Work-in-Progress and Demo Proceedings - 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)

Vincent Nélis, Sophie Quinton

2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)

Work-in-Progress and Demo Proceedings

Vienna, Austria

April 11-14, 2016

http://2016.rtas.org/work-in-progress/
http://2016.rtas.org/demos/

Edited by Vincent Nélis and Sophie Quinton

Message from the Work-in-Progress and Demo chairs

Dear colleagues,

Welcome to Vienna, and to the Work-in-Progress (WiP) and Demo Session of the 22nd IEEE Real-Time Embedded Technology and Applications Symposium (RTAS'16). This session is dedicated to new and on-going research as well as to work with concrete systems, tools and prototypes in the field of real-time and embedded systems. We are happy to present 12 WiP papers and 8 demos that cover innovative research and experiments from a wide range of topics, including real-time scheduling, cache contention, real-time issues related to COTS multicore platforms, and runtime monitoring.

The main goal of the WiP session is to provide researchers with an opportunity to discuss evolving and early-stage ideas, and solicit feedback from the real-time systems community at large. Similarly, the demo session offers a forum such that researchers can give a demonstration of and get feedback about their work with concrete systems, tools and prototypes in all areas of real-time embedded technology and applications. The presentations can only give a brief overview of the research approaches chosen and of the work achieved by the speakers introducing the selected contributions. We hope that these presentations will encourage you to ask questions, share your ideas, and provide valuable feedback to the authors during the poster and demo sessions that will follow. We would like to emphasize that stimulating discussions are the most important feature of the WiP and demo session.

We would like to thank the members of the WiP and demo session technical program committees for their help in publicizing the session and reviewing the papers. We would also like to thank the authors for their interesting contributions and their choice of RTAS as a means to share and improve their research. Last but not least, our special gratitude goes to the RTAS'16 program chair, Rob Davis, for his help and support.

We hope that you will enjoy the Work-in-Progress and Demos of RTAS 2016!

Vincent Nélis, CISTER/INESC TEC and ISEP, Portugal, Work-in-Progress chair
Sophie Quinton, Inria Grenoble Rhône-Alpes, France, Demo chair
RTAS 2016

Program Committees

Work-in-Progress

Chaired by Vincent Nélis, CISTER/INESC TEC and ISEP, Portugal.

Borislav Nikolic	CISTER/INESC TEC and ISEP, Portugal
Björn Brandenburg	Max Planck Institute for Software Systems, Germany
David Bol	Microelectronics laboratory – ICTEAM institute, Université Catholique de Louvain, Belgium
Benny Åkesson	CISTER/INESC TEC and ISEP, Portugal
Leandro Indrusiak	University of York, U.K.
Andrea Marongiu	Integrated Systems Laboratory, ETH, Swiss
Paolo Burgio	University of Modena, Italy
Dakshina Dasari	Research and Technology Centre at Robert Bosch, India
Gurulingesh Raravi	Distributed and Mobile Computing group, in Xerox Research Center India
Mircea Negrean	IAV GmbH, Germany

Demos

Chaired by Sophie Quinton, Inria Grenoble Rhône-Alpes, France.

Luís Almeida	University of Porto, Portugal
Loïc Fejoz	RealTime-at-Work, France
Daniel Lohmann	Friedrich-Alexander-Universitt Erlangen-Nrnberg, Germany
Martina Maggio	Lund University, Sweden
Gabriel Parmer	George Washington University, USA
Insik Shin	KAIST, Korea
Marcus Völp	University of Luxemburg, Luxemburg
Dirk Ziegenbein	Bosch GmbH, Germany

Table of Contents

Work-in-Progress

Demos

Work-in-Progress Papers

Towards Parallelizing Legacy Embedded Control Software Using the LET Programming Paradigm

Julien Hennig, Hermann v. Hasseln, Hassan Mohammad
Daimler AG
Email: {julien.hennig, hermann.v.hasseln, hassan.mohammad}
@daimler.com

Stefan Resmerita, Stefan Lukesch, Andreas Naderlinger
Department of Computer Sciences, University of Salzburg
Email: {stefan.resmerita, stefan.lukesch, andreas.naderlinger}
@cs.uni-salzburg.at

Abstract—**The growing demand for computing power in automotive applications can only be satisfied by embedded multi-core processors. Significant parts of such applications include OEM-owned legacy software, which has been developed for single-core platforms. While the OEM is faced with the issues of parallelizing the software and specifying the requirements to the ECU supplier, the latter has to deal with implementing the required parallelization within the integrated system. The Logical Execution Time (LET) paradigm addresses these concerns in a clear conceptual framework. We present here initial steps for applying the LET model in this respect: (1) Parallelization of legacy embedded control software, by exploiting existing inherent parallelism. The application software remains unchanged, as adaptations are only made to the middleware. (2) Using the LET programming model to ensure that the parallelized software has a correct functional and temporal behavior. The Timing Definition Language (TDL) and associated tools are employed to specify LET-based requirements, and to generate system components that ensure LET behavior. The work describes two conceptual ways for integrating TDL components in AUTOSAR.**

I. Introduction

By now the multi-core revolution has hit embedded computing full force. Most embedded microcontroller manufacturers offer multi-core based architectures as part of their mid and high-end lineups. As was the case for mainstream computing, porting, transforming and rewriting substantial legacy software to make effective use of the new processors' parallelism is trailing the advent of the new architectures. Parallelizing proven-in-use time-critical embedded control software is posing significant challenges that call for rigorous software design patterns and programming models. For a number of years, we participated in industrial development projects implementing real-time control applications for first-of-their-kind multi-core-based electronic control units (ECUs) in the chassis, driver-assistance and powertrain domains. We saw two distinct activities involved in such implementation efforts. The first activity is to expose parallelism in the legacy single-core application software by transforming the code to eliminate dependencies between individual functions. A functional redesign and parallelization of compute intensive parts is ideally avoided and only attempted when resource constraints demand even more parallelism. The second activity is to implement the exposed parallelism using specific implementation patterns that guarantee time- and value-deterministic parallel execution on the targeted multi-core processor. We observed that most of these patterns had coordination regimes in common where state variables are updated and buffered at controlled instances in time, typically at the beginning or at the end of periodic tasks. These patterns were typically deeply project and supplier-specific and often lacked formal foundations. An automotive OEM has to consider that significant amounts of OEM-owned application software is integrated as part of a supplier-owned execution platform. It is mandatory for the OEM that changing a supplier does not result in having to redesign the application software due to supplier-specific implementation patterns. Therefore, when we found that many implementation patterns shared key concepts of the LET programming model [1] we set out to evaluate seriously whether (1) LET could be an efficient and effective basis for implementing parallel execution of periodic control functions and (2) whether such an implementation could be integrated with the runtime facilities of an important automotive software standard AUTOSAR [2]. If successful, the LET paradigm would also offer the added advantage to extend naturally to the coordination of time triggered event chains which are distributed across multiple interconnected ECUs. Predictable behavior of such distributed event chains is becoming more and more important with the growing sophistication of vehicle control functions.

The main contribution of this paper thus consists in describing first significant steps towards LET implementations for multi-core architectures using facilities of the automotive industry standard AUTOSAR. After describing a legacy powertrain control application, we sketch our approach to expose its parallelism. Then we outline the steps taken to implement the exposed parallelism using the Timing Definition Language (TDL) as an implementation of the LET programming model. We conclude with some preliminary results and a summary of next steps.

II. Related Work

The quest for achieving predictable behavior in embedded systems is not new [3]. Embedded multi-core processors aggravate this problem to the point where rigorous restrictions at all levels of system design are required in order to have any hope for success [4]. The LET paradigm has been designed from the beginning to be a programming model that provides a basis for predictable behavior both at the design and at the implementation level [5]. The LET paradigm was and typically still is met with healthy skepticism by the real-time and embedded community because of its underlying restrictions concerning general expressiveness and WCET estimate requirements and the hard to afford computational costs regarding buffer space and execution time in particular. But a recent application of the LET programming model to an industrial engine control software for a single-core execution platform has shown that these overhead costs can be controlled [6]. [6] also illustrates that a process for applying the LET paradigm exists by which legacy software can be transformed iteratively. At the same time, LET gains popularity again as a design principle to analyze and engineer real-time control systems [7]. Existing AUTOSAR timing services are still insufficient to support the LET model. Elements of LET were behind an early AUTOSAR concept proposal "Support for Predictable Software Execution" [8] but this concept eventually didn't receive enough support to be pursued further. If the experiment we describe in this paper is successful, we hope to revive this AUTOSAR community effort.

III. Powertrain Control Case Study

The subject of our LET based parallelization effort is the central coordinator of an electric powertrain. Its core responsibility is command and control of a predictive operating strategy for the components of an electric powertrain: inverter, battery, auxiliaries, vehicle interface. A supplier is responsible for the ECU hardware and the AUTOSAR basic software (BSW) as well as for functions that directly interface with the ECU hardware. The supplier also provides an AUTOSAR compliant execution environment for integrating the application software (ASW). The ASW itself is developed by Mercedes-Benz and consists mainly of torque coordination, energy management, thermal management, auxiliary management, prediction and monitoring. The C functions that implement these features are called from a single fixed-period OS task. A reduced call rate is hard-coded in the OS task for each function whose period is a multiple of the base period. At the beginning of the task, task-external input from sensors and from the attached networks is processed, then the application functions are executed in a predefined static order, and finally output signals are processed and propagated either to their respective actuators or to the network. I/O itself is carefully coordinated either synchronously (polling) or asynchronously (interrupt based). This is a proven and pervasive software execution pattern for single-core embedded control units from which we would not want to deviate were it not for the fact that in the not too distant future single-core performance will no longer suffice for implementing advanced and innovative powertrain control features. We are thus faced with the typical exercise of parallelizing legacy software: partition the main task into independent subtasks which can be executed in parallel without affecting the overall functional correctness that is partly incorporated in the static execution order and the explicit or implicit dependencies between the individual functions.

