>Yasmina Abdeddaim, Sébastien Faucou, and Emmanuel Grolleau</i>	141
Participants	142

3 Keynote

3.1 Mixed Criticality – A Personal View

Alan Burns (*University of York, GB*)

License © Creative Commons BY 3.0 Unported license
© Alan Burns

In this talk I want to address four topics:

- The notion of mixed criticality
- A overview of the literature on mixed criticality
- An augmented system model for mixed criticality
- Open Issues in mixed criticality research

The third of these topics is addressed in a separate abstract in Section 3.2. Notes on the other topics are provided below. As this is an extended abstract, derived from a talk, I will not include citations of the many works that have been published on Mixed Criticality. For accurate accrediting of the work alluded to below I refer readers to the Review from York (updated every 6 months and funded by the MCC project) available from: <http://www-users.cs.york.ac.uk/~burns/>.

It is important to be clear on the notion of ‘criticality’ as it is used in the, now extensive, literature on mixed critically. To me the notion is primarily based on the consequences and, to some extent, the likelihood of failure. A classification is therefore obtained by some form of hazard (or risk) analysis following a process usually defined in a Standard. All potential hazards much be mitigated during the design and implementation of both the hardware and software architectures. Software components, perhaps implemented within a run-time thread or task, will be assigned a criticality level (although different names are used for this classification in different Standards and application domains). If the late running of a task can contribute to a potential hazard then there must be evidence to support the view that such a deadline miss is sufficient unlikely. Such evidence will come from WCET analysis of the code and scheduling theory. It may also rely on run-time checks and enforcement.

The level of hazard, and the assignment of an assurance, integrity or criticality level will dictates the level of hardware redundancy and the procedures required in the design, verification and implementation of the code. There is considerable cost implications in (justifiable) begin able to reduce the classification of the software within a system.

To me, ‘mixed criticality’ is a means of dealing with the inherent uncertainty in a complex system. It is a means of providing efficient resource usage in the context of this uncertainty. It is also the means of protecting the more critical work when faults occur; including where assumptions are violated (rely conditions are false).

A mixed critically system is therefore not a mixture of hard and soft deadlines, nor is it a mixture of critical and non-critical components. Moreover it is not only concerned with delivering isolation and non-interference. And it is certainly not about dropping tasks to make a system schedulable. All of these ideas are, I believe, misconceptions about the nature of a mixed criticality system.

So if a mixed criticality approach is a means of dealing with uncertainly, where does this uncertainly come from? The primary source of uncertainly, as recognised in Vestal’s initial paper, comes from WCET estimation. We know that WCET cannot be known with certainty. All estimates have a probability of being wrong (too low). But all estimates are attempting to be safe (pessimistic). In particular $C(LO)$ is a valid engineered estimate with

the belief that $C(LO) > WCET$ ¹. Beliefs can be misplaced of course, which is why systems must be built to be resilient in the face of faults. But there must be a high level of confidence, perhaps expressed as a probability, that the assertion $C(LO) > WCET$ is true.

Other forms of uncertainty can come from the environment of the system. An event driven system must make assumptions about the intensity of the events it must deal with (in a timely fashion). Again this cannot be known with certainty. So the ‘Load’ parameters (however they may appear in the scheduling analysis) need to be estimated (safely). In particular, the minimum arrival interval (T) for a sporadic task (i.e. the assumed minimum interval between two events from the same source) as assumed at even the lowest criticality level of the system must be safe, that is $T(LO) < T(real)$.

Critical systems need to demonstrate survivability. Faults will occur and some level must be tolerated. One source of faults is that relate to the assumptions upon which the verification of the timing behaviour of the system was based eg. WCET, arrival rates, etc. A common notion in the fault tolerance literature is the idea of a fault model. Fault models provide a means of assessing/delivering survivability. For example:

- full functional behaviour with a certain level of faults;
- Graceful Degradation for more severe faults.

Graceful Degradation is a controlled reduction in functionality, aiming to preserve safety. So within the mixed criticality domain: if any task executes for more than $C(LO)$ and all HI-criticality tasks execute for no more than $C(HI)$ then it can be demonstrated that all HI-criticality tasks meet their deadlines.

As a strategy for Graceful Degradation a number of schemes in Mixed Criticality literature have been proposed:

- Drop all lower critical work
- Drop some, using notions of importance etc.
- Extend periods and deadlines (elastic task model)
- Reduce functionality within low and high criticality tasks

This strategy should perhaps be extended to issues concerning the $C(HI)$ bound also being wrong!

If tasks are dropped/aborted then this cannot be arbitrary – the approach must be related back to the software architecture and task dependencies. So if task A is closely coupled to task B then either drop both or neither. Recovery must also relate to the needs of the software (e.g. dealing with missing/stale state).

What I want to emphasize with the above discussion is that the dropping of functions can never be seen as part of the normal behaviour of the system. That would not be acceptable to any system’s developer. Rather it is a means of protecting the most critically functions during a system overload, which itself is due to a fault, with may occur due to the inherent uncertainty in the system’s behaviour and environment.

Another issue that arises in the mixed criticality literature is the use of a ‘criticality mode’ to capture the behaviour of the system when all functions are being delivered, and other modes that relate to reduced functionality. We have tended to call these modes (in a dual-criticality system) *LO*-criticality mode when all is well, and *HI*-criticality mode when only *HI*-criticality functions are guaranteed. This terminology is I feel misleading, the normal behaviour of the system when all functions are timely should be called ‘normal’.

¹ Many papers on mixed criticality assume two criticality levels, *LO* and *HI*, and two estimates of WCET related to these two levels: $C(LO)$ and $C(HI)$, with $C(LO) \leq C(HI)$.

After a fault, and degraded functionality it should be possible for the system to return to full functionality (i.e. normal mode). After all, a 747 can fly with 3 engines, but its nice to get the 4th one back. Fault recovery is therefore also an issue for mixed criticality behaviour. Indeed, as I have tried to explain, much of the work on mixed criticality systems need to be more cognisant of the available literature on fault tolerance.

Since Vestal's paper there has been at least 180 articles published (one every 2 weeks!).

Some top level observations follow. For uniprocessors:

- For FPS, AMC seems to be the 'standard' approach.
- For EDF, schemes that have a virtual deadline for the *HI*-criticality tasks seem to be standard.
- Server based schemes have been revisited.
- Not too much work on the scheduling schemes actually used in safety-critical systems, e.g. cyclic executives and non-preemptive (or cooperative) FPS.

For multiprocessor systems there are a number of schemes (extensions from uni-criticality systems). Similarly for resource sharing protocols. Work on communications is however less well represented. As indicated above, there is lots of work on graceful degradation (although few papers use that term).

Almost all papers stick to just two criticality levels. But remember *LO*-criticality does not mean no-criticality! Some papers pay lip service to multiple levels, but not many. It is still not clear what is the model we require for, say, 4 or 5 levels? To me it does not seem to make sense to have five estimates of WCET.

Notwithstanding the obvious synergy with fault tolerance there is actually little work on linking mixed criticality to fault tolerance in general. There is also little work on probabilistic assessment of uncertainty. There is some implementation work, but arguable not enough. Similarly, there is some comparative evaluations, but again not enough. There is however good coverage of formal issues such as speed-up factors.

I will finish by recording the open issues that I have identified from reading the extensive mixed criticality literature.

1. As well as looking at mixing criticality levels within a single scheduling scheme (e.g. different priorities within FPS) we need to look at integrating different schemes (e.g. Cyclic Executives for safety-critical, FPS for mission critical on the same processor).
2. More work is needed to integrate the run-time behaviour (monitoring and control) with the assumptions made during static verification.
3. We need to be more holistic in terms of ALL system resources (especially communications media).
4. There are a number of formal aspects of scheduling still to be investigated.
5. We need to be sure that techniques scale to at least 5 levels of criticality.
6. There are still a number of open issues with regard to graceful degradation and fault recovery.
7. There is little work as yet on security as an aspect of criticality.
8. We need protocols for information sharing between criticality levels.
9. We need better WCET analysis to reduce the (safe) $C(HI)$ and $C(LO)$ values (or at least improve our confidence in the numbers used).
10. We should look to have an impact on the Standards relevant to the application domains we hope to influence.
11. Better models for system overheads and task dependencies are needed.
12. How many criticality levels to support (and how many estimates of the sources of uncertainty to accommodate)?
13. We do not as yet have the structures (models, methods, protocols, analysis etc) that allow tradeoffs between sharing and separation to be evaluated.

To conclude, the Dagstuhl seminar is both timely and necessary in moving our research forward.

3.2 Keynote addenda: An Augmented Model for Mixed Criticality

Alan Burns (University of York, GB)

License  Creative Commons BY 3.0 Unported license
© Alan Burns

Inevitably not all papers on mixed criticality have used the same system or task model. But following on from the initial paper of Vestal [1] the most common form of the mixed criticality model is one that has a small number of criticality levels and that for each level tasks have an assigned estimation of worst-case execution time (WCET), C . Early publications on mixed criticality have often further restricted the model to have just two criticality levels, HI and LO , and therefore only two computation parameters $C(HI)$ and $C(LO)$. As we move back to using four or five criticality levels then the question arises – do we really have this number of ways of estimating WCET?

A criticality level determines many aspects of how a software function, embedded in a run-time task, is to be produced and verified. But it does not follow that distinct means of estimating or measuring execution time are available at each criticality level. In this short note we argue that two estimates are sufficient for a suitably expressive model to be defined.

Assume that the application domain of the defined system has four levels of criticality: A, B, C, D (with A being the highest level) and one non-critical level E. Code in E, if it exists, will have no or soft deadlines and not be crucial for any function of the system. Nevertheless, it may include house-keeping functions that are useful and should be executed if possible.

Level D is the lowest criticality level. We term this the *normal* mode of the system in that during normal, fault-free, execution all code from all four criticality levels are guaranteed to meet defined timing constraints. To validate normal behaviour it is necessary for all critical code to have an estimate of its WCET that is appropriate for level D criticality. We call these estimates $C(normal)$ – in existing literature this would be called $C(D)$.

As code of a particular criticality level has to be produced and verified to the standard dictated by the assigned level, there must be an estimate of WCET that is linked to that criticality level. So level A has a $C(A)$ estimate, level B a $C(B)$ estimate etc. In general, we can say that all critical code has an estimate commensurate with its own criticality, we term this $C(self)$.

To summarise, all tasks have two estimates of WCET: $C(self)$ and $C(normal)$, with $C(self) \geq C(normal)$. For tasks of the lowest level of criticality (level D in our framework), these two estimates are the same.

The run-time behaviour of our system, following the basic idea of Vestal, is as follows:

- If all critical tasks execute for no more than their $C(normal)$ values then all critical deadlines are met.
- All tasks are prevented, by run-time monitoring, to execute for more than $C(self)$.
- If any task of criticality level X executes for more than $C(normal)$ then all tasks of critically level X and higher must continue to meet their deadlines, using estimates of WCET of $C(self)$ for tasks of criticality X and $C(normal)$ for tasks of criticality higher than X.

- In the above scenario, tasks of criticality levels below X are no longer guaranteed (and may be subject to forms of graceful degradation necessary to ensure the continuing correct execution of levels X and higher).

Note that this is a different behaviour from the one used by AMC [2], for example. In that protocol the second case would use estimates of $C(X)$ for the higher criticality tasks (not $C(normal)$). Of course the augmented model presented here can be directly expressed in the original Vestal model if all intermediate estimates of WCET between $C(normal)$ and $C(self)$ are assigned the same value as $C(normal)$.

As well as simplifying the model, the above behaviour is supported by at least some industrial practice. Code of level A is likely to be subject to coding standards that restrict the expressive power of the programming language employed. For example, recursion may be prohibited as may the arbitrary use of pointers and ‘while’ loops. It follows that level A code is more predictable and less likely to execute for more than $C(normal)$.

This augmented model is motivated by the fact that software development processes are unlikely to deliver more than two estimates of WCET. However, this does not mean that run-time behaviour cannot use computed estimates that facilitate more fine-grain control over graceful degradation. For example, if a task of level A executed for more than $C(self)$ (i.e. $C(A)$) then the above model will allow all tasks of all lower criticality levels to be abandoned (to ensure level A work is preserved). However a Real-Time Systems engineer could quite reasonably argue that this reaction is overly conservative. It would be quite straightforward to use the scheduling analysis to compute a value of $C(C)$, with $C(self) > C(C) > C(normal)$, and enforce the run-time behaviour: if a level A task executes for more than $C(normal)$ but no higher than $C(C)$ then only tasks of level D need to degrade. Sensitivity analysis for fixed priority scheduling has already been used [3] to solve a related problem.

This useful step, of computing intermediate WCET estimates, does not however detract from the application model being advocated here. This model restricted the number of external/given estimates of WCET to two.

Vestal [1], and much follow on work, has focused on WCET as the main source of uncertainty in the model of the system. Other forms of uncertainty exist – including load from the environment, faults in the hardware, power from (perhaps unreliable) sources etc. For all of these sources of uncertainty we argue that there should be two estimates. One for the normal all inclusive criticality of the system and one that reflects the particular criticality of the component.

References

- 1 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
- 2 S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE RTSS*, 2011, pages 34–43.
- 3 T. Fleming and A. Burns. Incorporating the notion of importance into mixed criticality systems. In L. Cucu-Grosjean and R. I. Davis, editors, *Proc. 2nd Workshop on Mixed Criticality Systems (WMC)*, *RTSS*, 2014, pages 33–38.

4 Overview of Talks

Mixed-criticality needs feedback from real implementation

4.1 Mixed Criticality in Multicore Automotive Embedded Systems

Sebastien Faucou (University of Nantes, FR)

sebastien.faucou@univ-nantes.fr

License  Creative Commons BY 3.0 Unported license
© Sebastien Faucou

Introduction

The automotive industry pursues an effort toward the standardization of in-vehicle embedded systems technologies. If we focus on the topics of interest of this seminar, two standards stands out: ISO 26262, the functional safety standard; and AUTOSAR OS, the RTOS component of the standardized AUTOSAR architecture. Studying these standards gives some insight on the way mixed criticality is handled today in automotive embedded systems and allows to identify direction for future works.

ISO 26262: ASIL and freedom from interference

ISO 26262 defines two key concepts. The first one is the risk classification scheme composed of four *ASILs* (Automotive Safety Integrity Level) that range from A (the least critical) to D (the most critical). ASILs are attached to hazardous events and mapped to software components as a result of the hazard analysis. Mixed criticality requirements arise when the software components of a system have different ASILs.

The second key concept of ISO 26262 is *freedom from interference*. Freedom from interference is established when no error can propagate from low-criticality components to high criticality components. Freedom from interference does not implies full isolation but rather that interferences between criticality classes are bounded. It encompasses functional and extra-functional concerns, including timeliness and communication. In a system built on top of a shared platform, if freedom from interference can not be proved, then every component shall be designed with the requirements associated with the highest ASIL among all the co-hosted components.

AUTOSAR OS

AUTOSAR OS extends OSEK/VDX OS with several features, including *protection facilities* and support for multicore platforms. Among the protection facilities, timing protection monitors the run-time behaviour of the jobs, assuming a sporadic model. This allows to use for instance the schedulability tests developped for Vestal's model [1] and may be for some of its extensions [2] in order to validate the capacity of the system to survive to timing faults and preserve the timeliness of its most critical functions, *ie.* establishing freedom from interference in the time domain.

Protection facilities also include the possibility to partition the memory and the peripherals of the platforms between *OS-Applications* (set of tasks, interrupt handlers and shared resources) and to enforce this partitionning at run-time. These features contribute to freedom from interference in the communication domain

MC in multicore automotive embedded systems

The two parts of AUTOSAR OS protection facilities presented above are useful but with the advent of multicore platforms, this is not sufficient. Indeed, these mechanisms do not address the management of shared hardware resources in a mixed-criticality context. Examples of shared hardware resources include the memory bus, the SRAM banks, the shared cache level found in high-end microcontroller for infotainment. These shared resources are channels that allow low criticality tasks to interfere on the execution of highest criticality ones.

Two directions for future works

Vestal's model and its extensions offer a solid theory for real-time scheduling of mixed criticality systems. Some works have been carried on in order to evaluate the pertinence of this theory in the context of Linux-based real-time systems [3]. The same type of works remains to be done in the context of smaller (embedded) real-time systems, taking into account and exploiting some distinctive features such as: limited hardware resources, static software, sub-millisecond deadlines, etc.

According to the current state of the art, the second direction that should be considered a priority is the design of methods to bound interferences in multicore systems. Once again, the distinctive features of automotive embedded systems should be exploited to propose low footprint mechanisms, amenable to static analysis.

References

- 1 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
- 2 S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE RTSS*, 2011, pages 34–43.
- 3 Huang-Ming Huang, Christopher D. Gill, and Chenyang Lu. Implementation and evaluation of mixed-criticality scheduling approaches for periodic tasks. In *RTAS*, pages 23–32, 2012.