IV. Towards Parallelization

To achieve the most possible parallelization in our legcy software, we present a method which is mainly an application of [9]. Neither the functional architecture nor the software architecture of the legacy code was designed to support a distribution on cores of a multi-core processor. Nevertheless, since the code consists of a large number of 'elementary' software modules, or runnables, the dataflow architecture should be rich enough in structure to identify sufficient parallelism among the runnables to foster a possible distribution on cores. The dataflow architecture is represented as a graph, consisting of nodes identified as the runnables, and directed edges identified as global variables for data exchange among runnables. Assuming that runnables are mapped onto periodic tasks for implementation, forward dependencies represent data exchanges updated in the same cycle of the task, whereas backward dependencies represent variables updated during the next cycle of the task. The goal is to maximize inherent parallelism by identifying subsets ('clusters') of totally independent runnables. Manipulation of the graph are made possible, by allowing backward dependencies to be changed to forward dependencies while maintaining all forward dependencies. Concretely, the method consists of the following three steps (cf. Fig. 1):

1.) For the set of runnables mapped onto a periodic task, the corresponding dataflow graph with forward and backward dependencies has to be identified.

2.) In application of the procedure given in [9], with the additional requirement that there are no forward and no backward dependencies among runnables of a cluster, we start with the identification of 'starting nodes' of the data flow graph. A starting node is a node with no dependent predecessor nodes. After this first step, all edges from the runnables in this cluster are removed such that a set of new starting nodes emerge. This is continued until all nodes have been visited. From inspection of the first part of Fig. 1 we see that runnables R1, R2 and R7 satisfy our conditions and define therefore our first cluster. In the next step we remove all edges from runnables R1, R2 and R7, and find the next subset of starting nodes. These are runnables R3 and R9. Going to all nodes we eventually arrive at a clustering shown in the second part of Fig. 1. This procedure can of course be done manually as well as automatically.

3.) Since the runnables in each cluster are totally independent, the chronological sequence in which they are executed is arbitrary, and in particular they can be executed in parallel. These sets are then subject to distribution on cores of a multi-core processor, as shown in the third part of Fig. 1, where a distribution on two cores is shown. This distribution of the independent runnables should in further steps be subject to more considerations, e.g., measured executions times of the runnables on a certain processor, number of dependencies among different cluster and thus generated cross-core overhead, functional and non-functional requirements, and further system-level considerations.

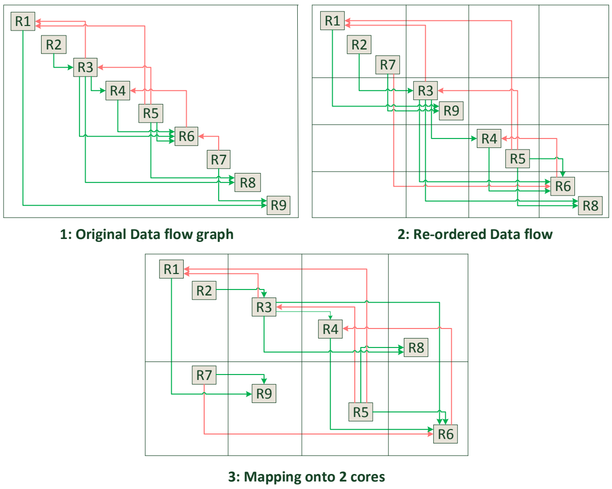


Figure 1. Parallelization with Data Flow Graph

V. TDL for AUTOSAR Parallel Legacy Software

TDL allows specification and implementation of timing properties of real-time applications according to the Logical Execution Time (LET) paradigm. In the LET programming model, a fixed logical duration is associated to each execution of a computational unit, or task [1]. The inputs of the task's execution are those available at the LET start and the outputs of the task's execution are made available at the LET end. Thus, the LET model achieves a pre-specified, platform-independent observable temporal behavior of a set of software functions, leading to both time and value determinism [5].

The top-level unit in TDL is called a module, which contains declarations of sensors, actuators, tasks, and modes. Sensors and actuators model data sources and sinks, respectively; they are employed to communicate with the environment. A task declaration specifies an application function as well as corresponding input and output ports. A mode is a periodic sequence of activities: task LET instances, actuator updates, and mode switches. Mode activities are carried out by a runtime system (virtual machine) called E-machine. More details about TDL can be found in [10].

A. *TDL Modeling of Legacy Systems*

TDL is accompanied by a commercial tool suite that can be integrated in top-down, model-based development processes, as well as in bottom-up, legacy-based development. The tool suite supports four development stages, which are described for the legacy case below.

1) *Programming:* The TDL program can be automatically generated from legacy information about the application functions associated with TDL tasks such as: connectivity, execution sequencing, periodicity, execution time (WCRT), and distribution on cores in a multi-core system, or on nodes in a distributed system. The resulting TDL program satisfies the legacy constraints (e.g., execution sequencing is preserved) and may include default LETs generated according to a user-specified policy (e.g., LET=WCRT). The user may then manually adjust LET values, e.g., by increasing some LETs in order to increase robustness against future additions of new functionality.

2) *Distribution and scheduling:* Mapping of TDL modules to computational nodes and cores is performed. Network communication schedules can also be generated in this step.

3) *Code generation:* The following runtime components are generated.

- The timing code, also called E-code, is compiled from the TDL program. The E-code is interpreted by the E-machine at runtime, triggering executions of LET start and LET end operations.

- Functions for implementing LET start and LET end operations for each TDL task, also called LET drivers. In order to achieve the LET data transfer semantics, certain legacy variables may need to be buffered. Input variables are buffered at the LET start of a task, while output variables are buffered throughout the execution and updated at the LET end. Optimization algorithms are put in place for minimizing the buffering.

- Setter/getter functions for the legacy variables that are subject to buffering requirements.

4) *Integration:* The integration of TDL components affects all levels of the legacy system.

- The E-machine employs some timing measurement capability of the hardware - a programmable timer, typically. The E-machine is executed within an interrupt service routine triggered by the timer.

- LET drivers are included in dedicated high-priority OS tasks, synchronized with the E-machine. Each LET driver task is also synchronized with OS tasks containing TDL-modeled legacy functions.

- The generated setter/getter functions are integrated at the application level.

B. *TDL Modeling of Parallelized Powertrain Control Software*

Consider a legacy application with a parallelization structure as described in section IV. Assume that a cluster contains runnables with the same execution period (otherwise split the clusters accordingly). We propose the following TDL-based procedure for deploying the application on a platform with n cores. First, assign to each cluster a number of n TDL tasks (one per core) with the same LET, then group the runnables of the cluster into n sequences and associate each sequence to a TDL task. If execution time information is available per runnable, then this grouping can be done such that the resultant LET value is minimized, which means core load is balanced within the LET and data propagation delay is minimized. In general, other (non-TDL) criteria may be considered in the assignment of runnables to cores - for example, one may wish to reduce inter-core communication by allocating runnables from different clusters with intensive data transfer to the same core.

The TDL program is generated such that the task's LETs are sequenced according to data dependencies between clusters. A TDL module contains then all the TDL tasks allocated

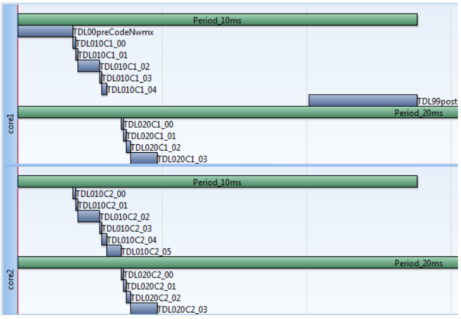


Figure 2. Visualization of a TDL program example

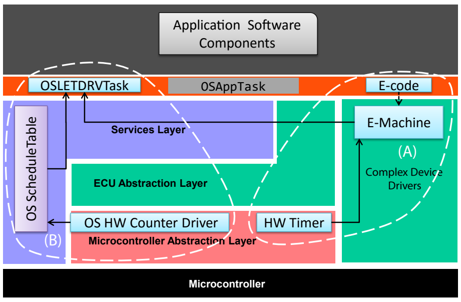


Figure 3. Integration of TDL in AUTOSAR with the two alternatives for the E-machine

to the same core. In the second stage, the module-to-core mapping is done by specifying the actual cores. In the third stage, a buffer analysis step determines a minimal number of legacy variables that must be buffered. In general, this is the case for communication variables between concurrent TDL tasks with overlapping LETs. Fig. 2 shows an extract from a TDL program, consisting of TDL tasks with periods 10ms and 20ms. The full program has 31 TDL tasks mapped to two cores, with $LET = 3 \cdot WCET$, plus rounding for alignment. The number of variables for inter-task communication exceeds 1500, of which only 7 need to be buffered.

In the fourth stage, integration of setter/getter functions is easily done by re-defining the implementations of ports and connections between AUTOSAR software components. LET drivers and E-code are included in the RTE. AUTOSAR OS events are employed for synchronization between the E-machine, the LET driver OS tasks, and the legacy OS tasks.

We are currently investigating two ways of integrating the E-machine into an AUTOSAR environment: (A) as a complex device driver (CDD) and (B) via OS schedule tables - see Fig. 3. The E-machine as CDD represents an additional basic software component that is able to deal with TDL programs of arbitrary complexity (in terms of modal structure and number of TDL tasks). The E-machine as OS schedule tables requires no additional AUTOSAR service or interface, but scalability is a concern that needs to be further investigated.

VI. Conclusion and Future Work

This paper describes an approach for distributing single-core legacy software on a multi-core platform, where the application source code is kept unchanged by using runnables as the granularity for distribution. In order to ensure functional correctness, dataflow constraints across parallel execution threads are guaranteed by employing a multi-core implementation of the LET programming model. This is provided by the Timing Definition Language (TDL) and its tool suite. First analysis results indicate that the TDL overhead can be controlled to acceptable levels. Furthermore, we describe primary concepts for integrating the LET paradigm within AUTOSAR.

In the next steps, we will evaluate TDL runtime overhead and resource usage on a prototype system. We will investigate policies for choosing LET values, and for dealing with LET exceptions (violations of LET specifications). Further research will extend the approach to distributed functions, with components running on different nodes of a network (e.g., FlexRay or Ethernet).

Special attention will be dedicated to paving the way for LET standardization as part of AUTOSAR. We consider that there is enough room for different LET implementation techniques, in keeping with the AUTOSAR motto ”cooperate on standards, compete on implementation”. The main challenge is to gather convincing evidence that LET addresses a variety of use cases, involving different HW configurations (single- and multi-core, networked ECUs), software development processes (model-based design, including legacy software, simulation and testing, debugging), and industry players (OEMs, system suppliers, HW vendors).

References

[1] T. Henzinger, B. Horowitz, and C. Kirsch, “Giotto: A time-triggered language for embedded programming,” *Proceedings of the IEEE*, vol. 91, pp. 84–99, January 2003.

[2] AUTOSAR, “AUTomotive Open System ARchitecture,” http://www.autosar.org/, accessed: 2016-01-11.

[3] P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, N. Guan, B. Jonsson, P. Marwedel, J. Reineke, C. Rochange *et al.*, “Building timing predictable embedded systems,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4, p. 82, 2014.

[4] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, and R. Wilhelm, “Predictability considerations in the design of multi-core embedded systems,” *Proceedings of Embedded Real Time Software and Systems*, pp. 36–42, 2010.

[5] T. A. Henzinger, “Two challenges in embedded systems design: predictability and robustness,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1881, pp. 3727–3736, 2008.

[6] S. Resmerita, A. Naderlinger, M. Huber, K. Butts, and W. Pree, “Applying real-time programming to legacy embedded control software,” in *Real-Time Distributed Computing (ISORC), 2015 IEEE 18th International Symposium on*. IEEE, 2015, pp. 1–8.

[7] D. Ziegenbein and A. Hamann, “Timing-aware control software design for automotive systems,” in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 56.

[8] C. Aussaguès, “Deterministic and dependable (also known as predictable and robust) embedded real-time systems… …with the OASIS and PharOS technology,” 2012, invited Talk at the 17th IEEE International Conference on Engineering of Complex Computer Systems.

[9] T. C. Hu, “Parallel sequencing and assembly line problems,” *Operations research*, vol. 9, no. 6, pp. 841–848, 1961.

[10] J. Templ, “Timing Definition Language (TDL) 1.5 specification,” University of Salzburg, Tech. Rep. T024, July 2009, http://www.softwareresearch.net.

Towards Correct Transformation: From High-Level Models to Time-Triggered Implementations

Hela Guesmi*, Belgacem Ben Hedia*, Simon Bliudze†, Mathieu Jan* and Saddek Bensalem‡

*CEA, LIST, PC 172, 91191 Gif-sur-Yvette, France. Email: firstname.lastname@cea.fr
†EPFL IC IIF RiSD, Station 14, CH-1015 Lausanne, Switzerland. Email: simon.bliudze@epfl.ch
‡Verimag, 38610 Gieres, France. Email: Saddek.Bensalem@imag.fr

Abstract—**In embedded systems, high-level component-based design approaches have been proposed in order to allow specification and design of complex real-time systems. However, their final implementations mostly rely on the generation of code for generic execution platforms. On the other hand, a variety of Real-Time Operating System (RTOS), in particular when based on the Time-Triggered (TT) paradigm, guarantee the temporal and behavioural determinism of the executed software. However, these TT-based RTOS do not provide high-level design frameworks enabling the scalable design of complex safety-critical real-time systems. The goal of our work is to couple a high-level component-based design approach based on the RT-BIP (Real-Time Behaviour-Interaction-Priority) framework with a safety-oriented real-time execution platform, implementing the TT approach. Thus, we combine their complementary advantages, by deriving correct-by-construction TT implementations from high-level componentised models. To this end, we propose an automatic transformation process from RT-BIP models into applications for the target platform based on the TT execution model. This transformation is already partially implemented.**

I. Introduction

The Time-Triggered (TT) paradigm for the design of real-time systems was introduced by Kopetz [11]. TT systems are based on a periodic clock synchronization in order to enable a TT communication and computation. Each subsystem of a TT architecture is isolated by a so-called *temporal firewall.* It consists of a shared memory element for unidirectional exchange of information between sender and receiver task components. It is the responsibility of the *TT communication system* to transport, by relying on the common global time, the information from the sender firewall to the receiver firewall. The strong isolation provided by the temporal firewall is key to ensuring the determinism of task execution and, thereby, allowing the implementation of efficient scheduling policies.

Developing embedded real-time systems based on the TT paradigm is a challenging task due to the increasing complexity of such systems and the necessity to manage, already in the programming model, the fine-grained temporal constraints and the low-level communication primitives imposed by the temporal firewall abstraction. Several Real-Time Operating Systems (RTOS) implement the TT execution model, such as for instance [3], [10]. However, they do not provide high-level programming models that would allow the developers to think on a higher level of abstraction and to tackle the complexity of large safety-critical real-time systems. Model-based design frameworks, such as [1], [7], allow the specification, the design and the simulation of real-time systems. In particular, the framework of [1] is a component-based framework for the design of real-time systems. It allows verification of behavioural properties, such as deadlock-freedom, and lends itself well to model transformations.

To the best of our knowledge, few connections however exist between high-level component-based design frameworksand TT execution platforms. A model transformation for generating distributed implementations from (non-real-time) BIP models is presented in [5]. A method for generating a mixed hardware/software system model for many-core platforms from a high-level non-real-time application model and a mapping between software and hardware components is presented in [8]. Nevertheless, these two approaches do not target the platforms based on TT execution model, thereby falling short of exploiting the strong temporal determinism guaranteed by the latter. A design framework based on UML diagrams and targeting the TT architecture is presented in [13]. [4] presents a transformation from SCADE [7] to PharOS [3]. The former does not target generic TT implementations since it assumes the underlying TT protocol to be the FlexRay standard, while the latter is limited to the relatively simple temporal behaviours. [6] presents a method to reduce the gap between models used for timing analysis and for TT code generation. Nevertheless, these approaches do not rely on a single semantic framework.

In this work, we establish a link between the model-based design framework RT-BIP [1] and a RTOS based on TT approach. Generating TT implementations from high-level RT-BIP models is achieved by a two-step transformation. The first step [9] transforms a generic RT-BIP model into a restricted one, which lends itself well to an implementation based on TT communication primitives. The second step, which is the subject of this paper, transforms the resulting model into the TT implementation provided by the PharOS RTOS. We identify the key difficulties in defining this transformation, propose solutions to address these difficulties and study how this transformation can be proven to be semantics-preserving.

This paper is structured as follows. In Section II, we provide the necessary background on RT-BIP and PharOS. The transformation is presented in Section III, while the open issues that remain to be addressed are discussed in Section IV.

II. Background

A. The RT-BIP Component Framework

RT-BIP is a component framework for constructing systems by superposing three layers of modelling: Behaviour, Interaction, and Priority. The Behaviour layer consists of a set of components represented by timed automata extended by data and functions given in C. The Interaction layer describes possible interactions between atomic components. Interactions are sets of ports allowing synchronizations between components. The third layer includes priorities between interactions using mechanisms for conflict resolution. Thus, in RT-BIP, systems are built by composing *atomic components* with *interactions* (presented by *connectors*) and *priorities*.

A *component* in RT-BIP is essentially a timed automaton [2] labelled by ports that represent the component's interface for communication with other components. A transition in RT-BIP automata can be constrained by a guard , i.e., a predicate on a set of its variables. A transition can also be constrained by timing constraint tc which is a guard over a set of clocks. Timing constraint is used to specify when actions of a system are enabled regarding system clocks. If c is a clock, a timing constraint tc over c is of the form: $l_c \leq c \leq u_c$, where $l_c, u_c \in \mathbb{R}_+$. Furthermore, in RT-BIP automata, a state l can be constrained by a time progress conditions (tpc) used to specify whether time can progress at a given state of the system. Any time progress condition tpc can be written as: $tpc = c \leq u_c$, where $u_c \in \mathbb{R}_+ \cup \{+\infty\}$. In the example of Figure 1, we display two RT-BIP components $C1$ and $C2$, composed by a binary connector. Let us assume that the system reaches the state $L1$ of $C1$ with a tpc equals to $c \leq 2$ and $c \in [1, 2]$. It can then either let the time progress until $c = 2$, or execute the transition enabled for these instants. If the state $L1$ is reached when $c = 2$, the system can not let time progress. It has to execute the transition p_1.

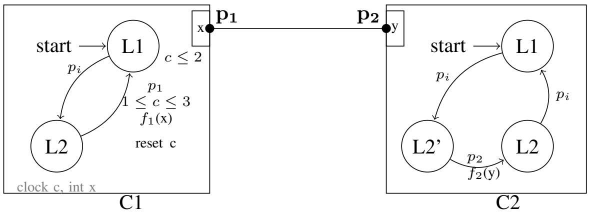


Fig. 1. Example of RT-BIP automata

B. TCA computation model and PharOS platform

Time-Constrained Automata (TCA) [12] is a formal computation model of TT tasks. The temporal behaviour of a task is specified using a directed graph, where arcs represent the successive jobs of the task to be executed (one at a time), and the nodes bear the temporal constraints of the jobs. There are four kinds of nodes:

- After node (*after(d)*): defines d as the relative release date of the following job. It is symbolized by $\triangleright_d$;
- Before node (*before(d)*): defines d as the relative deadline of the preceding job. It is symbolized by $\triangleleft_d$;
- Advance node (*advance(d)*): is a combination of $\triangleright_d$ and $\triangleleft_d$ nodes. It is symbolized by $\diamond_d$ and defines the absolute visibility date of the job data;
- No constraint node: imposes no temporal constraints on preceding and following jobs. It is symbolized by $\circ_d$.

A job can consult data whose absolute visibility dates are less or equal than the absolute release date of the job. The execution of an application can be seen as cyclically walking in the graph of each task and let the underlying scheduler choose when each encountered job is actually executed in the time interval defined by its release date and its deadline.

In the TCA example displayed in Figure 2, we have six jobs, labelled a to f, and five nodes. The release date of job d is one unit of time after the previous advance node. Two units of time after, job d should have ended. After jobs e and a are executed, communications take place since an advance node is used. The visibility date of data produced by e is three units of time later the previous after node.