4.2 Efficiently Safe: Decoding the Dichotomy in Mixed-Criticality Systems

Arvind Easwaran (Nanyang Technological University, SG)
arvinde@ntu.edu.sg

License © Creative Commons BY 3.0 Unported license
© Arvind Easwaran

An increasing trend in embedded systems is towards open computing environments, where multiple functionalities are developed independently and integrated together on a single computing platform. This trend is evident in industry-driven initiatives such as ARINC653 Integrated Modular Avionics (IMA) in avionics and AUTOSAR in automotive. An important notion behind this trend is the safe partitioning of separate functionalities, primarily to achieve fault containment. This raises the challenge of how to balance the conflicting requirements of partitioning for safety assurance and efficient resource sharing for economical benefits. The concept of *mixed-criticality*, first introduced by Vestal [1], appears to be important in meeting these dichotomous goals.

In many safety-critical systems, the correct behavior of some functionality (e.g., flight control) is more important (“critical”) to the overall safety of the system than that of another (e.g., in-flight entertainment). In order to certify such systems as being correct, they are conventionally assessed under certain assumptions on the worst-case run-time behavior. For example, the estimation of Worst-Case Execution Times (WCETs) of code for highly critical functionalities involves very conservative assumptions that are unlikely to occur in practice. Such assumptions make sure that the resources reserved for critical functionalities are always sufficient. Thus, the system can be designed to be fully safe from a certification perspective, but the resources are in fact severely under-utilized in practice.

In order to close such a gap in resource utilization, Vestal [1] proposed the mixed-criticality task model that comprises of different WCET values. These different values are determined at different levels of confidence (“criticality”), based on the following principle. A reasonable low-confidence WCET estimate, even if it is based on measurements, may be sufficient for almost all possible execution scenarios in practice. In the highly unlikely event that this estimate is violated, as long as the scheduling mechanism can ensure deadline satisfaction for highly critical applications, the resulting system design may still be considered as safe.

To ensure deadline satisfaction of critical applications, mixed-criticality studies make pessimistic assumptions when a single high-criticality task executes beyond its expected (low-confidence) WCET. They assume that the system will either immediately ignore all the low-criticality tasks (e.g., [2, 3, 4]) or degrade the service offered to them (e.g., [5, 6, 7]). They further assume that all the high-criticality tasks in the system can thereafter request for additional resources, up to their pessimistic (high-confidence) WCET estimates. Although these strategies ensure safe execution of critical applications, they have a serious drawback as pointed out in a recent article [5]. When a high-criticality task exceeds its expected WCET, the likelihood that all the other high-criticality tasks in the system will also require more resources is very low in practice. Therefore, to penalize all the low-criticality tasks in the event that some high-criticality tasks require additional resources seems unreasonable.

In practice, most mixed-criticality systems are comprised of independently developed components. For wide applicability, it is then natural that mixed-criticality strategies must consider the impact of WCET violations across component boundaries. To the extent possible, these strategies must limit this impact to within components, so that other components in the system (high- as well as low-criticality ones) can continue their execution uninterrupted. Considering the fact that different approaches may be used to compute the WCET estimates within different components, we believe this is a reasonable requirement because otherwise components can be unfairly penalized due to ill-computed WCET estimates of other components. One extreme solution that addresses this requirement is the worst-case reservation-based approach that completely isolates components but severely under-utilizes the resources. On the other hand, most of the recent mixed-criticality studies such as those mentioned above, completely ignore these component boundaries but still under-utilize resources due to unrealistic assumptions.

Challenges

Based on the above discussions, we now summarize some of the main challenges in designing an “*efficiently safe*” mixed-criticality system.

1. What is a good scheduling and execution strategy that can use component boundaries to provide partitioning between functionalities, but at the same time is resource-efficient and adequately supports low-criticality tasks? Some initial results in this direction are presented in a recent article [8].

2. Is there any motivation to provide hierarchical scheduling for component-based mixed-criticality systems? Since mixed-criticality scheduling strategies naturally isolate the critical tasks from non-critical ones, can we meet the partitioning requirements using a non-hierarchical scheduling framework?
3. The failure of existing studies to understand the implication and feasibility of abruptly stopping/modifying the active low-criticality tasks is another important shortcoming that has been highlighted [5]. Can we also address this issue by limiting the impact of WCET violations to within components?

References

- 1 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
- 2 S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE RTSS*, 2011, pages 34–43.
- 3 S. Baruah, V. Bonifaci, G. D’Angelo, H. Li, and A. Marchetti-Spaccamela. The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems. In *ECRTS*, 2012.
- 4 A. Easwaran. Demand-based Scheduling of Mixed-Criticality Sporadic Tasks on One Processor. In *RTSS*, 2013.
- 5 A. Burns, S. Baruah, K. M. Phan, and I. Shin Towards a more practical model for mixed criticality systems In *WMC*, 2013.
- 6 H. Su and D. Zhu An elastic mixed-criticality task model and its scheduling algorithm. In *DATE*, 2013.
- 7 P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptations for mixed-criticality systems. In *ASP-DAC 2014*. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC).
- 8 X. Gu, A. Easwaran, K. M. Phan, and I. Shin Resource Efficient Isolation Mechanisms for Mixed-Criticality Systems (Technical Report: Nanyang Technological University). <http://ntu.edu.sg/home/arvinde/preprints/ECRTS15.pdf> In *ECRTS*, 2015.

4.3 Adding Cache and Memory Management to the MC² (Mixed Criticality on Multicore) Framework

James H. Anderson (The University of North Carolina at Chapel Hill, US)
anderson@cs.unc.edu

License © Creative Commons BY 3.0 Unported license
© James H. Anderson

Keywords: cache coloring, set partitioning, way partitioning, memory banks, multicore

The multicore revolution is having limited impact in safety-critical application domains. The key reason is the “one out of m ” problem: when checking real-time constraints on a platform with m cores, analysis pessimism can easily negate the processing capacity of the “additional” $m - 1$ cores. Two major approaches have been investigated previously to address this problem: mixed-criticality allocation strategies that seek to provision less critical software components less pessimistically, and hardware management strategies that seek to make the underlying platform itself more predictable. While both approaches seem somewhat promising, neither by itself has proven capable of practically resolving the “one out of m ” problem. In this talk,

the results of an ongoing development effort will be discussed in which both approaches are being applied together. This effort is based on a new variant of the MC² (mixed-criticality on multi-core) [1, 2, 3] framework that enables tasks to be isolated by criticality level with respect to the hardware resources they access. Experimental results will be presented that demonstrate the efficacy of the overall framework (if such results are available by the time the workshop is held).

References

- 1 J. Herman, C. Kenna, M. Mollison, J. Anderson, and D. Johnson. RTOS support for multicore mixed-criticality systems. In *RTAS*, 2012.
- 2 M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos. Mixed criticality real-time scheduling for multicore systems. In *ICESS*, 2010.
- 3 B. Ward, J. Herman, C. Kenna, and J. Anderson. Making shared caches more predictable on multicore platforms. In *ECRTS*, 2013.

... and the industry has created the need for the mixed-criticality

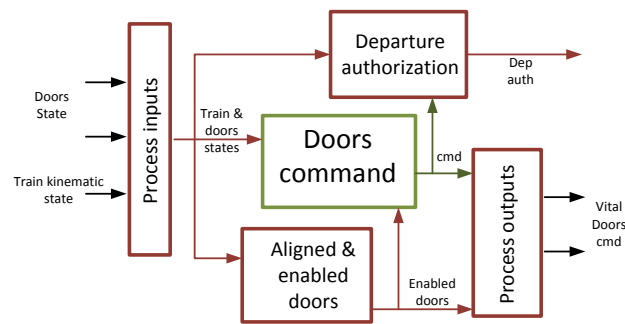
4.4 Mixed-criticality in Railway Systems: A Case Study on Signaling Application

A. Cohen (INRIA, FR), V. Perrelle (Technological Research Institute SystemX, FR), D. Potop-Butucaru (INRIA, FR), E. Soubiran (Alstom Transport, FR), Z. Zhang (INRIA & Technological Research Institute SystemX, FR)
 albert.cohen@inria.fr

License © Creative Commons BY 3.0 Unported license
 © A. Cohen, V. Perrelle, D. Potop-Butucaru, E. Soubiran, Z. Zhang

Since the early 2000's almost every new metro project in the world make use of a standardized railway signalling system called Communication Based Train Control (CBTC) (IEEE 1474). Previously to CBTC, conventional signalling train control systems were relying almost exclusively on track circuits, wayside signals and operating procedures to ensure train protection and operation. In order to ensure better operational performance (e.g. effective utilization of the transit infrastructure), CBTC systems rest on three pillars: “*Automatic train control (ATC) based on high-resolution train location determination, independent of track circuits*”; “*high-capacity and bidirectional train-to-wayside data communications*”; and “*train-borne and wayside computing units that execute vital functions*”. Functions are classified within three families that are: Automatic Train Protection (ATP), Automatic Train Operation (ATO) and Automatic Train Supervision (ATS). The level of criticality differs from a family to another and without loss of generality, one can state that ATP functions are mostly safety critical functions (SIL4 regarding to CENELEC 50126), whereas ATO and ATS gather functions of low criticality (from SIL0 to SIL2). As a matter of fact, CBTC systems are in essence Mixed-critical systems. Furthermore the mainstream evolution of those systems tends toward more functional integration on more powerful computing units. ATP and ATO functions that were traditionally distributed on different computing units (both on wayside and train-borne) tends now to be deployed on the same computing units and thus sharing resources.

FSF (Safe and reliable embedded system) is an IRT SystemX project positioned on two topics, the first one is about the conception of signalling applications (typically ATO/ATP



■ **Figure 1** Simplified view of a mixed-critical path in the Passenger Exchange component.

application) that contain both critical and non-critical parts and the second one is on execution platforms that execute those applications while offering high guarantee of safety and availability. Industrial expectations around the execution platform include the use of multi-core COTS, the use of modern RTOS that offer spatial and temporal isolation, the use of safety and availability architectural patterns (e.g. voting and redundancy), and the whole being finally hidden behind a “system abstraction layer”. On top of this platform, a tool framework is prototyped and allow one to develop, verify and deploy component based applications where components may arbitrary contains both vital and non-vital code. The project has started in May 2013, the aim of this communication is to propose a first return of experience and a positioning on how MICS will be addressed in FSF.

Alstom Transport has defined an applicative case study that, while being limited to one single ATC function, is representative of the complexity in term of vital/non-vital code interweaving, operational performance and availability. The system function is called “passenger exchange”. This function takes control on the train when this one is safely docked at a station; it organizes the exchange of passengers (train and station doors opening/closing) while protecting them from any untimely train movement or non-aligned doors opening and finally gives the departure authorization when all safety conditions are met. The functional specification is made of more than 300 requirements (natural language + SysML), and the functional architecture is made of about twenty sub functions.

PE is designed as a system component with a vital and a non-vital part. At this level a component is roughly a packaging unit that exposes to the exterior world a set of ports (in or out) and that is characterized by a set of behaviours that depend on the operational context. One shall notice that there are no restrictive design constraints on dataflow dependency between the vital and non-vital part. This component is then implemented as a set of software components which are this time exclusively vital or non-vital.

The vital and the non-vital parts need to communicate together. To illustrate this fact, we give an example from the case study. Let’s take two constraints from the vital requirements. The first one states that the component shall not transmit a departure authorization when the doors are open or opening. It obviously implies that the component isn’t sending any command to open the doors. The second requirements state that the system shall not send commands to doors which are not safe to open. (e.g. because they are not aligned) To meet these requirements, a vital subcomponent compute which doors shall be enabled. This information is given to a non-vital component which compute which commands to send and when to send them to achieve the assigned mission. Since this last component is not bound by the same safety constraints, we can’t give the same confidence to its output. Hence, we

need to process these output with a vital component. The door commands are matched against the enable set of doors initially computed and truncated if necessary to fulfil the requirements. Finally, another vital component reads the door state and the commands which have just been computed to decide whether to give a departure authorization or not.

This example shows that in our case study, there can be numerous communications between small components of mixed criticality.

Synchronous approaches

Synchronous languages

Data-flow synchronous languages, such as LUSTRE [1] or SIGNAL [2] have been designed in the 80's for program real-time safety critical embedded systems. Since then, they have been widely used in industrial applications [3]. These languages emphasise a correct-by-construction approach, ensuring bounded memory and execution time. Moreover, they are praised for their predictable behaviour and formally defined semantics.

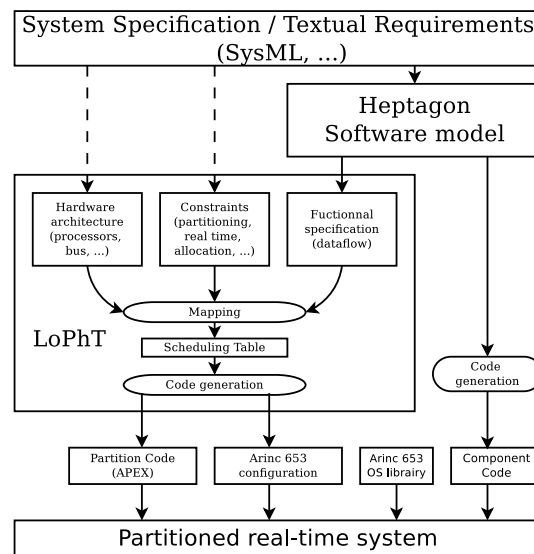
Recently, the problem of scheduling multi-rate, mixed-critical synchronous programs have been addressed. At first for uni-processor [6] then for multi-processors [14]. Outside the scope of mixed-criticality there were also several attempts to distribute synchronous data-flow languages [4, 5]. Recent work have been done to develop these languages to target multi-core platforms through the programming of parallelism [12]. This work introduces futures in LUSTRE-like languages giving the guarantee that the sequential semantics is preserved.

Automatic allocation, partitioning, and scheduling

Due to their use in the avionics industry, synchronous languages have been considered early on as an input formalism for the automatic or semi-automatic synthesis of real-time implementations. Most significant in this direction are previous results by previous work by Sorel *et al.* [7] on the AAA/SynDEX methodology and tool for distributed, but not time-triggered, real-time implementation of multi-periodic synchronous specifications, previous work by Caspi *et al.* on the use of Lustre/Scade in the real-time implementation of Simulink over multi-processor platforms based on the time-triggered partitioned bus TTA [8], and previous work by Forget *et al.* [9] on the specification and implementation of multi-periodic applications over a time-triggered platform using the Prelude language.

But none of these approaches allow us to take into account all the characteristics of our case study in order to allow automatic mapping. In particular, none of them has support for ensuring the time and space separation between application parts with different *criticalities*.

This is why we considered in this project a new tool, named LoPhT [11, 10], which allows the automatic mapping of applications onto platforms following the ARINC 653 time and space partitioning mechanisms. The LoPhT tool has the flow pictured in Fig. 2. It takes as input deterministic functional specifications provided by means of synchronous data-flow models with multiple modes and multiple relative periods. These specifications are extended to include a real-time characterization defining task periods, release dates, and deadlines. Task deadlines can be longer than the period to allow a faithful representation of complex end-to-end flow requirements. The specifications are also extended with allocation constraints and partitioning information meant to represent the criticality of the various tasks, as well as information on the preemptability of the various tasks. Starting from such specifications, the LoPhT tool performs a fully automatic allocation and off-line scheduling onto partitioned time-triggered architectures. Allocation of time slots/windows to partitions can be fully or partially provided, or synthesized by LoPhT. The mapping algorithms of LoPhT take into



■ **Figure 2** The design flow.

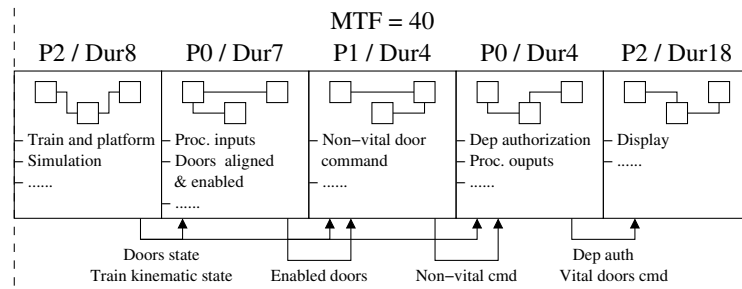
account the communication costs. The off-line mapping algorithms of LoPhT use advanced mapping techniques such as software pipelining and pre-computed preemption to improve schedulability and minimize the number of context switches.

Case study

The case study PE has been implemented and a first demonstrator has been produced. The challenge for this first demonstrator was to propose a framework for on the one hand the design and implementation of components and on the other hand the design of signalling application its partitioning and scheduling.

Choice of software modelling language. We chose to use the language HEPTAGON, very similar to LUSTRE and featuring novel constructions and novel optimisations. Two criteria have influenced the choice of the language. First, the functional specification defined at system level and allocated to software components have been written in a reactive and mostly equational way. It was thus very natural to implement it in a synchronous data-flow language. Second, the normative referential (CENELEC 50128) recommends the use of formal methods for the development of critical software while making no restrictive assumption on the language used for the non critical part. Synchronous languages are a good trade-off since they enable the use of formal methods (for instance model checking or abstract interpretation) while providing a sufficient power of expression to implement non-critical components. Finally, having a single language to develop both critical and non-critical components allows not only the early simulation of functional behaviour without integration effort but also the rationalisation of competence in the software development team.

Scheduling and partitioning with LoPhT. Entering in the flow of the LoPhT tool, detailed above, requires the definition of its input specification. For the functional specification part, direct translation is possible from Scade/Heptagon to the input formalism of LoPhT (which is also a data-flow synchronous language).