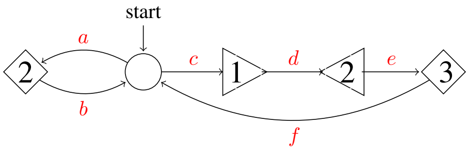


Fig. 2. Example of TCA automata

TCA computation model is implemented in PharOS [3]. PharOS is a method to design, implement and execute safety-critical multitasking applications based on the time-triggered paradigm. PharOS implements different variants of TT communication mechanisms. We are specially interested in the *temporal variables* which are real-time data flows. Values, available to all agents, are stored and updated by a single writer –the owner agent– at a predetermined temporal rhythm.

III. Work-in-progress: From RT-BIP to PharOS

In order to derive TT implementation from a high-level RT-BIP model, we follow a two-step transformation (see Figure 3).

Step1: RT-BIP model adaptation. This transformation consists in adapting the initial model, in order to comply with the TT paradigm, and especially the TT communication pattern. Each intertask interaction –initially held by connectors– is transformed in order to be handled by a dedicated component, standing for a medium between communicating tasks. The obtained model consists only of atomic RT-BIP components and connectors allowing unidirectional data transfer. Ports in TT-BIP model are send, receive or internal ports. This transformation is published in a previous work [9], and is proven to be semantics preserving.

Step 2: RT-BIP to TCA transformation. In this step, the output of the previous step is transformed into TCA automaton. In this section, we first detail challenges of the second step transformation. Then, we present the actual algorithm and how we are going to tackle the correctness proof part.

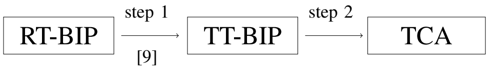


Fig. 3. From RT-BIP to the TCA TT computation model: a 2-step transformation.

A. Transformation subtleties

Transforming a component-based high-level model into a RTOS based system requires to address several subtleties.

Timing constraints mapping subtlety. The initial TT-BIP model is based on an abstract notion of time. In particular, it assumes that actions, corresponding to the computational steps of the system, are atomic and have zero execution times. Only start instant of these actions have timing constraints (tc) and timing progress conditions (tpc). However in TCA models, both release date and deadline of actions can be specified.

This issue is addressed by making use of the tpc notion in TT-BIP model, in order to extract the deadline of the following action. In fact the semantics of the tpc (used to specify whether time can progress at a given state of the system) and the *before* node in TCA are strictly similar. Both of them constraint the action preceding the node to finish before a certain date. The action succeeding the node starts right after the previous one. Timing constraint $l_c \leq c \leq u_c$ in TT-BIP, constrains only the start instant of the transition. In order to keep the same semantics in TCA, an empty action can be executed between nodes *after(l_c)* and *before(u_c)*. Right after, the actions of the initial transition can be executed.

From absolute to relative constraints subtlety. In TT-BIP, all constraints are defined using absolute labelling. However, TCA nodes bear only relatively expressed constraints, i.e., as an increment to the previous $\triangleright_d$ or $\diamond_d$ node.

In order to address this second issue, we make use of the variable d_{ref}. It is initiated to zero and updated whenever a *before* node is instantiated. It stores then the current local clock value. Relative constraint ($d_{relative}$) is computed from its corresponding absolute constraint ($d_{absolute}$) following this formula:

$$d_{relative} = d_{absolute} - d_{ref} \qquad (1)$$

Communication mapping subtlety. In the initial TT-BIP model, all tasks are related to communication components via connectors. Connectors provide not only unidirectional data transfer but also synchronization between sending and receiving actions of respectively the sender and the receiver components. In TCA, one communication model is called *temporal variable*. New values of temporal variables are made visible by their owners, i.e senders, at each of their synchronization points. Receivers of these data can consult their new values when their current time is equal or higher to the timing of these synchronization points. In our transformation two requirements need to be satisfied; (1) the receive must consult an updated temporal variable (i.e., after the sending action of the sender task) and (2) we need to respect communication semantics of the initial model.

This subtlety is addressed by generating TCA synchronization points (advance nodes) that depends on whether the TT-BIP transition is triggered by a send, receive or an internal port. After each nodes that corresponds to a communicating transition we instantiate an *advance(n)* node defined over a fine-grained clock. For example let a sender and receiver components having the same clock, and suppose they are meant to communicate in the same instant t in TT-BIP model. We can define a smaller clock, allowing to instantiate advance nodes (send and receive) at $t + \epsilon$. If we take the example of time lines displayed in Figure 4, the visibility instant of the sender data is $4*t+1$ of the clock g. The receiver will consult these data in the instant $4 * t + 2$ of the clock g.

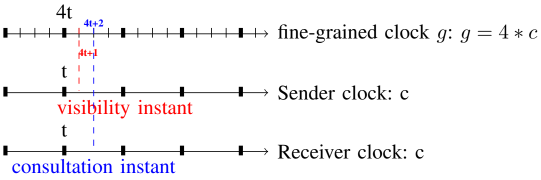


Fig. 4. Example of advance nodes defines on a fine-grained clock.

B. Transformation Algorithm

In Algorithm 1, for each transition of each automaton in TT-BIP model, we check if the transition source state has a tpc constraint. If this constraint exists, we instantiate a *before* node defined on the relative bound instant of this tpc. This before node defines the deadline of the previous action. The out transition of that node executes actions of the initial TT-BIP transition. If the transition source state tpc does not exist, and a timing constraint is defined over the transition, we instantiate two consecutive *after* and *before* nodes, defined successively over the lower and the upper bounds of the timing constraint of the initial transition. The transition relating these two states executes no actions. And then, we instantiate a transition to

Algorithm 1 Translation algorithm: Step 1

```
$d_{ref} = 0$;
for t=transition do
    if t.source.hasTpc then
        newTcaNode(before(t.source.tpc.value - $d_{ref}$));
        Synchro(t.labelPort);
        newOutTransition(t.actions);
    else if t.hasConstraint then
        newTcaNode(after(t.constraint.lowBound - $d_{ref}$));
        newOutTransition(update($d_{ref}$));
        newTcaNode(before(t.constraint.upBound - $d_{ref}$));
        Synchro(t.labelPort);
        newOutTransition(t.actions);
    else
        newTcaNode(NoConstraint());
        Synchro(t.labelPort);
        newOutTransition(t.actions);
```

execute the initial actions. If neither tpc nor timing constraint are defined in the initial automata, a node with no constraint is instantiated. Its out transition executes the initial actions.

NewTcaNode() and newOutTransition() functions and different considered cases answer to the first subtlety of the transformation. The use of the variable d_{ref} and update(d_{ref}) function goes with solving the second subtlety.

The Synchro() function in algorithm 1, answers to the third issue. It is responsible of adding synchronization points depending on if the transition is triggered by a send, receive or an internal port. Mainly it instantiates after/before variables updating, the suitable advance node bearing a well defined label at the rhythm of a fine-grained clock. The computation of the relation between the task clock and the fine-grained clock as well as the constraint supported by each instantiated advance node are subject of ongoing work.

C. Approach to prove the transformation correctness

An essential point to the transformation correctness proof approach is that the semantics of an RT-BIP component is defined as a Labelled Transition System (LTS).

In order to prove this correctness, we follow the method displayed in Figure 5. In this method we (i) express the semantics of TCA models in terms of LTS, (ii) consider the transformation between LTSs instead of the transformation between models directly and (iii) we prove that this transformation is semantics preserving using weak bisimulation technique. Then the direct transformation between models is correct by construction. This correctness proof method concerns correctness between an atomic component of TT-BIP model and its associated TCA model. However, to prove that the hole TT-BIP model is equivalent to the set of the obtained TCA models and their communication system calls, we need to prove that synchronization points preserve the same production/consumption order as in the initial model.

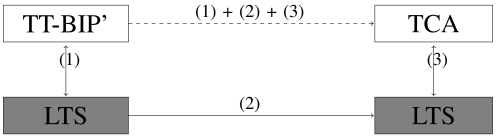


Fig. 5. Approach of the transformation correctness proof.

IV. Open challenges

We have identified several open challenges that we believe should be addressed when defining an optimized and generic transformation process.

Identification of OS service patterns potentially existing in the initial model: any OS has a number of services (communication, synchronization, etc.). We strongly think that in some initial models, and in components intended for handling communication, we can identify exactly the same behavioural pattern of one or more OS services. Transformation should take this redundancy into account, and only transform into TCA automata the part of the component which can not be mapped into an OS services. The identified pattern is thus mapped to a system call.

What about a generic transformation process? We strongly believe that the transformation process defined above can be generalized to any RTOS-based implementation approach with TT execution model. In fact, we just need to present the semantics of the computation model of the target platform as an LTS system.

V. Conclusion

In this paper we outline our approach to generate correct-by-construction TT implementation from high-level RT-BIP models. We divide this transformation into two steps; first we transform RT-BIP model in order to express intertask communication according to TT communication system, then we transform the obtained model into TCA automata (the computation model of PharOS applications). The first step is being done in previous work, we are now working on the second one. We define different subtleties induced by this transformation, and we give a short outline of the planned transformation strategy as well as the correctness proof process. We further identify open challenges related to our approach, that we plan to address with further research.

References

[1] Tesnim Abdellatif. *Rigourous Implementation of Real-Time Systems*. PhD thesis, UJF, 2012.

[2] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[3] C Aussagues, D Chabrol, V David, D Roux, N Willey, A Tournadre, and M Graniou. PharOS, a multicore OS ready for safety-related automotive systems: results and future prospects. *Proc. of The Embedded Real-Time Software and Systems (ERTS2)*, 2010.

[4] Simon Bliudze, Xavier Fornari, and Mathieu Jan. From model-based to real-time execution of safety-critical applications: Coupling SCADE with OASIS. In *Embedded Real Time Software and Systems*, ERTS2, page 10, February 2012.

[5] Borzoo Bonakdarpour, Marius Bozga, Mohamad Jaber, Jean Quilbeuf, and Joseph Sifakis. From high-level component-based models to distributed implementations. In *Proceedings of the tenth ACM international conference on Embedded software*, pages 209–218. ACM, 2010.

[6] Etienne Borde, Smail Rahmoun, Fabien Cadoret, Laurent Pautet, Frank Singhoff, and Pierre Dissaux. Architecture models refinement for fine grain timing analysis of embedded systems. In *Rapid System Prototyping (RSP), 2014 25th IEEE International Symposium on*, pages 44–50. IEEE, 2014.

[7] Jean-Louis Boulanger, François-Xavier Fornari, Jean-Louis Camus, and Bernard Dion. SCADE: Language and applications. 2015.

[8] Paraskevas Bourgos. *Rigorous Design Flow for Programming Manycore Platforms*. PhD thesis, Grenoble, 2013.

[9] Hela Guesmi, Belgacem Ben Hedia, Simon Bliudze, Saddek Bensalem, and Jacques Combaz. Towards time-triggered component-based system models. In *ICSEA15*, pages 157–169, Barcelone, Spain, November 2015. ThinkMind.

[10] Robert Kaiser and Stephan Wagner. Evolution of the pikeos microkernel. In *Proceedings of the 1st International Workshop on Microkernels for Embedded Systems*, pages 50–57, 2007.

[11] Hermann Kopetz. The time-triggered approach to real-time system design. *Predictably Dependable Computing Systems. Springer*, 1995.

[12] Matthieu Lemerre, Vincent David, Christophe Aussaguès, and Guy Vidal-Naquet. An introduction to time-constrained automata. In *Proc. of the* 3$^{\text{rd}}$ *Interaction and Concurrency Experience (ICE 2010)*, volume 38 of *EPTCS*, pages 83–98, 2010.

[13] Kathy Dang Nguyen, PS Thiagarajan, and Weng-Fai Wong. A UML-based design framework for time-triggered applications. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 39–48. IEEE, 2007.

Slot-Level Time-Triggered Scheduling on COTS Multicore Platform with Resource Contentions

Ankit Agrawal, Gerhard Fohler
Chair of Real-Time Systems
TU Kaiserslautern, Germany
{agrawal,fohler}@eit.uni-kl.de

Jan Nowotsch, Sascha Uhrig
Airbus Group Innovations
Munich, Germany
{jan.nowotsch,sascha.uhrig}@airbus.com

Michael Paulitsch
Thales Austria GmbH
Vienna, Austria
michael.paulitsch@thalesgroup.com

I. Introduction and Motivation

A number of safety-critical domains, such as avionics, use time-triggered (TT) architectures for reasons of reliability, ease of certification, reduced integration and maintenance costs, system-wide determinism, etc. [1].

The move to multicore platforms poses a number of fundamental problems for real-time scheduling in particular, even in idealized scenarios without consideration of overheads or platform characteristics. COTS multicore platforms generally share various hardware resources such as on-chip network, memory sub-system etc. amongst cores, introducing resource contentions and inter-core interferences. This results in large variability[1] in the execution time of a task depending on the latency and number of inter-core interferences from co-executing tasks on the other cores. Further, the execution of each new task, in turn, introduces additional inter-core interferences, affecting the variability in execution time of already co-executing tasks. E.g., it is shown in [2] that the single store request latency increases by 25.82 times when the number of active cores are increased from 1 to 8.

These challenges effect TT systems even more, as schedules have to be determined offline. The extension of offline scheduling to multicore platforms raises great concern in safety-critical application domains using single core platforms. E.g., in the avionics domain, in which certification and long product life are essential, only very limited steps are currently considered: The position paper from EASA and FAA proposes, as next step, at most 2 active processing cores [3]. Even with only one core active, certification is a challenge.

The problem of scheduling for TT systems on COTS multicore platform considering inter-core interferences is difficult because of three primary reasons: Firstly, we need to provide guarantees that offline computed bounds on variability in execution time of each task will hold at runtime, warranting runtime regulation of inter-core interferences during task execution. Secondly, we need to bound the variability in execution time of a task in the offline phase considering possible runtime inter-core interferences and task's deadline, as reserving hardware resources for each task considering worst-case inter-core interferences would be very pessimistic. Finally, we need to estimate at design time the maximum runtime inter-core interferences for each task in each slot, as we cannot obtain this information using traditional static WCET analysis tools due to unavailability of architecture models for the complex COTS multicore processors.

Related Work In [4], Yun et al. propose controlling memory accesses from all but one cores to limit inter-core interferences experienced by hard real-time tasks executing on just 1 core. Yun et al. extended the work in [5] allowing all cores to execute hard real-time tasks by regulating memory accesses using a memory server on each core. However, they assume that the given memory server budget reservations for each core are constant for each server period. Yao et al. [6] present a method to bound variability in execution time of each task on all cores considering round-robin arbitration between cores for memory accesses. However, the work does not consider additional arbitration and contention delay introduced by shared on-chip network in the analysis. In [2], [7], Nowotsch et al. consider a timeline divided into unequal length process frames and provide a method that bounds variability in execution time of each task in a given process frame by considering maximum inter-core interferences in the offline phase. The runtime mechanism enforces the offline computed bound in each process frame. However, they impose restrictions such as a new task is only allowed to execute on the completion of all tasks in a process frame.

A common way of operation in TT systems is to assume a minimum temporal granularity of operation, called slots [1], at runtime. Offline, a schedule table is created which assigns parts of task executions to these slots. Slots can be seen as units of resource reservation, i.e., reserving chunks of CPU time to the assigned tasks. In this paper, we propose to extend these reservations to several resources. We propose a two-part slot-level based resource-control method using a TT scheduling approach that enables the use of multicore platforms for executing hard real-time tasks in TT systems. We propose a runtime mechanism consisting of two servers running on each core - processing time server and memory access server - each having a fixed server period equal to the slot length. We guarantee the offline computed bound on variability in execution time of each task by enforcing offline computed slot-level server budget reservations on each core, thereby limiting

[1] By variability in execution time of a task, we mean the variability introduced beyond the traditional single-core WCET due to the inter-core interferences in a multicore system.

inter-core interferences from co-executing tasks, as well as, additional inter-core interferences introduced by the task under consideration. In the offline phase, we propose to generate schedule table containing mapping, scheduling and server budget reservations, that bounds the variability in execution time of each task due to inter-core interferences, such that all tasks meet their deadlines. The computation of the bound on variability in execution time of each task involves estimating maximum inter-core interferences from already scheduled co-executing tasks in each slot, as well as, limiting the slot-level inter-core interferences that may be introduced by the task at hand, at runtime.

Overall, our proposed method considers a real COTS multicore platform - Freescale P4080 - and accounts for the delay introduced by arbitration and contention in the on-chip network and the memory sub-system. Further, we did a preliminary bare-metal implementation of our proposed runtime mechanism on the P4080 platform. Moreover, our method adheres to the TT architecture model [1] preserving system-wide properties like slot-level determinism, clock synchronization, etc., enabling integration of COTS multicores in safety-critical systems using a TT approach.

II. System Model, Task Model, and Job Model

A. System Model

We consider an abstract multicore hardware architecture inspired by the readily available real COTS multicore system - Freescale QorIQ P4080 platform [8]. We focus only on the hardware resources essential for task scheduling on the P4080 platform. These are 8 e500mc cores, the memory sub-system, and the crossbar CoreNet on-chip network. Along similar lines, we assume that the abstract multicore hardware comprises only two types of hardware resources: *N* homogeneous processing cores (including private caches) from *1,..,n,...,N*, and 1 shared resource consisting of a on-chip network with a memory sub-system. As this work is a first step, we only consider 1 memory controller in the memory sub-system, even though the P4080 platform has 2 memory controllers. Further, we assume that the hardware mechanisms like pre-fetchers, cache-coherency etc., that may implicitly introduce unaccounted inter-core interferences are disabled.

The inter-core interference latency, includes the time taken by a load/store request issued from a core to access the on-chip network and the memory sub-system, considering contentions. We consider the same latencies for our abstract multicore architecture as used in [2], [7]. The measurement-based approach as described in [9], using which these latencies are obtained, tries its best to create worst-case inter-core interference scenario, but is not guaranteed to do so, as the hardware model is not provided by Freescale. However, this is not a potential limitation of our proposed method in Section III as, when available, it will also work with inter-core interference latencies obtained through static analysis.

As shown in Table I, the latency of inter-core interference varies with the number of active processing cores partly due to varying arbitration delay from shared hardware resources. Each inter-core interference latency δ_j depends on the j number of active cores. E.g., as shown in Table I, if (say) $j = 3$ cores are active from time $[t, t+1)$, we consider all inter-core interferences that occur during this time interval to have latency δ_3 (worst case) as listed in column 2 in Table I.

We consider a time-triggered (TT) scheduling approach and assume the timeline is divided into fixed equal length slices called slots [1]. A slot $S_{t,n}$ represents a time interval $[t, t+1)$, (where *t* is an integer multiple of slot length $|S|$) on core *n*. We also assume the system is preemptive at each slot boundary.

TABLE I: Inter-core interference latency and corresponding memory access server budget reservation for different number of active cores

No. of active cores (j)	Inter-core interference latency (δ_j in clock cycles)	Memory access server budget reservation in 1ms (Acc_j)
1	41	29385
2	164	7346
3	245	4917
4	463	2602
5	517	2330
6	737	1634
7	784	1536
8	1007	1196

B. Task Model and Job Model

The set Γ represents *V* hard real-time periodic tasks with arbitrary deadlines. Each task τ_i is characterized by the tuple $\langle C_i^s, MA_i, T_i, D_i \rangle$, where, C_i^s is the single core WCET excluding the time taken by memory accesses, MA_i is the maximum number of memory accesses to the shared resource, T_i is the period, and D_i is the relative deadline. C_i^s and MA_i are obtainable using a combination of static timing analysis tool like aiT and measurements [7]. Tasks may have precedence and communication constraints specified in graph $\mathcal{G}$.

In our proposed method (Section III), as we allow each instance of a task to have a different bound on variability in execution time, we convert the given task set to jobs, where each instance of a task is a job. The set $\mathcal{J}$ represents all jobs *W* of all tasks in a task set Γ in time $[0, H)$, where H is the hyperperiod of the task set Γ. Each job $\tau_{i,k}$ is characterized by the tuple $\langle C_{i,k}^s, MA_{i,k}, C_{i,k}^m, r_{i,k}, d_{i,k} \rangle$. $C_{i,k}^s$ and $MA_{i,k}$ are same across all jobs of task τ_i. $C_{i,k}^m$ is the multicore execution time i.e. the bound on variability in execution time of job $\tau_{i,k}$, computed offline, considering possible runtime inter-core interferences. $C_{i,k}^m$ may differ between different jobs of the same task τ_i. $r_{i,k}$ is the absolute release time and $d_{i,k}$ is the absolute deadline of job $J_{i,k}$, which are computed based on the related parameters of the corresponding task.