■ **Figure 3** The partitional scheduling result of LOPHT.

Technical realisation. We developed the Passenger exchange components following a five step process:

1. In a SysML environment, we produced a component design that realize the Passenger exchange function. System requirements are traced and refined to define atomic components that correspond to software components and that are either safety-critical, mission-critical or non-critical.
2. We matched every atomic component to a HEPTAGON node realizing the functional behaviour.
3. Depending on the SIL of the component verification activities have been led but are out of the scope of this communication.
4. Thanks to a dataflow model we have produced a small signalling application that gathers several components including Passenger exchange, Train/Station interfaces and a simulation of other system functions (train driving...). From HEPTAGON point of view the application is trivially a node assembly. At this stage, a first executable code is produced to simulate the application behaviour, however no insurance is given on spatial isolation.
5. In LOPHT, the application functions are decomposed into three partitions, which are “P0: critical”, “P1: non-critical” and “P2: environment”. Meanwhile the function durations are given (we suppose that each function takes one time scale unit). The scheduling result is presented in the Figure 3. Five windows are created. The first one has eight functions of the “environment” partition. In this window, the states of the doors and the train kinematic are analyzed and finally sent to the corresponding windows. The second window containing seven functions of the “critical” partition, which decide the critical control commands. The third window is composed of four functions of the “non-critical” partition. In this window, the non-critical control commands are generated. The fourth window does the remaining critical works and the last window gives the feedbacks to the “environment” component, such as a display screen. The code generated by LOPHT is simulated and tested on the POK OS [13].

To interface the software model with LOPHT, we needed to add a bit of glue code. Each HEPTAGON node representing an atomic function has been wrapped with a static memory. This wrapper exposes the two `reset` and `step` functions needed by the synchronous paradigm. We extended the C backend of HEPTAGON to be able to generate these wrappers automatically.

References

- 1 P. Caspi, D. Pilaud, N. Halbwachs and J.A. Plaice. LUSTRE: A Declarative Language for Real-time Programming. *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM*, 178–188, 1987.

- 2 A. Benveniste, P. Le Guernic and C. Jacquemot. Synchronous Programming with Events and Relations: The SIGNAL Language and Its Semantics. *Sci. Comput. Program., Elsevier North-Holland, Inc.*, 16, 103–149 1991.
- 3 A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. L. Guernic, L. Robert and D. Simone. The synchronous languages 12 years later. *Proceedings of The IEEE*, 64–83, 2003.
- 4 P. Aubry and P. Le Guernic. On the desynchronization of synchronous applications. *11th International Conference on Systems Engineering, ICSE*, 96, 1996.
- 5 P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis and P. Niebert. From Simulink to SCADE/Lustre to TTA: A Layered Approach for Distributed Embedded Applications. *Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool for Embedded Systems, ACM*, 2003.
- 6 S. Baruah. Semantics-preserving Implementation of Multirate Mixed-criticality Synchronous Programs. *Proceedings of the 20th International Conference on Real-Time and Network Systems, ACM*, 11–19, 2012.
- 7 M. Marouf, L. George and Y. Sorel. Schedulability analysis for a combination of non-preemptive strict periodic tasks and preemptive sporadic tasks. *Proceedings ETFA*, 2012.
- 8 P. Caspi, A. Curic, A. Magnan, C. Sofronis, S. Tripakis and P. Niebert. From Simulink to SCADE/Lustre to TTA: a Layered Approach for Distributed Embedded Applications. *Proceedings LCTES*, 2003.
- 9 C. Pagetti, J. Forget, F. Boniol, M. Cordovilla and D. Lesens. Multi-task Implementation of Multi-periodic Synchronous Programs. *Discrete Event Dynamic Systems*, 21, 307–338, 2011.
- 10 T. Carle and D. Potop-Butucaru. Predicate-aware, Makespan-preserving Software Pipelining of Scheduling Tables. *ACM Trans. Archit. Code Optim*, 11, 12:1–12:26, 2014.
- 11 T. Carle, D. Potop-Butucaru and Y. Sorel and D. Lesens. From dataflow specification to multiprocessor partitioned time-triggered real-time implementation. *INRIA*, 2012.
- 12 A. Cohen, L. Gérard and M. Pouzet. Programming Parallelism with Futures in Lustre. *Proceedings of the Tenth ACM International Conference on Embedded Software, ACM*, 197–206, 2012.
- 13 J. Delange, L. Pautet, and P. Feiler. Validating Safety and Security Requirements for Partitioned Architectures. *Proceedings of the 14th Ada-Europe International Conference on Reliable Software Technologies, Springer-Verlag*, 30–43, 2009.
- 14 E. Yip, M. Kuo, D. Broman, and P. S. Roop. Relaxing the Synchronous Approach for Mixed-Criticality Systems. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, pages 89–100. IEEE, 2014.

4.5 Confidence in Mixed-Criticality Multi-Core

Zoë Stephenson and Mark Pearce (Rapita Systems Ltd., UK)

{zstephenson,mpearce}@rapitasystems.com

License © Creative Commons BY 3.0 Unported license
© Zoë Stephenson and Mark Pearce

Keywords: WCET, assurance, assurance deficit, argument, partitioning

For aerospace applications, CAST-32 [1] indicates that applicants need to show both that applications running on a multi-core processor have the desired behaviour, and that the characteristics of the computing platform are understood and controlled.

We believe that our process for measurement-based worst-case execution time (WCET) estimation can be extended to account for mixed-criticality systems, many-core systems and measurement-based testing for characteristics other than timing.

Hypothesis-driven Analysis

As an example, our method for exploring execution time is to perform standard functional testing and *measure* end-to-end execution times, *hypothesise* that the longest measured time is the worst that can occur, *search* for evidence to contradict this, and *repeat* the exercise until further challenge to the hypothesis is no longer practicable. We use RapiTime for on-target measurements and WCET path prediction as part of this process, but the method is not restricted to a specific tool. Any lingering aspects that cannot be addressed through this approach need to be accounted for with additional margins and protection mechanisms (CAST-32 calls these “safety nets”).

Meeting CAST-32

CAST-32 contains many recommendations asking the applicant to demonstrate understanding of the target. It is explicit in MCP_Software_2 that the verification environment should be representative of the final intended hardware environment. By including iterative testing on the target as part of the analysis process, we ensure that this is the case. We can then use the guidance of CAST-32 to drive the search for counter-evidence challenging the execution time hypothesis, which provides insight into the behaviour of the processor.

Beyond Execution Time

CAST-32 is concerned with other effects that can occur because of the multi-core platform – delays in access to resources (data, devices, locks...), denial of access, out-of-order accesses or incorrect accesses. These are compatible with the hypothesis-driven approach. For example, a hypothesis that accesses are always in order may be established, and then testing improved to try to cause out-of-order behaviour. In the eventuality that the erroneous behaviour can be triggered, the testing process itself tells the analyst what to recommend to avoid triggering the behaviour.

Extension to Mixed Criticality

Mixing criticalities implies some protection between those different criticalities. This may be seen as another type of on-target constraint that the analyst can measure and challenge. It is likely that this will be needed as multi- and many-core systems become more prevalent. We expect that additional support may be needed from suppliers to be able to provide testing to show that protection works for a specific application in a specific set of configurations in a specific test environment.

Beyond Two Cores

The strategy in CAST-32 is to test what will run on each core individually, and then test them together. When viewed from the perspective of hypothesis-and-challenge testing, we suggest that it may be useful to test individual cores first in combination with a range of “test” behaviours on other cores, to try to undermine the application on the core under test with expected and unexpected use of shared resources.

Challenges

The recommendations of CAST-32 ask the applicant to explain what has been done to understand and control the target behaviour. This is a complex task, with evidence both about the behaviour of the software on target and about the process of exploring the unknown behaviour of that target. We advise using a structured argument to present this explanation as an assurance case.

Regardless of the approach taken, a significant challenge is to measure the behaviour on a multi-core system without causing further interference. To this end, we advise engagement to refine existing debug and trace capabilities so that interference is minimal and bounded, and ideally entirely non-existent.


References

- 1 CAST. *Multi-core Processors*. Position Paper CAST-32, May 2014

4.6 Challenges in Mixed Criticality Systems Design – Integration Issues

Rolf Ernst (TU Braunschweig, DE)

ernst@ida.ing.tu-bs.de

License  Creative Commons BY 3.0 Unported license
© Rolf Ernst

Current industrial developments lead to a growing number of tasks with different safety criticalities sharing the same components of an embedded system. At the same time, high performance is becoming more important. Prominent examples are the automotive and avionics domains. In the talk, we explain the complex side effects of switched Ethernet for automotive applications which make mixed critical designs hard. A main challenge arises from the many dependencies between the numerous layers of an architecture that are typically not overseen by a single person in the design process. We propose applying dependency analysis to identify possible hidden effects between function executions and between components. We conclude that mixed criticality are as complex as the underlying architectures and mechanisms. Solutions to individual problems, such as scheduling, are not sufficient, because safety (like security) is dominated by the weakest link. Challenges often arise from integration mechanisms that shall improve efficiency. Research should, therefore, address effective and efficient mechanisms for bounding interference on all levels, not only of time. Another important topic are mechanism which work under errors.

Hard and soft, low and high, how mixed-criticality makes the difference between important and urgent?

4.7 Real-time Performance Evaluation and VT Control mechanisms for the timing correct use of shared main memory

Kai Lampka (Uppsala University, SE)

kai.lampka@it.uu.se

License © Creative Commons BY 3.0 Unported license
© Kai Lampka

Joint work of Lampka, Kai; Georgia Giannopolou; Nikolay Stoimenov (ETH Zurich); Jonas Flodin; Yi Wang
Main reference K. Lampka, G. Giannopolou, R. Pellizzoni, Z. Wu, N. Stoimenov, “A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets,” *Real-Time Systems*, 50(5):736–773, 2014.

URL <http://dx.doi.org/10.1007/s11241-014-9211-y>

This presentation considers sets of real-time tasks executing in parallel on different cores and sharing parts of the memory hierarchy. For quantifying and handling contention at the DRAM controller this presentation presents the following recent innovations.

Worst-case response time analysis based on Timed Automata

The proposed method exploits the so-called superblock model of the work of Schranzhofer et al. [2], respectively the PRedictable Execution Model (PREM) of Pellizzoni et al. [3]. This limits the time non-determinism inherent to the occurrence of cache misses, respectively memory (data) fetches. To achieve scalability we suggest to replace some of the Timed Automata models with an abstract representation based on access request arrival curves, rather than using an individual component TA model for each core and its real-time workload.

Memory access bandwidth control

The proposed adaptive budgeting technique controls the access frequencies of applications to the main memory. To bound the interference of co-running soft real-time tasks, past works have proposed periodic server-based memory access reservation mechanisms [7, 5, 4, 6]. As the computed budgets are commonly extremely pessimistic, they reflect the worst-case rather than the normal resource use, it can be assumed that tasks under memory access budgeting experience a severe degradation of their average response time. For the hard real-time tasks, commonly implementing system control functions, this degradation is irrelevant, what matters is the guarantee that all deadlines are met. However, for user-centric soft-real time applications performance degradation should be reduced. The presented approach addresses this obstacle by dynamically changing sizes of budgets or simply ignoring them once a hard real-time task has terminated before its set worst case response time and there is no job release of some other hard real-time task.

References

- 1 K. Lampka, G. Giannopolou, R. Pellizzoni, Z. Wu, and N. Stoimenov. A formal approach to the WCRT analysis of multicore systems with memory contention under phase-structured task sets. *Real-Time Systems*, 50(5-6):736–773, 2014.
- 2 A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo. Timing analysis for resource access interference on adaptive resource arbiters. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 213–222, 2011.

- 3 G. Yao, R. Pellizzoni, S. Bak, E. Betti, and M. Caccamo. Memory-centric scheduling for multicore hard real-time systems. *Real-Time Systems Journal*, 48(6):681–715, Nov 2012.
- 4 W. Jing. Performance isolation for mixed criticality real-time system on multicore with xen hypervisor. Master’s thesis, Uppsala University, Department of Information Technology, 2013.
- 5 H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, pages 299–308, 2012.
- 6 H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, pages 55–64, 2013.
- 7 M. Behnam, R. Inam, T. Nolte, and M. Sjödin. Multi-core composability in the face of memory-bus contention. *SIGBED Rev.*, 10(3):35–42, Oct. 2013.
- 8 J. Flodin, K. Lampka, and W. Yi. Combining Performance Monitoring and Resource Budgeting on Multi-core for Real-Time Guarantees. In *MCC 2013 – Sixth Swedish Workshop on Multicore Computing*, 2013.

4.8 System-level, Inter-Criticality, Multi-Core Resource Sharing with Scalable Predictability

Gabriel Parmer (The George Washington University, Washington, DC, US)
gparmer@gwu.edu

License © Creative Commons BY 3.0 Unported license
© Gabriel Parmer

Background

Multi-core systems have proven to be a double-sided sword for embedded and real-time systems. They provide increases in computational power that promise to not only consolidate previously distributed systems together, but also to increase the computational capability, thus intelligence and functionality, of embedded systems. However, these parallel systems present a significant challenge due to the interference between tasks caused by increased resource sharing between cores. For example, different cores often share hardware resources such as last-level caches (LLC) and memory buses. Past research has addressed each of these in turn by, for example, partitioning memory [1] or cache [2]. An inescapable challenge not addressed by these techniques is the *interference caused by the sharing relationships of data-structures within software due to cache coherency*. This problem is complementary to previous approaches, and it is particularly important: a store to a cache-line can (on our 40-core, 4 socket, cache-coherent hardware) take three cycles, or *more than 27 μ s*, depending on coherency behavior.

Note that this is relevant to all shared structure access, and is orthogonal to the mechanisms for mutual exclusion and their resource sharing protocols. When considering such implementations, even predictable (FIFO) spin-locks that are known to be scalable in the average case (MCS locks), have worst-case latencies of 50 μ s which increases to 65 μ s if even a single cache line is modified within the critical section.

The impact of the overhead for loads and stores that access data-structures on shared cache lines not only impacts a task’s response time, but also increases the interference between competing tasks. One task’s data-structure access pattern in the kernel can increase

the latency of another. This cross-talk makes temporal isolation difficult across criticalities. A high criticality task, tested in isolation on a system could suffer memory access latency spikes when a low criticality task is added to the system that contends a shared kernel data-structure.

Scalable Predictability

Just as the real-time community designs techniques to provide isolation given access to shared hardware resources such as cache and memory buses, this talk will discuss the major challenges in designing software to enable controlled access to data-structures that are shared between tasks of different criticalities across cores. Specifically, our goal is to provide access to data-structures shared between cores that not only scales with increasing core counts in terms of average-case performance (i.e. the per-operation overhead doesn't increase), but also in terms of the *worst-case latencies*. We call this *scalable predictability*, and it is a strong form of scalability that focuses on the *worst-case overheads from cache-line coherency traffic* – in addition to the average behaviors that are often the focus of scalable systems – and on avoiding coherency traffic all-together. Scalable predictability means that the latency bounds provided on a single core and in isolation of all lower criticalities, don't increase with a rising number of cores.

Techniques for Scalable Predictability

This talk will be discussing recent work that will appear in RTAS, and more recent research into further techniques for scalable predictability. Methods and mechanisms to handle concurrent data-structure access are derived or borrowed from techniques in the High-Performance Computing and scalable software construction realms.

Existing techniques. Here we'll briefly survey existing concurrency control mechanisms for shared data-structures, and assess them for their scalability properties. We assume that the data-structures to be protected are both accessed on multiple cores, and by tasks of multiple criticalities (though the code that defines the access methods for the data-structure is of high assurance [3]). In other words, they are typical kernel data-structures.

- *Predictable locks.* Locks are often backed by at least one common cache-line. The cache-coherency traffic due to this cache-line “bouncing” between caches results in the significant overheads discussed previously.
- *Read-write locks.* Many data-structures are read mostly, thus enabling parallel access for readers to the data structures increases parallelism. However, these locks suffer from the same deficiency with respect to scalable parallelism as normal predictable locks: large worst-case latencies due to bouncing the lock's cache line (or in some cases, multiple cache-lines).
- *Read-Copy-Update (RCU).* RCU enables very low overhead reads to data-structures, often without *any* writes to shared structures. However, modifications to such structures involves ensuring that no readers are still accessing the modified portions before returning. This heavily penalizes writers that require coherency traffic for consensus. Thus RCU is mainly used in read-mostly workloads.
- *Reference counting.* Object liveness is often interrelated with mutual exclusion as references to an object can only be removed with proper coordination between the object, and the data-structure that is referencing it. Reference counting is the pervasive technique

often used to track this, but it suffers from average-case scalability overheads, let alone issues with scalable predictability.