III. Proposed Slot-Level Based Method

In this section, we present our slot-level based method using a TT scheduling approach. The runtime mechanism is described in Section III-A and the offline phase in Section III-B

A. Runtime mechanism

We propose two server types - processing time server and memory access server, implemented using built-in hardware monitors. Each server type runs on each core and controls only one type of resource.

1) Processing time server: On each core n, a processing time server τ_{sp_n} regulates the execution time in each server period based on the slot-level server budget reservations computed in the offline phase. During runtime, an executing job at time t consumes the server budget reservation $Q_{sp_n,t}$ for the computation time on core n and stall time due to cache misses and/or memory accesses, resulting in a corresponding decrease of the server budget.

2) Memory access server: The memory access server τ_{sm_n} regulates the total number of memory accesses allowed from each processing core n in each slot S_t based on slot-level offline computed server budget reservations $Q_{sm_n,t}$, thereby controlling the inter-core interferences. At runtime, an executing job uses the server budget reservation only for memory accesses, resulting in a decrease of server budget by 1 for each memory access issued.

3) Runtime behaviour: During runtime, each core-level scheduler, at the start of each slot, assigns a job to the respective core and sets the corresponding server budget reservations for each server based on the schedule table obtained in the offline phase. On each core, if the budget of any server reaches 0, the corresponding core-level scheduler suspends the executing job, irrespective of the remaining budget of the other server. Jointly, the two servers on each core guarantee that the server budget reservations provided for each slot in the offline schedule table hold at runtime, thereby enabling bounding of variability in execution time for each job in the offline phase considering possible runtime inter-core interferences.

4) Inter-relationship amongst two servers, slots, and inter-core interference latencies: We consider the server period of each server is equal to the slot length $|S|$. For each processing time server instance, we allow only two mutually exclusive server budget reservation values: Either the budget reservation equals to zero which means an idle slot (no task is allowed to execute), or it equals to some fixed positive value X chosen by the system designer, such that $X \leq |S|$. For each memory access server instance, we allow $N + 1$ mutually exclusive server budget reservation values. The N different budget reservation values directly associate with the different number of active cores. An additional budget reservation value of 0 relates to the idle slot, resulting in a total of $N + 1$ possible budget reservation values. Based on the description of each server, the relationship between the server budget reservations of the two servers and the inter-core interference latency δ_j, for each active core n, is given by the formula $Q_{sm_n,t} = \left\lfloor \frac{Q_{sp_n,t}}{\delta_j(t)} \right\rfloor, \forall t$, where j represents the number of active cores at time t. E.g., we consider a slot length $|S|$ of 1ms and processing time server budget of 1ms. Table I then, shows the memory access server budget reservation values Acc_j (column 3) for j active processing cores (column 1).

5) Preliminary implementation: We did a preliminary bare-metal implementation of our proposed runtime mechanism running on all cores of the P4080 COTS multicore platform. We implemented the processing time server using the multicore programmable interrupt controller (MPIC) timer that enables slot-level synchronization amongst all cores. The MPIC timer also allows to set multiple processing cores as interrupt recipients, and provides each recipient core a unique interrupt copy [8]. We implemented the memory access server using a core-level hardware performance monitor that counts requests to the on-chip network [10]. We implemented our proposed suspension rules in interrupt service routines of the MPIC timer and hardware performance monitor on each core.

Though the idea to regulate memory accesses from each core using memory access server may seem similar to MemGuard [5], there are three key differences. Firstly, MemGuard considers minimum guaranteed memory bandwidth as constant, whereas our proposed method considers it as variable depending on the number of active cores (see Table I). Secondly, MemGuard assumes the memory server budget reservations for each core as given and constant across all server periods, whereas we do not make such an assumption. Thirdly, MemGuard does not consider if the given server budgets meet task deadlines, whereas our proposed method (introduced later in the Section III-B) gives offline guarantees.

B. Offline phase: Bounding variability in execution time

In the offline phase, we bound the variability in execution time of each job by computing server budget reservations for each slot, such that all tasks meet their deadlines. This limits, at runtime, the maximum inter-core interferences from co-executing jobs as well as the additional inter-core interferences introduced by the job under consideration.

In the offline phase, let's consider at slot S_t on core n, the offline scheduler tries to schedule job $\tau_{i,k}$, such that the job $\tau_{i,k}$ meets its deadline without affecting the already scheduled jobs. In order to compute the multicore execution time $C^m_{i,k}$, the offline scheduler first tries the simple case, where it considers a job $\tau_{i,k}$ executes in each slot on some core n with constant memory access server budget reservation Acc_j. In this simple case, we compute the multicore execution time of job $\tau_{i,k}$ using the formula $C^m_{i,k} = \left\lceil C^s_{i,k} + \frac{MA_{i,k}}{Acc_j} \right\rceil$ based on some memory access server budget reservation Acc_j chosen by the offline scheduler (slot length S of 1ms).

However, it is possible that the offline scheduler is unable to find and reserve enough slots for the job $\tau_{i,k}$ that fulfill chosen memory access server budget reservations Acc_j on core n until its deadline. In that case, we propose a different way to compute the multicore execution time.

Let's consider on core n, in the time $[t, t + z + 1)$, due to already scheduled jobs on remaining cores, the offline scheduler only finds slots with different memory access server budget reservations in time $[t, t+z+1)$. Then (say) from slot S_{t+z+1}, the scheduler finds slots with constant memory access server budget reservations $Acc_{j'}$. In this case, we propose to first determine the minimum progress the job $\tau_{i,k}$ can make in time $[t, t + z + 1)$ (both computation time and memory accesses). Then, we subtract the same while computing the remaining multicore execution time $c^m_{i,k}(t + z + 1)$ based on new budget reservation $Acc_{j'}$ from time $t+z+1$ onwards. The offline scheduler then reserves the slots in time $[t, t + z + 1)$

with the available server budget reservations, and from time $t+z+1$ to $t+z+c^{m}_{i,k}(t+z+1)$ with $Acc_{j'}$. If the job $\tau_{i,k}$ still cannot meet its deadline, the offline scheduler will try to reschedule some already scheduled jobs, e.g., by backtracking.

IV. Example

Figure 1 shows an example of our proposed method considering only 2 cores due to the space constraints. Each core n has two servers: processing time server τ_{sp_n} and memory access server τ_{sm_n}, with server period equal to the slot length $|S|$ of 1ms. For each server, the dotted horizontal lines depict the possible server budget reservation values. During runtime, at the start of each slot, each core-level scheduler assigns a job to the respective core and sets the corresponding server budgets for each server, based on the offline schedule table.

At time $t=0$, as both the cores are active, each core-level scheduler sets the corresponding processing time server budget reservation to 1ms and memory access server budget reservation to $Acc_2 = 7346$ accesses (based on Table I). At time $t=1$ms, only job $\tau_{0,0}$ is active resulting in memory access server budget $Q_{sm_1,1} = Acc_1 = 29385$ accesses (based on Table I) and processing time server budget of 1ms. In the time interval $[1, 1.33)$ms, the job $\tau_{0,0}$ issues memory access as shown by corresponding decrease in memory access server budget. Then in the time interval $[1.33, 2)$ms, it does not perform any memory accesses as shown by memory access server budget being constant. Later, it again briefly issues memory accesses for the next 100μs. In the time interval $[1.6, 2)$ms, since, only the processing time server budget decreases and not the memory access server budget, the job $\tau_{0,0}$ only performs computations. At time $t=3$ms, the job $\tau_{1,0}$ completes execution and the scheduler of core 2 discards the unused memory access server budget from the previous server instance $\tau_{sm_2,2}$. Further, at time $t=3$ms, as only job $\tau_{0,0}$ is active, the memory access server budget $Q_{sm_1,3}$ equals Acc_1. The job $\tau_{0,0}$ issues memory accesses in the first half of the slot as shown by decrease in memory access server budget. Then, for the next 200μs, it does not issue any memory accesses as the memory access server budget does not decrease and at time $t=3.7$ms completes execution, resulting in discarding of unused server budgets by the core-level scheduler.

V. Conclusion and Future Work

In this work, we presented an initial step towards enabling time-triggered (TT) scheduling on a real COTS multicore platform P4080. It takes into account inter-core interferences in the on-chip network and the memory sub-system. Our proposed method comprises a runtime mechanism and an offline phase. For the runtime mechanism, we proposed a processing time server and a memory access server for each core. Jointly, the two servers on each core, enforce slot-level offline computed server budget reservations, thereby limiting the maximum inter-core interferences introduced and experienced by each task considering different inter-core interference latencies. In the offline phase, we proposed a procedure for the offline scheduler to compute the bound on variability in execution time of each task while allowing different slot-level memory access server budget reservations. Overall, our proposed method facilitates integration of COTS multicore platforms in TT systems, while maintaining features of TT architecture like slot-level determinism, clock synchronization, etc.

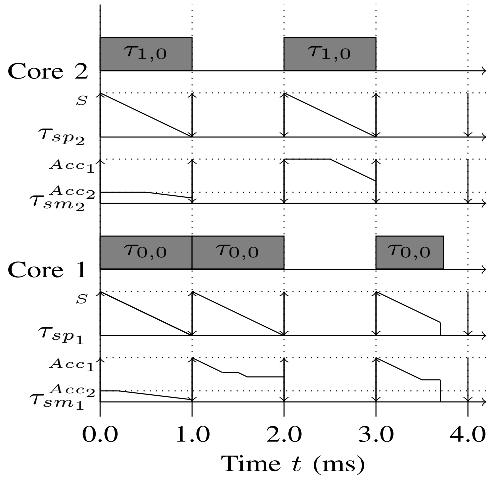


Fig. 1: Example of our proposed slot-level based method

We did a preliminary bare-metal implementation of our proposed runtime mechanism on a real COTS multicore platform P4080. In future work, we aim to provide safe bounds for the variability in execution time and will integrate the procedure in our existing offline scheduler to generate schedule tables containing mapping, schedule and server budget reservations.

Acknowledgment

The work was supported by ARTEMIS project 621429 EMC2. We thank the referees for several useful comments.