Techniques for scalable predictability. This talk will include an overview of a few techniques we propose as serving as a foundation for a community investigation of worst-case scalability. These include:

- *Lock-less lookup structures with fine-grained consistency control.* Operating systems often must map between an opaque identifier (*e.g.* a file descriptor, mailbox id, process identifier), and the data-structure that backs it. The very data-structures that typically provide this map often require protection with locks. Thus, regardless of which criticality is accessing the namespace, and on which core, data-structure modification, and lock cache-line bouncing significantly impact high-criticality latencies. Lock-less data-structures that rely on liveness based on the very timing properties provided by the system (see quiescence below) provide a strong foundation for worst-case scalability.
- *Explicit mapping of namespaces to cache-line accesses.* Though the objects tracked in the kernel can be located without any cache-line modifications using the previous technique. However, once located, any modifications to the object must not impact the cache-lines of other objects. The goal is to enable the users of the API to tailor their access to the namespace to explicitly avoid accesses and modifications to cache lines for any other object. When data-structures require modification, the goal here is to enable the modifications to be at the finest possible granularity, so that scalable predictability is only compromised when cores and criticalities access the exact same object.
- *Quiescence-based liveness.* By avoiding any shared cache-line modification on kernel object lookup, the liveness question must be answered. Can a kernel object be deallocated, or are there parallel accesses to it on another core? Quiescence-based memory reclamation answers this question by ascertaining a point in the future when all references accessed before the object is freed, cannot persist. Real-time and predictable systems offer a significant benefit here: we can base our quiescence period on the latency bounds provided by the system itself.

Case Study: The SPeCK Kernel

This talk will discuss a case study we've conducted for scalable predictability in the SPeCK kernel [4] which is our new kernel for the COMPOSITE component-based OS. Through a combination of using the techniques listed above, it is able to provide scalable predictability guarantees for many of its most important operations. Notably, the operations used by computation in hard real-time components are worst-case scalable, and even operations that have traditionally never had scalable solutions, such as TLB coherence on page unmap, are usable in real-time computation. Additionally, SPeCK provides the features required by COMPOSITE systems: all resource management policies and most system abstractions are defined in user-level components including scheduling, memory mapping management, and I/O.

References


- 1 H. Kim, D. deNiz, B. Andersson, M. Klein, O. Mutlu, and R. (Raj) Rajkumar, "Bounding memory interference delay in COTS-based multi-core systems," in *RTAS*, 2014.
- 2 H. Kim, A. Kandhalu, and R. Rajkumar, "A coordinated approach for practical OS-level cache management in multi-core real-time systems," in *ECRTS*, 2013.

- 3 E. Armbrust, J. Song, G. Bloom, and G. Parmer, “On spatial isolation for mixed criticality, embedded systems,” in *2nd International Workshop on Mixed Criticality Systems (WMC)*, 2014.
- 4 Q. Wang, Y. Ren, M. Scaperoth, and G. Parmer, “Speck: A kernel for scalable predictability,” in *Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2015.

4.9 Mixed Criticality Support on Networks-on-Chip

Leandro Soares Indrusiak (University of York, UK)

lsi@cs.york.ac.uk

License  Creative Commons BY 3.0 Unported license
© Leandro Soares Indrusiak

Overview

Networks-on-Chip (NoCs) are a widely used on-chip interconnect architecture for large multi and many-core processors. They provide packet-switching infrastructure for multiple types of system-wide communications, such as message passing between tasks running on different cores, data transfers between external memories and local scratchpads, or paging and coherency mechanisms for multi-level caches. In the work reported here, we focus on the first two types of communication, which deal with coarse-grain communications (i.e. messages and data blocks rather than cache lines). Thus, we consider that mixed-criticality application tasks executing over such a processor exchange data packets of different criticality levels through the NoC infrastructure. This leads to a situation where the transmission of a packet has potential impact over the latency of all the others. Therefore, the design of the NoC infrastructure must logically separate packets of different criticality, so that their distinct requirements can be satisfied even though they share the same interconnect.

Mixed-Criticality Networks-on-Chip

WPMC [1] is a protocol applied to NoCs with virtual channels (VCs) that are arbitrated at the flit-level using a priority-preemptive mechanism. This means that each output port will send out, in every cycle, a data word (flit) from the input VC with the highest priority. WPMC aims to provide hard real-time guarantees to all criticality levels (i.e. all packets will arrive by their deadlines even in the worst-case scenario) and supports sporadic as well as periodic traffic patterns. It follows Vestal’s assumption [2] that application components of high criticality will be given more generous upper bounds for their timing behaviour, e.g. due to more strict analysis or to larger safety margins; and that components of low criticality, which are likely to be analysed with less strict techniques or which are given smaller safety margins, will have tighter upper bounds for their timing behaviour. In line with that approach, WPMC assumes that high-criticality traffic is likely to have potentially larger packets, or having packets injected more often into the NoC, as this would be a safer upper bound on the load it may impose to the NoC.

A key idea of WPMC, which was also used in the AMC scheduling algorithm [3], is that traffic of high criticality could also be analysed with the same techniques and safety margins used to profile low criticality traffic, and thus be given tighter upper bounds to its timing behaviour. The tight upper bounds can be used to dimension the NoC in such a way that all packets will always meet their deadline, as long as they don’t exceed their low

criticality upper bounds (i.e. maximum packet size, minimum packet inter-arrival interval). WPMC uses runtime monitoring to check whether all high-criticality traffic stays within their low-criticality upper bounds. The moment one of them exceeds that bound, the system is said to change into a high-criticality mode. To guarantee the timely delivery of all high-criticality packets under that mode, the NoC is allowed to drop all low-criticality traffic (as a way to achieve graceful degradation). Thus, to ensure the system is dimensioned to cope with the high-criticality mode, it must be able to support only the high-criticality traffic, but considering their more generous upper bounds. In [1], we defined the NoC mechanisms to perform the runtime monitoring, signalise mode change, and to change the NoC arbitration policies to drop low-criticality traffic. We also provided schedulability analysis to evaluate whether a given NoC is properly dimensioned to cope with the traffic produced by a given (set of) application(s) under the default low-criticality mode, as well as during and after a change to the high-criticality mode is detected.

A number of extensions and improvements to WPMC are currently being researched and developed, including:

- An alternative mode-change propagation strategy that floods the network and forces the whole NoC to change its criticality level.
- An improved credit-based flow control that allows low criticality packets to be transferred without impact on high criticality packets, even after a mode change.
- The set of conditions that must be satisfied before a mode change from high to low-criticality mode, which has not been supported by WPMC.
- Task allocation heuristics that can be used to improve schedulability of a given mixed-criticality application mapped onto a specific NoC.

This contribution will provide an overview of WPMC, and will present the progress on each one of the extensions and improvements mentioned above.

References

- 1 A. Burns, J. Harbin and L.S. Indrusiak. A Wormhole NoC Protocol for Mixed Criticality Systems In *Proc. IEEE RTSS*, pages 184–195, 2014.
- 2 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
- 3 S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE RTSS*, 2011, pages 34–43.

How do we map criticalities to certification levels – a probabilistic attempt

4.10 Mapping criticalities to certification levels – a probabilistic attempt

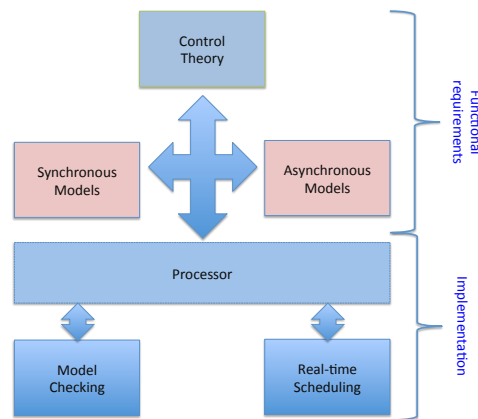
Liliana Cucu-Grosjean and Adriana Gogonel (INRIA, FR)

{liliana.cucu,adriana.gogonel}@inria.fr

License © Creative Commons BY 3.0 Unported license
© Liliana Cucu-Grosjean and Adriana Gogonel

Some context

The main feature of time critical embedded systems concerns the respect of temporal constraints. The correctness of each computation within these systems depends on both the



■ **Figure 4** Different phases of the design of a time critical embedded system.

logical results of the computation and the time at which these results are produced.

The design of a time critical embedded system may have basically three main phases: (i) the description of the physical process that should be controlled, (ii) the description of the functional requirements that should be fulfilled and (iii) the description of the implementation of the time critical embedded system. During the first phase the characteristics of the physical process are described using control theory. Then a model is proposed using synchronous or asynchronous modelling and this model is verified using model checking. At the end of the second phase the designer has a model of the system that is correct with respect to the expected functional requirements. During the last phase (of implementation), the processors are taken into account and the time feasibility of the system is checked using methods like formal verification or real-time schedulability analysis. The relations between different phases of conception is provided in Figure 4.

The pessimism of all existing solutions comes mainly from the implementation phase where an absolute value is considered for the worst case execution time of a program. The arrival of modern and more complex processors (e.g., use of caches, multi- and many-core processors) increases the timing variability of programs, i.e., the absolute worst case execution time is becoming significantly larger. For instance, larger execution times require an increased number of processors or more powerful processors.

Our open problem

An intuitive solution to overcome this pessimism is the introduction by Steve Vestal [1] of the notion of mixed criticality for time critical embedded systems. This solution defines several possible values for the worst case execution time of a program on a processor and it has propagated from the original work on scheduling theory [2] to synchronous languages [3], predictable processors [4], model checking [5], etc.

Nevertheless today the mixed criticality solutions are heterogeneous and they are proposed for different phases of design without a common framework. In conclusion we identify as vital **the need for a modular framework unifying heterogeneous solutions of the design problem of mixed criticality systems without re-writing the entire theory of time critical embedded systems.**

Our intuition is that **probabilistic description of some parameters or properties of existing models is a possible solution** to the problem of designing time critical embedded systems.

Nevertheless **the introduction of probabilities is not trivial** as not every probabilistic approach may be used to study time critical embedded systems. Indeed the introduction of probabilistic descriptions in all phases of the design of time critical embedded systems should be done such that the two following properties are ensured:

1. worst case values are rare events;
2. probabilistic worst case reasoning is applicable.

References

- 1 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
- 2 A. Burns and R. I. Davis. Mixed Criticality Systems – A Review. *Department of Computer Science, University of York, Report. Fourth edition, July 31, 2014.*
- 3 E. Yip, M. Kuo, D. Broman, and P.S. Roop. Relaxing the Synchronous Approach for Mixed-Criticality Systems. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, pages 89–100. IEEE, 2014.
- 4 M. Zimmer, D. Broman, C. Shaver, and E.A. Lee. FlexPRET: A Processor Platform for Mixed-Criticality Systems. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, pages 101–110. IEEE, 2014.
- 5 A.J. Boudjadar and A. David and J. Kim and K.G. Larsen and M. Mikucionis and U. Nyman and A. Skou. Degree of Schedulability of Mixed-Criticality Real-Time Systems with Probabilistic Sporadic Tasks. In *the book Theoretical Aspects of Software Engineering Conference*, 2014

4.11 Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters

Dorin Maxim (*The Polytechnic Institute of Porto, PT*)
dorin@isep.ipp.pt

License © Creative Commons BY 3.0 Unported license
© Dorin Maxim

Joint work of Maxim, Dorin; Cucu-Grosjean, Liliana;

Main reference D. Maxim, L. Cucu-Grosjean, “Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters,” in *Proc. of the IEEE 34th Real-Time Systems Symposium (RTSS’13)*, pp. 224–235, IEEE, 2013.

URL <http://dx.doi.org/10.1109/RTSS.2013.30>

Introduction

We consider a system of n synchronous tasks $\{\tau_1, \tau_2, \dots, \tau_n\}$ to be scheduled on one processor according to a preemptive fixed-priority task-level scheduling policy. Without loss of generality, we consider that τ_i has a higher priority than τ_j for $i < j$. By synchronous tasks we understand that all tasks are released simultaneously the first time at $t = 0$.

Each task τ_i generates an infinite number of successive jobs $\tau_{i,j}$, with $j = 1, \dots, \infty$. All jobs are assumed to be independent of other jobs of the same task and those of other tasks.

Each task τ_i is a generalized sporadic task [1] and it is represented by a probabilistic worst case execution time (pWCET) denoted by \mathcal{C}_i^2 and by a probabilistic minimum inter-arrival time (pMIT) denoted by \mathcal{T}_i .

² In this paper, we use calligraphic typeface to denote random variables.

The probabilistic execution time (pET) of a job of a task describes the probability that the execution time of the job is equal to a given value. A safe pWCET \mathcal{C}_i is an upper bound on the pETs $\mathcal{C}_i^j, \forall j$ and it may be described by the relation \succeq as $\mathcal{C}_i \succeq \mathcal{C}_i^j, \forall j$. Graphically this means that the CDF of \mathcal{C}_i stays under the CDF of $\mathcal{C}_i^j, \forall j$.

Following the same reasoning the probabilistic minimal inter-arrival time (pMIT) denoted by \mathcal{T}_i describes the probabilistic minimal inter-arrival times of all jobs. The probabilistic inter-arrival time (pIT) of a job of a task describes the probability that the job's arrival time occurs at a given value. A safe pMIT \mathcal{T}_i is a bound on the pITs $\mathcal{T}_i^j, \forall j$ and it may be described by the relation \succeq as $\mathcal{T}_i^j \succeq \mathcal{T}_i, \forall j$. Graphically this means that the CDF of \mathcal{T}_i stays below the CDF of $\mathcal{T}_i^j, \forall j$.

Hence, a task τ_i is represented by a tuple $(\mathcal{C}_i, \mathcal{T}_i)$. A job of a task must finish its execution before the arrival of the next job of the same task, i.e., the arrival of a new job represents the deadline of the current job. Thus, the task's deadline may also be represented by a random variable \mathcal{D}_i which has the same distribution as its pMIT, \mathcal{T}_i . Alternatively, we can consider the deadline described by a distribution different from the distribution of its pMIT if the system under consideration calls for such model, or the simpler case when the deadline of a task is given as one value. The latter case is probably the most frequent in practice, nevertheless we prefer to propose an analysis as general as possible and in the rest of the paper, we consider tasks with implicit deadlines, i.e., having the same distribution as the pMIT.

Problem description: We address the problem of computing the response time distributions and, implicitly, Deadline Miss Probabilities (DMP) of tasks with pMIT and pWCET. The response time of a job is the elapsed time between its release and its completion. Since we consider jobs with probabilistic parameters, the response time of a job is also described by a random variable. The DMP of a job is obtained by comparing the response time distribution of said job and its deadline, be it a probabilistic deadline or a deterministic one. This is a novel problem, and the fact that the system under consideration has more than one task parameter given as a distribution makes it a complex one.

Probabilistic response time analysis

The probabilistic worst case response time (pWCRT) \mathcal{R}_n of a task τ_n in the critical instance is computed by coalescing all the distributions $\mathcal{R}_n^{i,j}$ (called copies) resulted by iteratively solving the following equation (from [2]):

$$\mathcal{R}_n^{i,j} = (\mathcal{R}_n^{i-1,head} \oplus (\mathcal{R}_n^{i-1,tail} \otimes \mathcal{C}_m^{pr})) \otimes \mathcal{P}_{pr} \quad (1)$$

The iterations end when there are no more arrival of any job i of any higher priority task τ_m that occurs within the response time distribution at the current step. A stopping condition may be explicitly placed in order to stop the analysis after a desired response time accuracy has been reached. For example, the analysis can be terminated once an accuracy of 10^{-9} has been reached for the response time. In our case, the analysis stops when new arrivals of the preempting tasks are beyond the deadline of the task under analysis, i.e., the type of analysis required for systems where jobs are aborted once they reach their deadline.

Once the jobs' response time distribution can be computed, the Deadline Miss Probability can be obtained by comparing the response time distribution with that of the deadline, as follows:

$$\mathcal{B}_i = \mathcal{R}_i \ominus \mathcal{D}_i = \mathcal{R}_i \oplus (-\mathcal{D}_i), \quad (2)$$

where the \ominus operator indicates that the values of the distribution are negated.

Note that the analysis can handle any combination of probabilistic and deterministic parameters, and in the case that all parameters are deterministic the returned result is the same as the one provided by the worst case response time analysis in [3]. More details about the analysis can be found in [2].

References

- 1 A. Ka-Lau Mok (1983). *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Massachusetts Institute of Technology.
- 2 D. Maxim and L. Cucu-Grosjean. *Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters*. In *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013*.
- 3 M. Joseph and P.K. Pandya. *Finding response times in a real-time system*. In *The Computer Journal* 29(5):390–395, (1986).

What is the meaning of mixed-criticality when time is the keyword?

4.12 Viewpoints on the Timing Aspect of Mixed Criticality Systems

David Broman (KTH Royal Institute of Technology, SE)
dbro@kth.se

License  Creative Commons BY 3.0 Unported license
© David Broman

Mixed criticality systems can be informally defined as systems where software components, with different levels of criticality, execute on the same hardware platform. Starting with the paper by Vestal [1] in 2007, a large body of research results has been presented within the real-time community. The common research problem can be seen as the challenge of reconciling the two requirements of partitioning for safety, and sharing resources [2]. There are, however, several different viewpoints on the timing aspects of mixed criticality systems; in particular of the meaning of criticality levels. We separate between two distinct viewpoints: i) the implementation view, and ii) the specification view.

In the *implementation view*, the model and meaning of criticality level also include aspects of the implementation. That is, consideration needs to be taken to the actual hardware platform and operating system (OS) that are used. Vestal's classic task model [1] falls into this category; different WCET estimate numbers are used for different criticality levels. To be able to get these numbers, programs need to be executed and measured on the real hardware platform, or accurate timing models of the hardware need to be used when computing safe bounds of the WCET. Other variants of Vestal's model, for instance Burns and Baruah's variant [3], can also be considered to fall into the same category. Both these examples are based on software scheduling for mixed criticality systems.

Another approach is to perform the scheduling in hardware. The FlexPRET [4] processor platform is an example of hardware-based scheduling using fine-grained multithreading. In this approach, several hardware threads are used, which either fall into the category of hard real-time threads or soft real-time threads. Hard real-time threads are guaranteed to have both temporal and spatial isolation, whereas soft real-time threads do not have temporal isolation, but can steal cycles from the hard real-time threads when they are not active. Tasks with different levels of criticality can then be scheduled on either hard real-time or

soft real-time threads using only hardware scheduling or with a combination of hardware and traditional software scheduling. Clearly, this approach for ensuring the timing aspects of mixed criticality systems falls under the category of the implementation view.

An alternative is to use a *specification view* for the timing aspects. For such a viewpoint, nothing within the definition of the criticality levels should say anything about implementation aspects. One such approach is to define the criticality levels using frequency bounds. Yip et al. [5] proposes such approach, where tasks are divided into three different criticality levels. Each periodic task has two frequency parameters: f_{max} and f_{min} , meaning that the task is allowed to be executed with a frequency that falls within this interval. For life critical tasks, $f_{max} = f_{min}$ and for mission critical tasks $f_{max} > f_{min}$. For non-critical tasks, f_{max} is the goal frequency and $f_{min} = 0$. Note that this task model does not say anything about the implementation technique or WCET numbers for specific platforms. The different criticality levels are specified using constraints on timing.

The idea of using timing specifications can be taken further to make it more expressive. We call this approach *programming with time*, meaning that time and timing become part of a programming model. At the Dagstuhl seminar, I presented some work-in-progress about incorporating time in a small language, by formalizing the semantics using small-step operational semantics.

References

- 1 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
- 2 A. Burns and R. I. Davis. Mixed Criticality Systems – A Review. *Department of Computer Science, University of York, Report. Fourth edition, July 31, 2014.*
- 3 A. Burns, S. Baruah, K. M. Phan, and I. Shin. Towards a more practical model for mixed criticality systems. In *WMC*, 2013.
- 4 M. Zimmer, D. Broman, C. Shaver, and E. A. Lee. FlexPRET: A Processor Platform for Mixed-Criticality Systems. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, pages 101–110. IEEE, 2014.
- 5 E. Yip, M. Kuo, D. Broman, and P. S. Roop. Relaxing the Synchronous Approach for Mixed-Criticality Systems. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*, pages 89–100. IEEE, 2014.

4.13 Mapping the landscape of mixed criticality systems research

Sanjoy K. Baruah (University of North Carolina at Chapel Hill, US)

baruah@cs.unc.edu

License © Creative Commons BY 3.0 Unported license
© Sanjoy K. Baruah

There appears to be general agreement on the definition of mixed-criticality systems: a mixed-criticality system is a system in which functionalities of different specified criticalities are implemented upon a shared platform. Beyond this general definition, however, the situation parallels that described in John Godfrey Saxe’s poem *The Blind Men and the Elephant*: different interpretations abound, each highlighting selected aspects of mixed-criticality systems while choosing to minimize (or ignore) other aspects. It is important to be cognizant of these different perspectives, and to better understand the different contexts

and requirements that motivate the different interpretations; else, we end up with different sub-communities speaking across one another and misunderstanding each other – the same terms mean very different things to different people. Some of the different perspectives that I am aware of are listed below.

Perspective 1. A perspective that is found in the safety-critical systems industry holds that different criticality levels are user-specified attributes that have no additional semantic interpretation – different criticality levels are not really comparable to each other. Different requirements, such as correctness criteria, are specified for different criticality levels; each functionality is expected to satisfy the requirements for its specified criticality level. For instance, a presentation³ advocating this perspective explicitly states:

What [a mixed-criticality system] is NOT: A system where system approach sacrifices lower criticality applications for whatever purpose.

If the different criticality levels are assumed incomparable in this manner, then a reasonable objective of mixed-criticality research should be to devise mechanisms and policies that enable *isolation* amongst the different criticality levels. The research questions here then seek to determine how best to provide such isolation upon modern platforms (such as multicores), particularly as systems become increasingly more complex.

Perspective 2. In much of the mixed-criticality real-time scheduling literature, it is assumed that the different criticality levels correspond to different degrees of importance: a functionality that is semantically more important is assigned greater criticality. (For example, safety-critical functionalities may be accorded greater criticality than mission-critical ones.) The research objective here is to seek more resource-efficient implementations of such mixed-criticality systems. Such research may be classified into two broad categories according to the different approaches adopted:

- **Perspective 2A (Run-time adaptation).** Under this approach a mixed-criticality system starts out executing functionalities of different criticalities, based upon optimistic assumptions regarding resource requirements. If these optimistic assumptions are observed during run-time to not hold, then the system adapts its run-time behavior to allocate additional resources to the more critical functionalities; less critical functionalities receive less resources and may experience a consequent degradation in performance.
- **Perspective 2B (Pre-run-time verification).** Such research is based on the principle that resources must be provisioned under more conservative assumptions to more critical functionalities, in order to meet their more stringent correctness criteria – those typically include a requirement that such functionalities have their correctness validated to higher levels of assurance. Some of these provisioned resources may then be reclaimed during system design time itself, and used to make performance guarantees at lower levels of assurance to less critical functionalities. (This is the approach that is currently commonly referred to within the real-time scheduling theory community as the *Vestal approach*, in recognition of the fact that it was first proposed in a paper⁴ by Vestal.) The research

³ Michael Paulitsch (Thales) and Jan Nowotzsch (Airbus Group). *Monitoring Techniques in COTS Multicore Processors in Mixed-Criticality Systems with Focus on Temporal Aspects*. Torrent Workshop, Toulouse, December 12, 2014. Slides available at <http://www.irit.fr/torrents/seminars/20141212/20141212-paulitsch.pdf> (Date accessed: 2015/02/18).

⁴ Steve Vestal. *Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance*. Proceedings of the IEEE Real-Time Systems Symposium (RTSS), pp. 239–243. 2007.

activities within this category include seeking novel innovative ways of achieving such design-time resource reclamation.

Summary. Above, three different perspective to mixed-criticality systems research have been identified. Perspective 1 differs greatly from the two perspectives 2A and 2B, while the differences between 2A and 2B are somewhat more subtle. It is important to identify additional perspectives that may exist, and to study the relationships between these different perspectives in order to understand whether there are commonalities that allow for mutually beneficial interaction amongst advocates of the different perspectives, perhaps leading to the development of common research agendas.

4.14 Some Open Problems in Mixed-Criticality Scheduling

Pontus Ekberg (Uppsala University, SE)

pontus.ekberg@it.uu.se

License  Creative Commons BY 3.0 Unported license
© Pontus Ekberg

I list a few open problems that I consider foundational for our understanding of mixed-criticality scheduling theory. Sprinkled among the questions are a few related claims, these are accompanied by much waving of hands. The questions concern the scheduling of mixed-criticality workload of the common “Vestal-type”. In addition, they are restricted to systems with two criticality levels (LO and HI) running on a preemptive uniprocessor. Not because that is necessarily the most interesting case, but because we need to understand the basics first.

Let us start by considering static collections of mixed-criticality jobs, and denote with MC-JOB-SCHEDULABILITY the decision problem of whether a given collection of jobs has a correct online (i.e., non-clairvoyant) schedule. Further, let us slightly abuse established notation and denote with EDF-VD the *family* of schedulers that follow these rules:

1. Schedule all jobs J_i in EDF order, but according to their *virtual deadlines* v_i instead of absolute deadlines d_i .
2. In LO-criticality mode,
 - a. $v_i = d_i$ for LO-jobs and
 - b. $v_i \in [a_i, d_i]$ for HI-jobs.
3. In HI-criticality mode,
 - a. $v_i = \infty$ for LO-jobs and
 - b. $v_i = d_i$ for HI-jobs.

Baruah et al. [1] showed MC-JOB-SCHEDULABILITY to be strongly NP-complete for any constant number of criticality levels. The hardness part of their proof is a reduction from 3-PARTITION. It is fairly easy to see that all feasible job collections they construct are schedulable by some scheduler in the EDF-VD family, and given such a scheduler it can be verified whether it is correct in polynomial time. Therefore, it must be hard to identify which scheduler in the EDF-VD family to use.

Claim

Finding an optimal assignment of virtual deadlines for MC job collections when using EDF-VD is strongly NP-hard.

Why this focus on the EDF-VD family? It is because I believe the answer to the following question to be “yes”.

Question

Given any collection of (2-level) MC jobs that is online schedulable, is there always a correct scheduler in the EDF-VD family for it

Now we turn to MC sporadic tasks instead. Let MC-SPORADIC-SCHEDULABILITY be the corresponding decision problem and let us abuse notation again and denote with EDF-VD the family of schedulers that behave as before, but with a static virtual deadline $v_i \in [0, d_i]$ per task τ_i , that is applied to all jobs of τ_i . Unfortunately, a strong link that we usually have between job collections and sporadic tasks does not exist for MC systems.

Claim

The synchronous arrival sequence is not a worst case for MC sporadic tasks.

If non-integer arrival times are allowed, the situation is even worse.

Claim

There are sporadic MC task systems which are unschedulable only when some arrival times are non-integer.

The ability to restrict attention to one or a few concrete cases is a very useful property for analysis. Can the SAS be replaced by some other case?

Question

For a given MC sporadic task set, can we efficiently identify some restricted set of job sequences that are the worst cases?

The lack of the SAS as a guaranteed worst case means that we can not trivially extend the hardness proof of Baruah et al. to sporadic MC tasks, but it seems fair to suspect that MC-SPORADIC-SCHEDULABILITY is also NP-hard. However, it is easy to see that it is coNP-hard via a reduction from the corresponding non-MC problem, and therefore it seems reasonable to suspect that it is neither in NP nor in coNP.

Question

What is the complexity of MC-SPORADIC-SCHEDULABILITY?

A closely related question is how to optimally schedule a set of sporadic MC tasks.

Question

What scheduling policies are optimal for sporadic MC tasks?

Unfortunately, the family of EDF-VD schedulers does not appear to be it, though hopefully there are some others that are also efficient at runtime.

Claim

When non-integer arrival times are allowed, there are (2-level) sporadic MC task sets that are online schedulable, but for which there are no correct schedulers in the EDF-VD family.

References

- 1 S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, and A. Marchetti-Spaccamela. The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems. In *ECRTS*, 2012.

WCET – the central notion of mixed-criticality

4.15 Runtime monitoring of time-critical tasks in multi-core systems

Christine Rochange (Paul Sabatier University – Toulouse, FR)

rochange@irit.fr

License © Creative Commons BY 3.0 Unported license
© Christine Rochange

Joint work of Kritikakou, Angeliki; Baldellon, Olivier; Pagetti, Claire; Rochange, Christine; Roy, Matthieu
Main reference A. Kritikakou, C. Pagetti, O. Baldellon, M. Roy, C. Rochange, “Run-Time Control to Increase Task Parallelism In Mixed-Critical Systems,” in Proc. of the 26th Euromicro Conf. on Real-Time Systems (ECRTS'14), pp. 119–128, IEEE, 2014.
URL <http://dx.doi.org/10.1109/ECRTS.2014.14>

Existing WCET computation methods [1] consider pessimistic situations with permanent conflicts. This leads to WCET estimates that are safe but far from the frequent case, and then to over-provisioning time slots for tasks.

The objectives of the proposed approach are to relax constraints on scheduling so that it can consider less safe but more realistic predictions of execution times. A recovery mechanism monitors high-criticality tasks to check whether they can miss their deadlines: this is done based on the remaining WCET which is dynamically updated along the execution. If a hazard is detected, highly-criticality tasks are allowed to finish their execution in a contention-free mode.

We introduced a scheme based on extended control flow graphs and partial timing information that is computed offline and stored in a table looked up at runtime to update the remaining WCET.

References

- 1 R. Wilhelm et al. *The worst-case execution-time problem: overview of methods and survey of tools*. ACM Transactions on Embedded Computing Systems (TECS), 7(3), 2008.

4.16 Timing Analysis for Multi/Many-core Platforms

Jan Reineke (Universität des Saarlandes, DE)

reineke@cs.uni-saarland.de

License © Creative Commons BY 3.0 Unported license
© Jan Reineke

Joint work of Reineke, Jan; Doerfert, Johannes; Wilhelm, Reinhard

Timing analysis seeks to answer the following question: Can a given task set be scheduled to meet all deadlines on a particular execution platform? If the execution platform is a single-core processor, timing analysis is a fairly well-understood problem. For such platforms timing analysis is commonly split into two phases:

1. *Worst-case execution time (WCET) analysis* determines for each task a bound on its execution time, independently of the other tasks.
2. *Schedulability analysis* determines whether all deadlines can be met based on these WCET bounds.

If the execution platform is a multi- or many-core processor such a clean separation into WCET and schedulability analysis is hard to maintain. Due to interference on shared resources, such as buses, caches, and DRAM-based main memory, the execution time of a task depends strongly on its execution context [1].

I discuss four approaches to timing analysis for multi- and many-core processors and their respective benefits and drawbacks:

1. The *Integrated-analysis approach*: Analyze the entire task set at once in a combined WCET and schedulability analysis. This is practically infeasible even for the analysis of two co-running tasks.
2. The *Murphy approach*: Determine a context-independent WCET bound. Perform schedulability analysis using these bounds. This can be extremely pessimistic: Radojkovic et al. [2] report a 14-fold slowdown due to interference on a shared L2 cache and memory controller, negating all performance benefits of using a multi-core processor.
3. The *Abstract interference approach*:
 1. Characterize the interference on shared resources generated by each task.
 2. Determine interference-aware WCET bounds, i.e., mappings from the amount of interference experienced to WCET bounds.
 3. Perform an extended schedulability analysis taking into account the information from 1 and 2.
4. The *Isolation approach*: Isolate tasks running on different cores by partitioning shared resources in time and space. This re-enables the single-core two-phase timing analysis approach. However, the question arises how to split the resources among the cores. An ingredient of an informed partitioning decision is *architecture-parametric timing analysis* [3]; a WCET analysis that determines how the execution time depends upon the amount of available resources.

References

- 1 A. Abel, F. Benz, J. Doerfert, B. Dörr, S. Hahn, F. Hauptenthal, M. Jacobs, A. H. Moin, J. Reineke, B. Schommer, R. Wilhelm. *Impact of Resource Sharing on Performance and Performance Prediction: A Survey*. In *In CONCUR* 2013.
- 2 P. Radojkovic, S. Girbal, A. Grasset, E. Quinones, E., S. Yehia, F. J. Cazorla: *On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments*. *ACM Transactions on Architecture and Code Optimization* 8(4), January 2012.
- 3 J. Reineke, J. Doerfert: *Architecture-Parametric Timing Analysis*. In *In RTAS* 2014.

4.17 Analysis of pre-emptive systems with caches

Sebastian Altmeyer (University of Amsterdam, NL)

altmeyer@uva.nl

License © Creative Commons BY 3.0 Unported license

© Sebastian Altmeyer

Main reference S. Altmeyer, “Analysis of Preemptively Scheduled Hard Real-time Systems,” Dissertation, Universität des Saarlandes, 2013.

URL <http://nbn-resolving.de/urn:nbn:de:bsz:291-sciodok-52797>

Proving timing correctness of an embedded system is traditionally a two-step approach: Timing analysis derives upper bounds on the execution times of tasks in isolation, called worst-case execution times (WCET). Scheduling analysis determines if each task complies with its timing constraints when scheduled according to a predefined scheduling policy. Timing constraints are typically defined by a task’s period and a task’s deadline, both determined by the physical environment. Hence, tasks are assumed to be fully characterized by a triple consisting of a period, a deadline and an execution time bound, i.e. the WCET of the task.

While this verification process provides a useful separation of concerns and a clean interface between the two steps, it fails to account for the complexity of modern embedded systems; already in the case of uniprocessor systems: History-sensitive hardware components, foremost caches, impact the system performance beyond the scope of a task. This is especially problematic in the case of pre-emptive scheduling, where the execution time of a pre-empted task strongly depends on whether previously cached data has been evicted during pre-emption or whether it is still resident in the cache. The additional execution time due to cache eviction is called cache-related pre-emption delay (CRPD). Consequently, a task’s execution time can not be analyzed independently anymore.

There are three different solutions to this problem: (i) one can inflate the execution time bounds to account for the CRPD, (ii) one can avoid CRPD by using cache partitioning, or (iii) one can adapt the timing verification process to include the CRPD as part of the task model. Solution (i) and (ii) enable the reuse of the common task model, but potentially at the cost of substantial pessimism or degraded performance. Solution (iii) requires the highest effort, but allows us to compute safe and precise bounds. The timing analysis must provide metrics for the cache-reuse (the set of useful cache blocks) and the memory footprint (set of evicting cache blocks) of each task. The scheduling analysis then needs to correctly account for these metrics and needs to identify the worst-case pre-emption scenarios, which strongly depend on the metrics provided by the timing analysis.