References

[1] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer-Verlag, 2011.

[2] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, July 2014, pp. 109–118.

[3] *CAST-32 Multi-core Processors*. Certification Authorities Software Team, May 2014.

[4] 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*, July 2012, pp. 299–308.

[5] ——, "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*, April 2013, pp. 55–64.

[6] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "Schedulability analysis for memory bandwidth regulated multicore real-time systems," *Computers, IEEE Transactions on*, vol. 65, no. 2, pp. 601–614, Feb 2016.

[7] J. Nowotsch and M. Paulitsch, "Quality of service capabilities for hard real-time applications on multi-core processors," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, ser. RTNS '13. New York, NY, USA: ACM, 2013, pp. 151–160.

[8] *P4080 QorIQ Integrated Multicore Communication Processor Family Reference Manual Rev. 2*, Freescale Semiconductor, 2014.

[9] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *Dependable Computing Conference (EDCC), 2012 Ninth European*, May 2012, pp. 132–143.

[10] *e500mc Core Reference Manual Rev. 3*, Freescale Semiconductor, 2013.

Scheduling Multi-Threaded Tasks to Reduce Intra-Task Cache Contention

Corey Tessler
Wayne State University
corey.tessler@wayne.edu

Nathan Fisher
Wayne State University
fishern@wayne.edu

Abstract—**Research on hard real-time systems and their models has predominately focused upon single-threaded tasks. When multi-threaded tasks are introduced to the classical real-time model the individual threads are treated as distinct tasks, one for each thread. These artificial tasks share the deadline, period, and worst case execution time of their parent task. In the presence of instruction and data caches this model is overly pessimistic, failing to account for the execution time benefit of cache hits when multiple threads of execution share a memory address space.**

This work takes a new perspective on instruction caches. Treating the cache as a benefit to schedulability for a single task with m threads. To realize the "inter-thread cache benefit" a new scheduling algorithm and accompanying worst-case execution time (WCET) calculation method are proposed. The scheduling algorithm permits threads to execute across conflict free regions, and blocks those threads that would create an unnecessary cache conflict. The WCET bound is determined for the entire set of m threads, rather than treating each thread as a distinct task. Both the scheduler and WCET method rely on the calculation of conflict free regions which are found by a static analysis method that relies on no external information from the system designer.

By virtue of this perspective the system's total execution execution time is reduced and is reflected in a tighter WCET bound compared to the techniques applied to the classical model. Obtaining this tighter bound requires the integration of three typically independent areas: WCET, schedulability, and cache-related preemption delay analysis.

I. Introduction

In the classical model of real-time systems, shared resources are often considered detractors to schedulability analysis and exclusively increase worst-case execution times (WCETs). Cache memory is one such shared resource viewed from this exclusively negative perspective. It is a natural perspective, derived from a preempting task invalidating cache lines, thus extending a preempted task's execution time.

For example in the classical periodic task model [1] it is implied that a task is a single thread of execution. These models lack a representation for tasks with multiple threads. To apply WCET and schedulablility techniques developed for the classical models, a task that executes multiple threads is treated as several duplicate tasks with a single thread of execution. Any task that releases a job with m threads will be converted to m tasks each releasing one job.

Under such models, tasks are assumed to be in competition for cache space. The inclusion of threads, which are converted to tasks, only amplifies the negative affect. However, threads are not always in competition with other threads, but in fact can mutually benefit from reusing the same resources. Threads of the same task share a memory space (also referred to as an address space). A cache miss during the execution of one thread can place values into the cache that produce a cache hit for a second thread. These unexpected cache hits reduce the execution time of the second thread and the system overall. This speed up is called the **inter-thread cache benefit**. While other researchers have developed techniques for limiting the impact of caches [2] [3] [4], we are unaware of any existing analysis technique that accounts for the benefit of caches between threads or tasks.

The purpose of this work is to illustrate the potential inter-thread cache benefit for instruction caches, and to argue for a new task model and schedulability analysis technique. Current approaches to Worst Case Execution Time (WCET), Cache-Related Preemption Delay (CRPD), and schedulability analyis typically operate independently. Accounting for the inter-thread cache benefit requires a unified approach, integrating all of these disciplines.

As a first step towards a complete approach this work is limited to a single task executing multiple identical threads. Preemptions between threads incur no time penalty. The main work-in-progress efforts described herein are the development of a new thread based scheduling algorithm called `BUNDLE`, and a method for calculating the WCET bound of m threads scheduled by `BUNDLE`. By the nature of the WCET calculation, the bound permits any number of preemptions between threads. When m is greater than one, the bound is guaranteed to be less than any WCET method using the competitive perspective from the classical model.

II. Motivating Example

An example in two parts provides the motivation for this work. This example illustrates the inter-thread cache benefit and highlights the pessimism in existing WCET and CRPD approaches. As noted, the classical model provides no representation of a threads distinct from a job. To aid the example the classical model is augmented with two new concepts: threads and ribbons.

Customarily, the term "thread" may refer to the execution of a sub-job, or it may refer to the subset of instructions reachable from an entry point. To clarify the distinction the

This research has been supported in part by an NSF CAREER Grant (No. CNS-0953585), an NSF CRI grant (No. CNS-1205338), and a grant from Wayne State University's Office of Vice President of Research.

term *ribbon* is introduced and refers to the subset of instructions reachable from a single entry point. A *thread* will refer exclusively to one instance of execution i.e., an instantiation of a ribbon.

The computational setting is a single processor with an instruction cache of l lines. The cache is write-through and direct-mapped, assigning exactly one cache line to a memory address. To simplify the presentation, every instruction completes in $\mathbb{I}$ time units. If an instruction is absent from the cache before execution, its completion will be delayed by the Block Reload Time (BRT): $\mathbb{B}$. Executing an instruction out of the cache (i.e. a hit) takes $\mathbb{I}$ time, while caching and executing a miss takes $(\mathbb{I} + \mathbb{B})$.

For periodic and sporatic tasks, the classical model accumulates the n tasks in the set $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$. Each task is characterized by a tuple of minimum inter-arrival time, relative deadline, and worst case execution time: $\tau_i = (p_i, d_i, c_i)$. Each c_i value is an upper bound on the amount of time one job of τ_i will take to complete if the job executes without preemption.

To more closely represent the execution of threads within jobs the model must be modified. With that purpose, each task is represented by a tuple of minimum inter-arrival time, relative deadline, and initial ribbon: $\tau_i = (p_i, d_i, r_i)$. A ribbon r_i is identified by a starting instruction within the object of a task, it includes all reachable instructions until an exit point. The set of m ribbons from all tasks is named $R = \{r_1, r_2, ..., r_m\}$, where $|R| \geq |\tau|$. Every ribbon has an associated worst-case execution time: $r_j = c(r_j)$.

TABLE I. SUMMARY OF MODEL PARAMETERS

Tasks	Task	Ribbons	Ribbon
$\tau_i \in \tau$	$\tau_i = (p_i, d_i, r_i)$	$r_j \in R$	$r_j = c(r_j)$

Cache Lines	Instruction Time	BRT
l	$\mathbb{I}$	$\mathbb{B}$

A. Example Part I: 1 Linear Ribbon, 2 Threads

To demonstrate the inter-thread cache benefit, consider the following task set of a single task $\tau = \{\tau_1\}$. The task has only one ribbon, the initial ribbon r_1 which readies two threads, r_1^1 and r_1^2 for every job release. For simplicity, r_1 contains no loops or branches.

TABLE II. EXAMPLE MODEL PARAMETERS

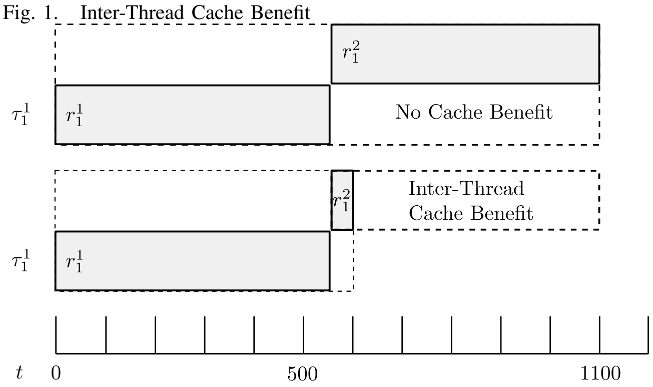
Tasks	Task	Ribbons	Ribbon Length
$\tau = \{\tau_1\}$	$\tau_1 = (p_1, d_1, r_1)$	$R = \{r_1\}$	$\|r_1\| = 50$

Cache Lines	Instruction Time	BRT
$l = 200$	$\mathbb{I} = 1$	$\mathbb{B} = 10$

For every job release, consider a scheduling algorithm that runs one thread of r_1 to completion before permitting the second thread to begin execution. The ribbon r_1 is 50 instructions long, without loops or branches all 50 instructions miss the cache and each take $(\mathbb{I} + \mathbb{B})$ = 11 cycles to complete for a total of 550 cycles per thread. An analysis that does not consider the benefit of the cache between threads of r_1 will reserve 1100 cycles for the combined execution of r_1^1 and r_1^2.

Since τ_1 (and r_1) contain less than l uniquely addressed instructions, every instruction of τ_1 maps to a distinct address in the cache. Assuming no cache flushes are permitted during the execution of r_1^1, the execution time of 1100 cycles is reduced to 600 cycles by the inter-thread cache benefit. Figure 1 illustrates the reduction.

Fig. 1. Inter-Thread Cache Benefit



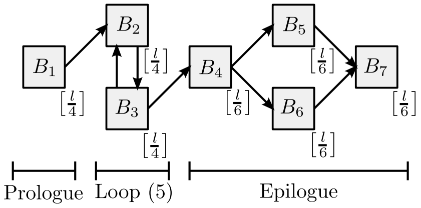
During the execution of r_1^1 every instruction is placed in the cache consuming 550 cycles. Since r_1^1 and r_1^2 share the same address space, all instructions are available in the cache during r_1^2's execution; taking 50 cycles to complete. Also note, in this task system 600 cycles is the worst case execution time for τ_1. Regardless of the thread execution order, one thread will execute an instruction before the other thread caching the instruction value. When the later thread executes an instruction, it will find the value present in the cache.