The timing verification process for pre-emptively scheduled uni-processors with caches may serve as a blueprint for the multicore timing verification. In the case of multicore systems, the independence-assumption of the timing analysis is violated not only due to a common memory hierarchy, but also due to a shared bus system. This shared bus causes additional interference and creates a dependency not only between tasks scheduled on the same core, but also between tasks scheduled on all other cores.

Based on what we have learned for the analysis of pre-emptive systems, we can formulate the educated guesses that the notion of WCET alone is not sufficient to correctly represent the complex behaviour on multicore systems, that a precise analysis restricts the hardware components to be used and that the complete timing verification process needs to be addressed and revised instead of just one of the two sides.

Mixed-criticality models are the answer to adaptive time critical systems?

4.18 Using Mixed-Criticality to Reason about Temporal Correctness in Uncertain & Dynamic Environments

Nathan Fisher (Wayne State University, US)
fishern@cs.wayne.edu

License © Creative Commons BY 3.0 Unported license
© Nathan Fisher

Starting with the seminal paper by Steve Vestal at RTSS 2007 [1], avionics has been the most frequently-cited motivating application domain for the development of mixed-criticality scheduling theory (MCST). The reasons are quite clear: integrating multiple avionic subsystems with different criticalities and certification levels requires guarantees that lower-criticality subsystems do not have a negative effect upon the temporal correctness of higher-criticality subsystems. Given the initial progress of the MCST research community towards addressing these system- integration goals in avionics (and related application domains), this Dagstuhl seminar is an ideal setting to reflect upon the potential broader implications (beyond system integration) of the resulting MCST from the past eight years. Specifically, I would like to raise the question of *how can MCST results be leveraged in the design of adaptive real-time systems executing in dynamic and uncertain physical environments (esp., power-aware control systems)?*

Similar Notions of Uncertainty. One important insight that has been gained from MCST research is the ability to make formal timing guarantees in the presence of uncertain execution times. For instance, in a system with two criticality levels, HI and LO, the typical model specifies that when each job's total execution time does not exceed the LO-criticality bound, then the system is considered temporally correct if all jobs (both HI and LO criticality) meet their respective deadlines; however, whenever any job exceeds its LO-criticality execution bound, then the system must only guarantee that each HI-criticality job meets its deadline. Thus, with this model of mixed criticality, a system designer is able to reason about the temporal correctness of the system without knowing an exact execution time bound for some subset of the jobs.

The area of adaptive real-time control systems often requires reasoning about execution uncertainty from a similar, but slightly different perspective. Consider the problem of maintaining the CPU temperature below a specified threshold. Under typical environmental conditions, the processor can execute normally and not exceed its temperature threshold. However, if the environmental temperature increases, the CPU is unable to dissipate the heat generated from computation as efficiently. To guard against a temperature violation, modern CPUs often have dynamic voltage/frequency scaling (DVFS) capabilities to permit a reduction in the CPU heat generation. For real-time systems these adaptive DVFS changes present a challenge in reasoning about the temporal correctness of the system given that the thermal operating environment may be dynamic and unpredictable; using DVFS will create uncertainty in the execution time of the underlying jobs and may require some to be aborted or deferred.

Opportunities & Challenges. The similar notions of execution-time uncertainty present an opportunity to “port” some of the scheduling algorithms and associated analysis developed for MCST to the domain to adaptive real-time systems. Recent work on using mixed-criticality

upon processors with varying execution speeds [2] may be one avenue to unify the notions of uncertainty used for MCST and power-aware real-time control systems. However, there are some fundamental differences in the settings that may present challenges in immediately applying MCST to such systems:

1. **Differing Mode-Change Semantics:** Traditional MCST appears to view changing modes from LO to HI as a rare event. Conversely, for systems executing in a dynamic environment, changing modes continuously to adapt is fundamental to their design. A recent talk by Alan Burns at WMC 2014 surveyed some adaptive criticality mode-change protocols that may prove useful for adaptive real-time system design [3]; the differences between these MCST-based protocols and multi-modal protocols developed specifically for power-aware control systems (e.g., [4]) warrant further discussion.
2. **Number of Operating Modes:** In power-aware systems, more operating modes (e.g., voltage/frequency levels) leads to more fine-grained control. Each of these operating modes can be viewed in MCST parlance as a “criticality level”. Unfortunately, it seems that scaling the number of criticality levels beyond two is a non-trivial objective. Thus, it may be a worthwhile exercise to investigate whether the setting of discrete control (e.g., mode changes will occur at periodic intervals corresponding to the controller’s sampling interval) can lead to some simplifications that permit an increased scaling of criticality levels.

References

- 1 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.
- 2 S. Baruah and Z. Guo. Scheduling Mixed-Criticality Implicit-Deadline Sporadic Task Systems upon a Varying-Speed Processor. *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pp. 31–40, 2014.
- 3 A. Burns. System Mode Changes – General and Criticality-Based. *Proceedings of the 2nd International Workshop on Mixed Criticality Systems*, pp. 3–8, 2014.
- 4 M. Ahmed and N. Fisher. Tractable Schedulability Analysis and Resource Allocation for Real-Time Multimodal Systems. *ACM Transactions on Embedded Computing Systems*. 13 (2s), January 2014.

4.19 Augmenting Criticality-Monotonic Scheduling with Dynamic Processor Affinities

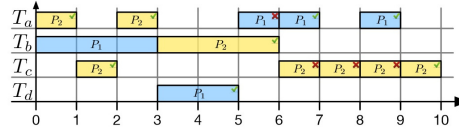
Bjoern B. Brandenburg (MPI-SWS – Kaiserslautern, DE)

bbb@mpi-sws.org

License  Creative Commons BY 3.0 Unported license
© Bjoern B. Brandenburg

Consider the problem of scheduling a dual-criticality workload consisting of high- and low-criticality sporadic real-time tasks on top of a fixed-priority (FP) scheduler. Each high-criticality (HC) task T_i has both a high- and a low- criticality WCET estimate, denoted e_i^L and e_i^H , resp., and low- criticality (LC) tasks are required to meet their deadlines only if no HC task exceeds its LC WCET estimate.

task	criticality	p_i	e_i^L	e_i^H
T_a	low	2	1	—
T_b	high	10	3	6
T_c	low	2	1	—
T_d	high	10	2	5



■ **Figure 5** In this example, T_b exceeds e_b^L at time 3. Its affinity is then set to $\{P_1, P_2\}$, which allows T_b to finish on P_2 . T_d is isolated; T_a and T_c miss one and three deadlines.

From a pragmatic point of view, FP scheduling with *criticality-monotonic* priorities [1], where HC tasks have higher priority than LC tasks, holds considerable appeal: it is simple, provides obvious isolation for HC tasks, and imposes no runtime overheads.

Unfortunately, as LC tasks may be more *urgent* than HC tasks (i.e., they may have shorter periods or more constraining deadlines), it is not always feasible to assign criticality-monotonic priorities [1]. For example, the task set $\tau_1 = \{T_a, T_b\}$ (as specified in Fig. 5), which consists of a LC task T_a that is urgent (i.e, it has a short period $p_a = 2$) and a HC task T_b that is less urgent ($p_b = 10$) but more costly ($e_b^L = 3$), cannot be scheduled on a uniprocessor with criticality-monotonic priorities: the LC task T_a , if given a lower priority than T_b , may miss deadlines even if no job of T_b exceeds e_b^L .

Similar urgency vs. criticality conflicts also arise on multiprocessors. For instance, the task set $\tau_2 = \{T_a, T_b, T_c, T_d\}$ cannot be scheduled with criticality-monotonic priorities on $m = 2$ cores using either *global* or *partitioned* FP scheduling: under global scheduling, the HC tasks T_b and T_d can cause the more-urgent LC tasks T_a and T_c to miss deadlines even with LC execution costs, and under partitioned scheduling, T_b and T_d need to be assigned to different partitions, but neither can be co-located with T_a or T_c . However, while scheduling τ_1 with criticality-monotonic priorities is infeasible on a uniprocessor, τ_2 can be scheduled with criticality-monotonic priorities on two processors—provided *processor affinities* are used to shield urgent tasks in the LC case.

Exploiting Arbitrary Processor Affinities (APAs)

Contemporary OSs such as Linux, Windows, QNX, or VxWorks provide flexible APIs to explicitly set a task’s processor affinity, which is the set of processors on which it may execute. In particular, task affinities can be restricted to arbitrary processor sets and changed at arbitrary times during runtime. This can be exploited to render criticality-monotonic scheduling feasible.

Consider the following strategy for scheduling τ_2 on two processors P_1 and P_2 : (1) Tasks are assigned criticality-monotonic priorities. (2) T_a and T_c may execute on both P_1 and P_2 . (3) T_b and T_d may initially execute only on processor P_1 . (4) When a HC job J_x of T_b (resp., T_d) fails to complete after e_b^L (resp., e_d^L) time units, it updates its processor affinity to include both P_1 and P_2 . (The processor affinity of any other task is *not* changed.) (5) A HC task’s affinity is reset when it completes its job.

A possible schedule is shown in Fig. 5: at time 3, when it becomes known that T_b ’s job requires more than $e_b^L = 3$ time units to complete, it relaxes its processor affinity to include P_1 and P_2 . Consequently, under a FP scheduler with *strong APA semantics* [2] — which, intuitively, is an APA scheduler that *shifts* higher-priority tasks from one processor to another if that is required to enable lower-priority tasks with more- constraining affinities to be scheduled — T_b shifts to P_2 , which enables T_d to be scheduled on P_1 . As T_b handles its increased demand on P_2 , T_d is protected from undue interference. LC tasks are not dropped, but may temporarily incur deadline misses.

Remarks and outlook

We have observed that an APA interface – readily available in current, already certified RTOSs – allows the timeliness requirements of urgent LC tasks to be reconciled with the desirable simplicity of criticality- monotonic scheduling. The sketched approach offers several practical benefits: HC tasks exceeding their LC WCET are effectively given a “dedicated” processor to cope with increased demand; only the currently executing task’s affinity is adapted, which keeps runtime overheads low and independent of the number of tasks; there is no “mode change” and LC tasks are not abandoned, just temporarily delayed; and budget enforcement is not required.

Of course, the above example works only because of simplifying assumptions. We believe, however, that it is possible to generalize the approach to an arbitrary number of HC tasks and also to *weak* APA schedulers [2] such as those found in QNX and Linux.

References

- 1 S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE RTSS*, 2011, pages 34–43.
- 2 F. Cerqueira, A. Gujarati, and B. Brandenburg. Linux’s processor affinity API, refined: *Shifting* real-time tasks towards higher schedulability. In *RTSS*, 2014.

4.20 Adaptive Uni-processor Fixed Priority Pre-emptive Probabilistic Mixed Criticality

Yasmina Abdedda (*Université Paris-Est, LIGM UMR CNRS 8049, ESIEE Paris, FR*)

yasmina.abdeddaim@esiee.fr

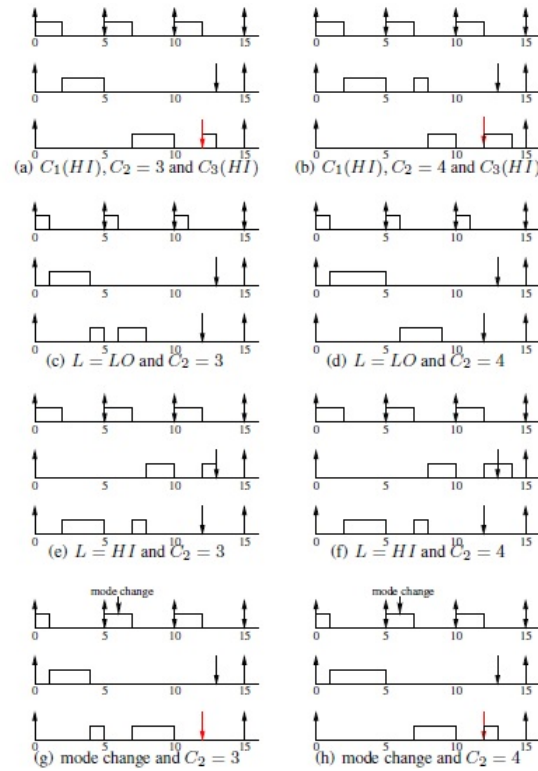
License  Creative Commons BY 3.0 Unported license
© Yasmina Abdedda

Keywords: fixed priority; probabilistic scheduling; mixed criticality

Extended Abstract

According to [1], the most effective fixed priority approach for scheduling mixed criticality systems is the Adaptive Mixed Criticality (AMC) approach. This approach uses the assumption that no low criticality task is released when the system moves to high criticality. Our goal is to propose an adaptive approach for a model where low criticality tasks have a probabilistic computation time [2]. When the systems moves to high criticality level, the set of low criticality tasks are not ignored but their tolerated probability deadline miss is modified. More formally, we consider a system defined as a set of probabilistic periodic real-time tasks $\{\tau_1, \dots, \tau_n\}$ having a certain level of criticality: high (HI) or low (LO). Each task τ_i is defined as a tuple (L_i, T_i, D_i, C_i) with $L_i \in \{LO, HI\}$ the criticality of the task, T_i its period, D_i its constrained deadline and C_i is its worst-case execution time discrete random variable. We consider that the random variables $C_i, i = 1, \dots, n$ are independent such that for every task τ_i :

1. If $L_i = LO$, the sample space of C_i is $\{C_i(1), \dots, C_i(m_i)\}$ and the probability distribution of C_i is the function $f_{C_i} : [1, m_i] \rightarrow \mathbb{N}^*$ with $\sum_{j=1}^{m_i} f_{C_i}(C_i(j)) = 1$.
2. If $L_i = HI$, the sample space of C_i is $\{C_i(LO), C_i(HI)\}$ with $0 < C_i(LO) \leq C_i(HI)$ and the probability distribution of C_i is a function $f_{C_i} : [LO, HI] \rightarrow \mathbb{N}$ with $f_{C_i}(x) = 1$ if $L = x$ and $f_{C_i}(x) = 0$ if $L \neq x$ where $L \in \{LO, HI\}$ is the criticality of the system.



■ **Figure 6** $\tau_1 = (HI, 5, 5, C_1)$, $\tau_2 = (LO, 15, 13, C_2)$, $\tau_3 = (HI, 15, 12, C_3)$, $C_1 = \{C_1(LO) = 1, C_1(HI) = 2\}$, $C_2 = \{3, 4\}$, $f_{C_2}(C_2 = 3) = f_{C_2}(C_2 = 4) = 0.5$, $C_3 = \{C_3(LO) = 3, C_3(HI) = 4\}$, $P^{LO} = 0$ and $P^{HI} = 0.5$. Priority order: τ_1, τ_2, τ_3 not feasible but feasible if $D_3 = 14$.

The system behaves as described below (see Figure 6):

1. At the beginning of the execution, the criticality of the system is $L = LO$, and if a task τ_i with $L_i = HI$ does not notify its completion after the execution of $C_i(LO)$ time unit, the criticality of the system moves from $L = LO$ to $L = HI$,
2. When $L = LO$: **(a)** a task τ_i is executed if it is active and no higher priority task is active, **(b)** the probability of a deadline miss of all high criticality tasks is 0 and the probability miss of all low criticality task is less then a constant P^{LO} .
3. When $L = HI$: **(a)** for every task τ_i , if $L_i = LO$, τ_i is executed if it is active and no higher criticality or priority task is active, and if $L_i = HI$, τ_i is executed if it is active and no high criticality task of higher priority is active, **(b)** the probability of a deadline miss of all high criticality tasks is 0 and the probability of a deadline miss of all low criticality tasks is less then $P^{HI} \geq P^{LO}$.


References

- 1 S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE RTSS*, 2011, pages 34–43.
- 2 D. Maxim and L. Cucu-Grosjean. *Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters*. In *Proceedings of the IEEE 34th Real-Time Systems Symposium, RTSS 2013*.

4.21 MC Scheduling on Varying-Speed Processors

Zhishan Guo (University of North Carolina at Chapel Hill, US)

zsguo@cs.unc.edu

License  Creative Commons BY 3.0 Unported license
© Zhishan Guo

Keywords: varying-speed processors, model combination

Introduction and Motivation

Most existing research on Mixed-Criticality (MC) scheduling (see [1] for a review) has focused on dealing with different WCET estimations of a single piece of code. This is typically a consequence of different tools for determining worst case execution time (WCET) bounds being more or less conservative than each other.

This narrative is now being repeated with respect to *processor speeds*. Modern powerful and energy-efficient processors are yielding innovations that result in varying speed during run-time. For example, [2] describes a mechanism such that late signals can be recovered by delaying the next clock tick, so that logical faults do not propagate to higher (i.e., the software) levels. In a Globally Asynchronous Locally Synchronous (GALS) circuit, local clocks can be affected by signals propagating between different synchronous modules in an asynchronous manner.

Research on such varying-speed platform may lead to better understanding of a wider range of problems. For example, in data communication of automobiles, aircrafts, or wireless sensor networks, time-sensitive data-streams must be transmitted over potentially faulty communication channels, where a high bandwidth is provided under most circumstances yet only guaranteeing a lower bandwidth.

Model and Existing work

A varying-speed processor is modeled as follows: under normal circumstances, it completes at least one unit of execution during each time unit, while it may fall into a degrade mode at any instant, during which it can only complete $x \in [s, 1)$ units of execution during each time unit, for some (known) threshold $s < 1$. It is not a priori known when, or whether, such degradation will occur. Similar to other MC scheduling problems, we seek a strategy that guarantees to complete all jobs by their deadlines under normal (LO-criticality) behaviors, while simultaneously guaranteeing to complete all HI-criticality jobs if either the platform (or the jobs) suffer from degradation (HI-criticality) behaviors. Note that here we are considering a combination of various aspects that MC may arise from, including periods, WCETs, processing speeds, etc.

Based upon the properties of the platform and the workload, we classify those problems into four categories:

1. Self-Monitoring: A self-monitoring (SM) processor immediately knows its execution speed during run-time⁵ while non-monitored (NM) one may not.
2. Number of processors: Either uniprocessor, or multiprocessor.
3. Workload model: One shot job set, or sporadic/periodic task set.
4. Single(S)- or Multiple(M)- Worst case execution time (WCET) per job.

⁵ Similar to Linux command `cpufreq-info`, SM platform has access to processor speeds, while NM processor may only identify degradation upon some job not signaling its finishing on time.

■ **Table 1** Existing work on scheduling MC sets on varying-speed uniprocessor.

–	Jobs & S-WCET	Tasks & S-WCET	Jobs & M-WCET	Tasks & M-WCET
SM Uniproc.	[3] [4] ⁶	[3] [4] ⁷ [8]	[5]	[8]
NM Uniproc.	[7]	[8]	[5]	[8]
SM Multiproc.	[6]	–	–	–
NM Multiproc.	–	– ⁸	–	– ⁵

Table 1 lists existing work on MC scheduling upon varying-speed platforms.

Further Directions

Most current work only deals with one-shot jobs or implicit-deadline sporadic tasks, and the generalization to constrained deadlines is not trivial. Also, as shown in Table 1, much remains to be done regarding multiprocessors – the degraded mode upon such platforms needs to be completely specified. If different processors are assumed to degrade to different speeds, the resulting degraded platform may become a heterogeneous one, for which the MC scheduling problem is totally open.

References

- 1 A. Burns and R. I. Davis. Mixed Criticality Systems – A Review. *Department of Computer Science, University of York, Report. Fourth edition, July 31, 2014.*
- 2 D. Bull, et al. A power-efficient 32b ARM ISA processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation. In *IEEE ISSCC 2010*, pages 284–285.
- 3 S. Baruah and Z. Guo. Mixed-criticality scheduling upon varying-speed processors. *IEEE RTSS 2013*.
- 4 Z. Guo and S. Baruah. Mixed-criticality scheduling upon varying-speed multiprocessors. *Leibniz Transactions on Embedded Systems*, 1(2): 3:1–3:19, 2014.
- 5 Z. Guo and S. Baruah. The concurrent consideration of uncertainty in WCETs and processor speeds in mixed-criticality systems. Under submission.
- 6 Z. Guo and S. Baruah. Mixed-criticality scheduling upon varying-speed multiprocessors. *IEEE DASC 2014*, pp. 237–244.
- 7 Z. Guo and S. Baruah. Mixed-criticality scheduling upon unmonitored unreliable processors. *SIES 2013*, pp. 161–167.
- 8 S. Baruah and Z. Guo. Scheduling Mixed-Criticality Implicit-Deadline Sporadic Task Systems upon a Varying-Speed Processor. *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pp. 31–40, 2014.

⁶ The strong NP-hardness of non-preemption scheduling under such case is also shown in [3] and [4].

⁷ Regarding scheduling tasks, [3] and [4] only provide necessary conditions and a sharing-based (fluid) scheduling scheme, which is not impractical due to too many preemptions.

⁸ We may model a NM varying-speed processor with the multi-WCET MC model, and apply some existing MC scheduling work, while being somewhat pessimism, which is similar as [8].

Mixed-criticality systems: different models for scheduling problems (open or not)

4.22 Speedup bounds for multiprocessor scheduling

Suzanne van der Ster (Vrije Universiteit Amsterdam, NL)

License  Creative Commons BY 3.0 Unported license
© Suzanne van der Ster

Introduction

When studying mixed-criticality (MC) task systems, we are interested in worst-case behaviors and determining feasibility. Since determining feasibility exactly is hard, we design approximate feasibility tests. If such a test returns “feasible”, the task system is guaranteed to be feasible on a processor running at speed γ , while if it returns “infeasible”, the task set is guaranteed to be infeasible when processed on a unit-speed processor. The factor γ is also called the speedup factor (also for scheduling algorithms corresponding to the feasibility test).

Known results

There are two main paradigms for scheduling task systems on multiprocessors: global and partitioned scheduling. In the former, all tasks can use all machines, and jobs can even be migrated from one machine to another. In the partitioned scheduling approach, each task has to be assigned to one of the machines such that all its jobs have to be executed on this specific machine. For MC sporadic task sets, the only known results on multiple machines are for 2-level implicit-deadline task sets, i.e., for task sets such that the period equals the relative deadline for all tasks. Those results are based on an earlier result for implicit-deadline task systems on a single machine. For single-machine scheduling, the algorithm EDF-VD (introduced in [1]) is a modification of the well-known EDF policy, where higher-criticality tasks are assigned tighter deadlines (that are called virtual deadlines), in order to be able to meet all their deadlines, even in case of a criticality switch. It was shown [2] that any feasible 2-level MC task system can be scheduled successfully by EDF-VD on a processor running at speed $4/3$.

This result is used in the partitioned scheduling policy in [3]. The algorithm given has a speedup for m machines of at most $8/3 - 4/3m$.

An alternative approach, only interesting from a theoretical point of view, is viewing the MC scheduling problem as a *VECTORSCHEDULING* problem (see [4] for a definition), where each dimension corresponds to a criticality level. For this problem, a PTAS exists, when the number of dimensions is a constant. Combining the PTAS with EDF-VD yields that any task system that is feasible on m unit-speed machines can be scheduled on m machines of speed $4/3 + \epsilon$. For global scheduling, the EDF-VD scheduling policy is combined with the fpEDF scheduling policy, designed for non-MC task systems. For the resulting global scheduling algorithm it is proven [3] that any 2-level implicit-deadline MC task system that is feasible on m unit-speed machines, can be scheduled on m machines running at speed $\sqrt{5} + 1$.

Open problems

- Extending results to more than two criticality levels. For a single processor, schedulability conditions for EDF-VD are known [1] and the question is how these can be incorporated into a partitioned or global scheduling algorithm for multiple processors.

- Extending results to different processor models, for instance unrelated machines. In [5], non-MC task systems are scheduled on unrelated machines with a speedup $8 + 2\sqrt{6} \approx 12.9$, via smart rounding of an integer linear program. An interesting question is if the ILP and the corresponding rounding procedure can be adjusted to accommodate schedulability conditions for MC task systems.

References

- 1 S. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. *In Proceedings of 19th Annual European Symposium on Algorithms (ESA)*, pp. 555–566, 2011.
- 2 S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, and A. Marchetti-Spaccamela. The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems. *In ECRTS*, 2012.
- 3 S. Baruah, B. Chattopadhyay, H. Li, and I. Shin. Mixed-criticality scheduling on multiprocessors. *Real-Time Systems* 50, 142–177, 2014.
- 4 C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM Journal on Computing* 33(4) 837–851, 2004.
- 5 A. Marchetti-Spaccamela, C. Rutten, S. van der Ster, and A. Wiese. Assigning sporadic tasks to unrelated machines. *Mathematical Programming*. DOI: 10.1007/s10107-014-0786-9

5 Working Groups

5.1 Report on Platforms and Experimental Evaluation

Robert I. Davis

Present: Sébastien Faucou, Leandro Indrusiak, Chris Gill, Gabe Parmer, Roman Obermaisser, Sebastian Stiller, Cristian Maxim, Jim Anderson, Albert Chen, Sophie Quinton, David Broman, Kai Lampka, Lothar Thiele

Benchmarks and workloads

Workloads

- Fudge factors relating measurements to execution time budget: Typically 20 to 50% for singlecore systems. Does this also make sense in multicore?
- How much bigger can C(HI) be than a ‘well’ measured C(LO) (that perhaps accounts for the paths through the code, but not variations due to HW)? Could we perhaps get an upper bound by turning the cache off?
- What type of systems offers a representative workload for MCS? Are UAVs a good candidate?

Benchmarks, WATERS workshop and Call to Action For RT Benchmarks

- Complaint: we need industrial benchmarks to design solutions to problems that would be of benefit to the industry. Using existing real code, even if it is not true level A code? Source code is fine, but should we also have benchmarks in the form of more abstract models.
- Do Mälardalen Benchmarks cover all the case-studies that we want? Is it possible to build realistic task sets from Mälardalen Benchmarks? Should we set up a set of different representative applications from the Mälardalen Benchmarks representative of cache access and memory footprint?

- A large goal here is to collect artifacts that are usable for experimental purposes by the community.
- If this is not possible, then perhaps we can create a set of these that might not functionally be interesting, but that maintain the interesting characteristics in terms of time/cache utilization/etc.
- What is the set of non-functional behaviors we care about? The top three are the cache usage, memory access patterns, and timing. Additional behaviors that would be nice down the line are synchronization/dependencies/system interactions.
- We want the benchmarks to be open and free.
- We need executable benchmarks: we want code that can be functionally irrelevant but which has realistic execution times, memory accesses, cache policies and ideally environment. Best case is that we have applications from industry. What about developing an obfuscation strategy? If this isn't possible, then we need a set of benchmarks that we can use to compare against each other, and seek industry blessing or modification afterwards.
- Papabench is a benchmark for the task models. Can we have a benchmark suite based on generating task models?
- Another idea, if we want more complicated tasks, perhaps we can run a few of the Mälardalen benchmarks composed sequentially to make at least temporally more interesting tasks. This might not be reasonable, but it might be reasonable to go to industry and get feedback on what we *should* do. Of course, this will not work for cache footprints.
- How can we generate task models for MC? Vestal's original paper seemed to have the WCET "fudge factors" between *around* 20% to 50%. Importantly, there are concrete examples in his paper, so we should heed those.

What do we need to do as a community?

- A call for benchmarks/artifacts/code/task models from the community. We can take this to industry and get their feedback. See the call in <http://waters2015.inria.fr/>, though the call for benchmarks should be community-wide and go beyond this venue.
- Should we have a MCBench workshop devoted to creating this benchmarking suite? Or should we fold this into an existing workshop like WMC?
- We want exemplars of different application scenarios. These are the end-to-end suites of software you'd see running on a real system. For example, think the collection of software required to run UAVs. These should be emphasized in any call for benchmarks.

Other Questions

- Is complete isolation needed (or even possible) between criticality levels? Answer: No.
- Criticality level similar to memory hierarchy (by going down a level, you have less confidence but more tasks/work/utilization)?
- Tackling the whole complexity on a simple platform is too difficult today?

Links

- TACLeBench: <http://tacle.knossosnet.gr/activities/taclebench>
- Debie: <http://www.irit.fr/wiki/doku.php?id=wtc:benchmarks:debie1>
- Mälardalen Benchmarks: <http://www.mrtc.mdh.se/projects/wcet/benchmarks.html>
- Papabench: http://www.irit.fr/recherches/ARCHI/MARCH/rubrique.php3?id_rubrique=

5.2 Report on WCET

Claire Maiza

Present: David Broman, Bjorn Lisper, Pontus Ekberg, Claire Maiza, Christine Rochange, Suzanne van der Ster, Liliana Cucu-grosjean, Jan Reineke, Pascal Richard, Sebastian Altmeyer.

In this subgroup, the idea was to discuss about worst-case execution time in the context of mixed-criticality. In the context of mixed-criticality systems, timing models at the scheduling phasis consider not only one guaranteed bound, but a set of execution time estimations. In this summary we first discuss where these different estimations come from, second we focus on mixed-criticality in multi-core systems and the specificities due to the timing interferences.

How to get different execution time estimations?

Note that as far as more than one estimation is considered, one can not name them “worst-case execution time”. The notion of an estimation which is supposed to be closer to the real execution time but not an upper-bound on all possible execution time is clearly not a “worst-case”.

We identified some sources of different execution time estimations:

- Due to the environment:

Using a static analysis, one usually look for a bound on the execution time for a specific “execution context”. The precision of this context may influence the execution time estimation. For instance, in automotive functionality may be developed for a large set of cars. However, once deployed, the specificities of the car in which the functionality is implemented could lead to a preciser estimation of the execution time.
- Due to the use of a margin:

In some companies, the WCET is measured or estimated and a large margin (e.g., a factor of 100) is applied to get an upper-bound on the execution time. In this case, the upper-bound is largely over-estimated, but may give the feeling of a more trustful bound...
- Due to the WCET analysis:

Timing analysis are based on three models: hardware, software and environnement. Different tools or analysis method could get different estimations due to the precision of these models and/or the uncertainty involved. For instance, a measurement-based timing analysis considers a subset of the hardware model states.

Multi-core context

In case of multi-core, the large set of possible interferences of one task execution on the execution of other ones, leads to a more complex notion of execution time estimation. Due to the complexity of an exhaustive analysis that would take into account all possible interferences, there is a usual tradeof between precision of the estimation and complexity of the analysis.

Some approaches try to get more precision by adapting the architecture. These approaches may try to get a multi-core platform that suffers less from interferences (predictable architecture) or to configure the architecture to get less interferences (e.g., partition). In this context, mixed-criticality is less an issue because the tasks with low-criticality should not influence the execution time of the high-criticality tasks.

When the platform is not designed to be predictable, execution time analysis may lead to a large set of different estimations. For instance, a bus analysis may consider a very large

guaranteed bound on the interferences or model precisely all possible accesses to the bus. In the first case, the bound should be over-estimated. In the second case, the complexity of the analysis might not scale real application size. That may be a reason for the need of different execution time bounds in the scheduling analysis. This lead to two open-questions: should WCET and scheduling analysis be one common analysis in the case of multi-core? Should the uncertainty in the multi-core hardware model lead to a new execution time analysis method?

5.3 Report on Criticality

Sanjoy K. Baruah

Present: Zoë Stephenson, Vincent Nelis, Joël Goossens, Sophie Quinton, Leen Stougie, Dorin Maxim, Alberto Marchetti-Spaccamela, Enrico Bini, Wang Yi, Marko Bertogna, Nathan Fisher, Gerhard Fohler, Emmanuel Grolleau, Zhishan Guo, Pengcheng Huang, Sanjoy K. Baruah.

Agenda

This subgroup was spawned off with a mandate to explore an agenda that includes the following issues

- Obtain a better understanding of the safety background that motivates consideration of criticality levels. Why do we even have criticality levels, what need do they address?
- Identify the role that the WCET concept plays in safety considerations in mixed-criticality systems. We should distinguish between WCET budgets used for runtime enforcement and mode changes, and WCET estimates which approximate the WCET with different levels of confidence. Distinguish between budgets and estimates.
- The above issues may help determine what characterizes a system as being a mixed-criticality one. Is it about where WCET values are usable or is it about having isolation/lack of interference or both?

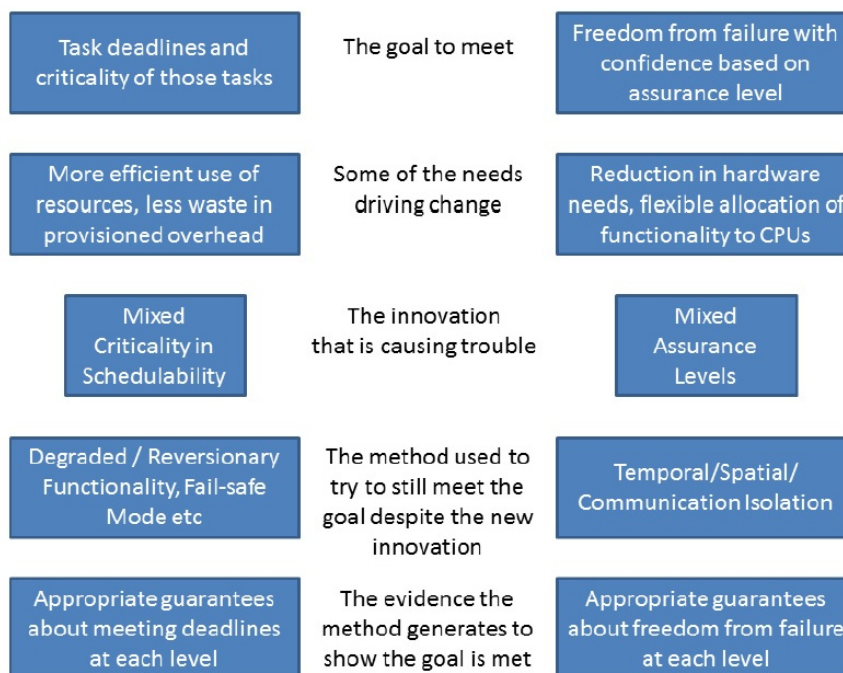
Discussions

1. Notions of criticality, as used in the research community, come from safety standards, e.g., IEC61508 and ISO26262. However, the use of some of the criticality-related terminology in the mixed-criticality systems (MCS) research community is not always consistent with their use in the standards (see Figure 7). It is incumbent on the research community to make an effort to familiarize practitioners with their research findings. Some possible avenues for achieving this were discussed:

- Issue is maybe of widening the scope of who gets involved with this work.
- We should speak of graceful degradation and fault tolerance rather than changing criticality.
- Mixed criticality in industry is currently mainly about isolation and separation; a significant portion of the research efforts are aimed at ensuring more efficient utilization of computational resources.

Where do criticality levels come from?

- In several application domains, criticality levels they are defined by standards
- The research community could think that criticality levels in MCS are related to those standards



■ **Figure 7** Mapping Research Concepts to Industrial Concerns.

2. ALARP – “As Low As Reasonably Practicable” – is a widely-adopted guiding principle in safety analysis for evaluating success in risk reduction. It is not expected that risk can be reduced to zero; nor it is desirable (cost-effective) to over-engineer for no tangible benefit. Neither is it desirable to miss out some risk area from analysis and mitigation.

It is important to be aware of these distinctions:

- *Safety* relates to inadvertent harm that a system can do. It is sufficiently safe if the risk of causing a hazard is reduced as low as reasonably practical.
- *Security* relates to intentional violation of access control – exposure of data through overt and covert channels, for example.
- *Surety* is not a term that is often used, it relates to having 100% confidence or 0% residual risk. Since there is always risk in the environment and in hardware failures, this induces a limit on the level of risk reduction for software that will ever be acceptable in practice. However, it is still important to reduce uncertainty in what the risks even are.

3. Industry often uses an isolation/ separation based approach to partition software of different assurance levels so that it can be known that there is sufficient freedom from interference with sufficient confidence. In order to justify dropping this approach, there would have to be a good reason to suffer the pain of arguing about why there is still sufficient freedom from interference with sufficient confidence. What would the gains be? – flexibility? Would it be possible to use current hardware for longer? Would it be feasible to reduce the confidence level with which one has to assess some kinds of interference?

4. Arguments were made in favor of the mixed-criticality approach advocated in the MCS research community vs. an isolation-based approach:

- Today there is a gap between actual and worst-case execution times requiring, especially in the case of isolation-based approaches, significant over-provisioning the computing resources.

- We can expect future architecture to increase this gap, will it be increased to the point of being unbearable?
 - If so, is the cost of loosening isolation worth the gain of computing resource utilization?
 - Other gains of mixed-criticality: dealing with different cases of uncertainty (not only WCET, but also periods, thermal aspects, etc.)
5. When we try to reason about uncertainty we need to be clear about how the standards relate aleatory uncertainty (e.g. MTBF of a hardware component) and epistemic uncertainty (e.g. I'm not 100% sure I got enough coverage in my testing).
6. MCS research and the certification process. Currently, correct by construction is the common way to demonstrate correctness for the purposes of obtaining certification. Evidence can be provided by analysis, but it is challenging to make this acceptable to certification authorities. The question was discussed: Can MCS research be used in certification? The following points were made:
- any new theory takes time to be accepted
 - perhaps we should be working on developing a theory that is ready to be applied whenever industry is ready
 - There was a discussion about how mixed-criticality is applicable or could be in the future in the industry: in mixed-criticality systems research there is room for every aspect: theory, operating systems, practical research more certification standard oriented, etc.

5.4 Report on Probabilistic Approaches

Liliana Cucu-Grosjean

Present: Arvind Easwaran, Zhishan Guo , Adriana Gogonel, Dorin Maxim, Sebastian Altmeyer, Yasmina Abdeddaim, Rob I. Davis, Liliana Cucu-Grosjean.

The discussions on probabilistic approaches took place during two time slots:

1. Following the presentations on probabilistic approaches, the first slot of discussions within this group has been dedicated to the application of Extreme Value Theory (EVT). This theory is used to solve the problem of estimating a probabilistic bound on all possible execution times, this bound is usually denoted by pWCET. We underline four different threads of discussions related to the utilization of pWCET to estimate WCET in the context of mixed criticality systems. Each thread had identified one or several open problems detailed below.
 - Currently the static analysis is extensively used to estimate the WCET. The users of static analysis need to understand the assumptions of EVT in order to use it while WCET estimating, but also to compare against state of the art approaches.
 - An important effort of popularization is necessary in order to increase the understanding of the steps of EVT when applied to the problem of estimating the WCET.
 - Today the differences between functional independence, probabilistic independence and statistical independence are not well understood by the community and this has a direct impact on the overall understanding of this method.
 - Once a pWCET is estimated, how do we calculate the probability of more than one overrun of C(LO) in a given time? It is generally admitted that an overrun never appears alone and that it is usually related to other possible overruns.

2. The second slot of discussions has been concentrated on the understanding how a probability distribution of a WCET defines different criticalities ?


Three different models have been identified as follows.

- A first model that associates to each level of criticality a pair (value for the WCET, probability of appearance of that value). For instance in the Vestal model this could correspond to a random variable with three possible values $C(LO)$, $C(HI)$ and ∞ .
- A second model that associates to each level of criticality a random variable describing the pWCET. For instance in the Vestal model this could correspond to $C(LO)$ ⁹ and $C(HI)$ where $C(HI) = C(LO) + constant$.
- A third model that associates to the highest level of criticality an unique WCET (that could be obtained using static analysis for instance) and to the lowest level of criticality a pWCET.

6 Open Problems

6.1 Unification of mixed criticalities, WCET, and probabilistic execution time

Enrico Bini (Scuola Superiore Sant'Anna, Pisa, IT)

License  Creative Commons BY 3.0 Unported license
© Enrico Bini

- I have no experience with mixed-criticality systems
- I believe that some concepts we have been listening about
 - mixed-criticality
 - probabilistic exec time
 - mode change
 do overlap significantly
- This presentation is an attempt to relate them with each other
- It may well be something very obvious to you (especially timing analysis people).

Execution time

What does the sequence of job execution times¹⁰ depend on?

- Let Ω be the sample space (input data, machine type, cache status, alpha particles flipping bits, etc.)
- $\omega \in \Omega$ is an event
- execution time is $c : \Omega \rightarrow \mathbb{R}$
- “Worst-case execution time”

$$C_{WCET}(\Omega) = \sup_{\omega \in \Omega} c(\omega)$$

⁹ We use calligraphic letters to denote random variables.

¹⁰ next arguments are valid for any task parameter

Criticality

- Sanjoy: criticality is a desired “level of assurance”
- It seems that “criticality” are then just subsets of Ω

$$\text{LO} \subseteq \text{HI} \subseteq \Omega \quad (3)$$

- Then, for any criticality level $\mathcal{L} \subseteq \Omega$, the corresponding \mathcal{L} -WCET is

$$C_{\text{WCET}}(\mathcal{L}) = \sup_{\omega \in \mathcal{L}} c(\omega)$$

- Notice that (3) implies

$$C_{\text{WCET}}(\text{HI}) \geq C_{\text{WCET}}(\text{LO})$$

- The partial ordering of set inclusion over Ω also induces a partial ordering of the criticalities

Property of criticality

One possible property of criticality:

- Let us have a chain of criticality levels

$$\mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \dots \subseteq \mathcal{L}_n$$

- Is an event $\omega' \in \mathcal{L}_{i+1} \setminus \mathcal{L}_i$ “worse” than any event $\omega \in \mathcal{L}_i$? This is reasonable to expect
- **Property 1 (monotonicity of $c(\cdot)$ over crit)**. If an event $\omega' \in \mathcal{L}_{i+1} \setminus \mathcal{L}_i$ “worse” than any event $\omega \in \mathcal{L}_i$?, then

$$\forall i = 1, \dots, n-1, \forall \omega' \in \mathcal{L}_{i+1} \setminus \mathcal{L}_i, \forall \omega \in \mathcal{L}_i, \quad c(\omega') \geq c(\omega)$$

Probability

- If Ω is equipped with a probability measure P , then $c : \Omega \rightarrow \mathbb{R}$ becomes a random variable
 - its cumulative distribution function ($\text{cdf}(x)$) is

$$\text{cdf}(x) = P(c \leq x) = P(\{\omega \in \Omega : c(\omega) \leq x\})$$

- $P(\text{HI})$ is then the probability that the system belongs to the criticality $\text{HI} \subseteq \Omega$;
- $P(\text{HI}) \geq P(\text{LO})$
- How is the measure P defined? I don’t know
 - it has to do with the probability of the input values, probability of being in some status, etc.

Criticality & Probability

- Given
 - a probability measure P over Ω ,
 - a criticality $\mathcal{L} \subseteq \Omega$ with $P(\mathcal{L}) \neq 0$, and
 - the computation time function $c : \Omega \rightarrow \mathbb{R}$
- we can define the *conditional probability* any event $A \subseteq \Omega$ given \mathcal{L} as

$$P(A|\mathcal{L}) = \frac{P(A \cap \mathcal{L})}{P(\mathcal{L})}$$

- the conditional random variable $c : \Omega \rightarrow \mathbb{R}$ given \mathcal{L} , has

$$\text{cdf}(x) = \frac{\{c(\omega) \leq x\} \cap \mathcal{L}}{P(\mathcal{L})}$$

Average execution time, with criticality \mathcal{L}

- the average execution time is

$$C_{\text{avg}}(\mathcal{L}) = E[c | \mathcal{L}] = \frac{1}{P(\mathcal{L})} \int_{\mathcal{L}} c(\omega) dP(\omega)$$

- nice property (maybe proved on the blackboard) is:
given $\text{LO} \subseteq \text{HI} \subseteq \Omega$, then

$$C_{\text{avg}}(\text{HI}) < C_{\text{avg}}(\text{LO}) \quad \Leftrightarrow \quad C_{\text{avg}}(\text{HI} \setminus \text{LO}) < C_{\text{avg}}(\text{LO})$$

- however of Property “monotonicity over crit” holds, then

$$\forall \text{LO} \subseteq \text{HI} \subseteq \Omega, \quad C_{\text{avg}}(\text{HI}) \geq C_{\text{avg}}(\text{LO})$$

7 New collaborations

7.1 Providing Weakly-Hard Guarantees for Mixed-Criticality Systems

Robert I. Davis (Real-Time Systems Research Group, Department of Computer Science, University of York, UK and AOSTE team, Inria Paris-Rocquencourt, FR)

Sophie Quinton (SPADES team, Inria Grenoble – Rhône-Alpes, FR)

Mixed Criticality Systems are systems running applications of different criticality levels [1]. Often only two criticality levels are considered, denoted LO-criticality and HI-criticality respectively. According to the definition most widely accepted by the research community, usually called the Vestal model, tasks are expected to run in normal mode as specified by their LO-criticality model (which is based on somewhat optimistic parameters) so that all task requirements are satisfied. In addition, one must consider the possibility for tasks to run out of the bounds defined by their LO-criticality parameters in degraded mode, following their HI-criticality model. In that case requirements for the HI-criticality tasks must remain satisfied but requirements for the LO-criticality tasks are dropped.

One criticism that is often made of this approach is that it is not realistic to consider that LO-criticality tasks may be dropped, even in a context where the safety of HI-criticality tasks may be at risk. We are interested here in how weakly-hard guarantees [2] (i.e. having to meet m out of k deadlines rather than all of them) can be used to avoid dropping LO-criticality tasks entirely. The simplest scenario that can be envisioned is that LO-criticality tasks have to meet all deadlines in normal mode, but have weakly-hard constraints in degraded mode, while HI-criticality tasks have to meet all deadlines (i.e. hard constraints) in both modes. The rationale behind this is that control algorithms can often tolerate some jobs missing their deadlines or not executing, but then need to guarantee that a number of jobs will meet their deadlines so that the system returns to a stable state [4]. A key consequence of using weakly-hard constraints is that this may allow postponement of the change in scheduling policy resulting from a switch from normal to degraded mode.

We believe that introducing weakly-hard constraints into the mixed criticality model might help increase the acceptance of the latter in industry. Note that various other scenarios

are interesting as well. For example we could consider that HI-criticality tasks have hard deadlines while LO-criticality tasks always have weakly-hard constraints (maybe weaker ones in degraded mode). Alternatively, all tasks could have weakly-hard constraints in both modes. Again in that case weakly-hard constraints may allow postponement of the change in scheduling policy: a HI-criticality task could be aborted rather than exceed its LO-criticality execution time.

We aim to collaborate on research integrating the concept of weakly-hard constraints into Mixed Criticality Systems. In the first instance, we will explore how these constraints can be incorporated into the Adaptive Mixed Criticality scheduling policy and analysis proposed by Baruah et al. [1].

References

- 1 Sanjoy K Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34–43. IEEE, 2011.
- 2 Guillem Bernat, Alan Burns, and Albert Llamosí. Weakly hard real-time systems. *IEEE Trans. Computers*, 50(4):308–321, 2001.
- 3 Alan Burns and Robert I. Davis. Mixed criticality systems – a review. <http://www-users.cs.york.ac.uk/burns/review.pdf>.
- 4 Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 53–62, 2014.

7.2 A Multicore Response Time Analysis Framework

Sebastian Altmeyer (University of Amsterdam, NL)

Robert I. Davis (Real-Time Systems Research Group, Department of Computer Science, University of York, UK and AOSTE team, Inria Paris-Rocquencourt, France)

Leandro Indrusiak (University of York, GB)

Claire Maiza (VERIMAG – Gières, FR)

Vincent Nelis (The Polytechnic Institute of Porto, PT)

Jan Reineke (Universität des Saarlandes, DE)

In this paper, we introduce a Multicore Response Time Analysis (MRTA) framework. This framework is extensible to different multicore architectures, with various types and arrangements of local memory, and different arbitration policies for the common interconnects. We instantiate the framework for single level local data and instruction memories (cache or scratchpads), for a variety of memory bus arbitration policies, including: Round-Robin, FIFO, Fixed Priority, Processor Priority, and TDMA, and account for DRAM refreshes. The MRTA framework provides a general approach to timing verification for multicore systems that is parametric in the hardware configuration and so can be used at the architectural design stage to compare the guaranteed levels of performance that can be obtained with different hardware configurations. The MRTA framework decouples response time analysis from a reliance on context independent WCET values. Instead the analysis formulates response times directly from the demands on different hardware resources.

7.3 Mixed criticality support for automotive embedded systems

Yasmina Abdeddaim (Université Paris-Est, LIGM UMR CNRS 8049, ESIEE Paris, FR)

Sébastien Faucou (University of Nantes, FR)

Emmanuel Grolleau (ENSMA – Chasseneuil, FR)

On the subject of probabilistic analysis on mixed criticality systems when some criticality levels have deterministic constraints and parameters descriptions, while other criticality levels allow for a certain probability of failure and hence can be modeled and analyzed probabilistically.

Participants

- Yasmina Abdeddaim
ESIEE – Noisy le Grand, FR
- Sebastian Altmeyer
University of Amsterdam, NL
- James H. Anderson
University of North Carolina –
Chapel Hill, US
- Sanjoy K. Baruah
University of North Carolina –
Chapel Hill, US
- Marko Bertogna
University of Modena, IT
- Enrico Bini
Scuola Superiore Sant’Anna –
Pisa, IT
- Björn B. Brandenburg
MPI-SWS – Kaiserslautern, DE
- David Broman
KTH Royal Institute of
Technology, SE
- Alan Burns
University of York, GB
- Albert Cohen
ENS – Paris, FR
- Liliana Cucu-Grosjean
INRIA – Le Chesnay, FR
- Robert I. Davis
University of York, GB
- Arvind Easwaran
Nanyang TU – Singapore, SG
- Pontus Ekberg
Uppsala University, SE
- Rolf Ernst
TU Braunschweig, DE
- Sébastien Faucou
University of Nantes, FR
- Nathan Fisher
Wayne State University, US
- Gerhard Fohler
TU Kaiserslautern, DE
- Christopher D. Gill
Washington University –
St. Louis, US
- Adriana Gogonel
INRIA – Le Chesnay, FR
- Joel Goossens
Free University of Brussels, BE
- Emmanuel Grolleau
ENSMA – Chasseneuil, FR
- Zhishan Guo
University of North Carolina –
Chapel Hill, US
- Pengcheng Huang
ETH Zürich, CH
- Leandro Soares Indrusiak
University of York, GB
- Kai Lampka
Uppsala University, SE
- Björn Lisper
Mälardalen University –
Västerås, SE
- Claire Maiza
VERIMAG – Gières, FR
- Alberto Marchetti-Spaccamela
University of Rome
“La Sapienza” IT
- Cristian Maxim
Airbus S.A.S. – Toulouse, FR
- Dorin Maxim
The Polytechnic Institute of
Porto, PT
- Vincent Nelis
The Polytechnic Institute of
Porto, PT
- Roman Obermaisser
Universität Siegen, DE
- Gabriel Parmer
George Washington University –
Washington, US
- Sophie Quinton
INRIA – Grenoble, FR
- Jan Reineke
Universität des Saarlandes, DE
- Pascal Richard
ENSMA – Chasseneuil, FR
- Christine Rochange
Paul Sabatier University –
Toulouse, FR
- Zoe Stephenson
Rapita Systems Ltd. – York, GB
- Sebastian Stiller
TU Berlin, DE
- Leen Stougie
CWI – Amsterdam, NL
- Lothar Thiele
ETH Zürich, CH
- Suzanne van der Ster
VU University of Amsterdam, NL
- Wang Yi
Uppsala University, SE