B. Example Part II: Existing WCET & CRPD Approaches

To illustrate the benefit of a new approach, the existing approaches to WCET and CRPD analysis are applied to a slightly more complicated ribbon. The advantage will be shown by applying the cache aware methods of Arnold [5] and Mueller's [6] for WCET calculation, and Lee et al.'s [7] CRPD determination. These methods were chosen for illustrative purposes and for their continued use in subsequent works.

The structure of r_1 from the previous example is ill suited for meaningful WCET or CRPD analysis. To continue, r_1 is modified to include a prologue, loop, and epilogue for a total instruction count of $\frac{5}{4} \cdot l$. Figure 2 gives the control flow graph [8] (CFG) of r_1 which connects serial sets of instructions, called basic blocks, by their logical control flow through the ribbon. Below each basic block is a counting term in square brackets listing the number of instructions in the block. The parenthesized value at the bottom of the figure indicates the number iterations the loop will execute.

Fig. 2. r_1 CFG



1) WCET: Predominantly, WCET analysis that includes cache behavior is limited to a single task, specifically between preemption points [5]. Arnold [5] and Mueller's [6] approach iterates over the control flow graph categorizing instructions as cache must-miss, first-miss, first-hit, and must-hit. Table III lists the result of categorization for each basic block using Arnold's approach.

TABLE III. Basic Block Categorization

B_1	B_2	B_3	B_4	B_5	B_6	B_7
must-miss	first-miss		must-miss			

Using these categorizations and the loop bound, the worst case execution time of r_1 is the sum of the excution times of the prologue, the entry executions of B_2 and B_3, the repititions of B_2 and B_3, and the epilogue. Table IV gives the intermediate values, using the model parameters of $\mathbb{B} = 10$, $l = 200$, and $\mathbb{I} = 1$ the total execution time taking into consideration reloads is: $\frac{l(\mathbb{B}+\mathbb{I})}{4} + \frac{2l(\mathbb{B}+\mathbb{I})}{4} + \frac{8l(\mathbb{I})}{4} + \frac{3l(\mathbb{B}+\mathbb{I})}{6} = \frac{l(5\mathbb{B}+13\mathbb{I})}{4} = 3150$

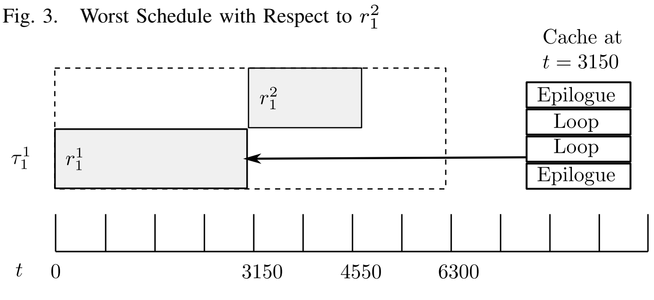
TABLE IV. Segment WCET

Section	Basic Blocks	WCET
Prologue	B_1	$(\frac{l}{4} \cdot (\mathbb{B}+\mathbb{I}))$
Loop Entry	$B_2 + B_3$	$(\frac{l}{4} \cdot 2 \cdot (\mathbb{B}+\mathbb{I}))$
Loop Repetition	$(B_2 + B_3) \cdot 4$ (repeats)	$(\frac{l}{4} \cdot 2 \cdot 4 \cdot (\mathbb{I}))$
Epilogue	$B_4 + (B_5$ or $B_6) + B_7$	$(\frac{l}{6} \cdot 3 \cdot (\mathbb{B}+\mathbb{I}))$

Using the WCET of 3150 for two threads of r_1, where the execution of each thread is considered a distinct task, the total demand for the one task system is 6300. However, this is overly pessimistic. The worst possible schedule for two threads of r_1 is the sequential execution of r_1^1 followed by r_1^2. It is the worst schedule because it inflicts the most cache misses during the execution of r_1^2.

The prologue consumes one quarter of the cache, the loop one half, and the epilogue another half. Meaning, after the epilogue executes all of the cache lines of the prologue have been invalidated. Scheduling r_1^2 after r_1^1 has completed requires r_1^2 to load the cache lines of the prologue, and half of the cache lines from the epilogue. With this understanding, the WCET of r_1^2 is: $\frac{l(\mathbb{B}+\mathbb{I})}{4} + \frac{5l(\mathbb{I})}{4} + \frac{l(\mathbb{B}+\mathbb{I})}{4} + \frac{l(\mathbb{I})}{4} = \frac{2l(\mathbb{B}+4\mathbb{I})}{4} = 1400$. The total demand for the task system is 4550 which is less than the 6300 calculated from the WCET analysis, demonstrating the pessimism of the Arnold and Mueller approaches. Figure 3 illustrates the worst possible schedule for r_1^2 including the cache contents at $t = 3150$, as well as the pessimistic estimate for τ_1's execution time.

Fig. 3. Worst Schedule with Respect to r_1^2



2) CRPD: Cache related preemption delay accounts for the execution time extension of one task due to the cache interference of another. A task executing in isolation may store and reuse values from the cache. When preempted, those stored cache values may be invalidated before they are reused. Upon resuming the preempted task must pay the BRT for each invalidated cache block, extending the execution time of the preempted task. This delay is called the Cache Related Preemption Delay (CRPD), and one method for calculating it is the Useful Cache Block (UCB) approach developed by Lee et al. [7].

The UCB approach borrows from the Arnold and Mueller approaches, iterating over the control flow graph to determine if cache blocks are useful or not. A useful cache block is "a cache block that contains a memory block that may be referenced before being replaced by another memory block." – within the same task.

From Figure 2 there are two basic blocks that contribute UCBs to the thread r_1: B_2 and B_3. Applying Lee's method, CRPD of a preemption of r_1 is $\frac{2l(\mathbb{B}+\mathbb{I})}{4} = 1100$ the sum of cache lines from B_2 and B_3. However, this bound is overly pessimistic.

Given the schedule in Figure 3 once the "Loop" instructions are cached they cannot be invalidated. If r_1 were to be preempted after the first iteration of the loop the cache lines mapping to the instructions of B_2 and B_3 would be populated. No other instructions of r_2 map to those cache lines, and cannot invalid them. Furthermore, there is no schedule of r_1^1 and r_1^2 which incurs any CRPD.

Lee's approach to CRPD calculation is known to be an overestimate, there are refinements such as the UCB-ECB [9], UCB-Union, and UCB-Union Multiset [10] approaches. However, the UCB calculation is a component of each of them and the advanced techniques suffer from the same inability to address cache memory as a benefit rather than a detriment. Similarly, the Arnold and Mueller approaches play a role in subsequent WCET methods. For this reason, the fundamental approaches of Lee, Arnold and Mueller were selected for our evaluation of the potential inter-thread cache benefit.

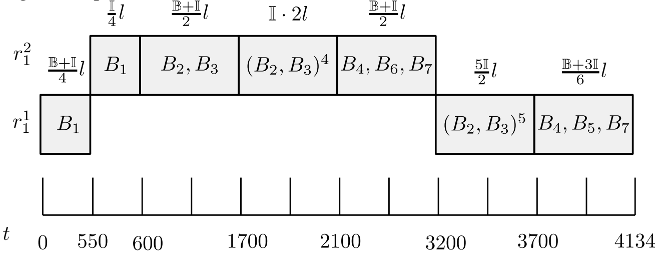
III. An Optimal Schedule

The maximum WCET and CRPD bound determined for r_1 in the previous section relies on knowledge of the scheduler. A further reduction in WCET and CRPD values is possible by crafting a schedule that considers the cache. Figure 4 is an optimal schedule created by considering the cache contents and thread flow. Each thread will take a different path through the epilogue, r_1^1 will take the "high" road, executing the basic blocks in the following order: $\langle B_1, (B_2, B_3)^5, B_4, B_5, B_7 \rangle$. The thread r_1^2 will take the "low" road, executing: $\langle B_1, (B_2, B_3)^5, B_4, B_6, B_7 \rangle$.

Examining the structure and cache usage of r_1, the prologue is the problematic section. Upon completion, r_1 will remove the cache contents of the prologue. To benefit from the cached values r_1^2 must be scheduled to run before the prologue has been invalidated.

The schedule is presented as the series of block executions, the distance between marks on the time axis are not to scale. Above each series of basic blocks is the execution time

Fig. 4. Optimal Schedule



required to complete the section. By preempting r_1^1 after the completion of B_1 the total execution time is reduced to 4134.

IV. Progress and Remaining Effort

Treating caches solely as detractors in schedulability analysis leads to overly pessimistic WCET bounds for multi-threaded tasks and task systems. This pessimism is further increased by the independent treatment of threads in CRPD analysis. By considering the three disciplines of schedulability, WCET, and CRPD analysis in concert a tighter bound on execution of tasks and task systems can be achieved.

A complete solution that accounts for multiple tasks each with an arbitrary number of ribbons is too ambitious for a first work. By focusing on a single task with one ribbon and m threads, a first useful solution may provide insight for later efforts. Our current research aims include developing a scheduling algorithm (BUNDLE), static analysis, and WCET bound for m threads of a single ribbon.

Static analysis provides *conflict free* regions that are used by the scheduling algorithm at run time and off-line timing analysis. A conflict free region is a sub-graph of a ribbon's CFG. It includes a starting instruction and all reachable instructions that cannot create a cache conflict. Conflicts that arise during the execution of a single thread are classified as intra-thread, and those incurred by preemption inter-thread.

At run time, BUNDLE uses conflict free regions to group and schedule threads. Intuitively, all threads of the same region are allowed to run in any order and preempt each other arbitrarily. However, when any thread would execute an instruction outside of the region it is blocked until all other threads reach a boundary. After all threads in a region have been blocked, another region is selected for execution.

Determining the WCET of a set of m threads requires the conflict free region boundaries and knowledge of the schedule produced by BUNDLE. It also relies upon the static analysis of structures within CFG to establish bounds on the regions. For each conflict free region serial, looping, and branching sub-structures are extracted and help characterize the region with an execution time bound.

Figure 5 illustrates the result of timing characterization for conflict free regions of a ribbon r with CFG G. This information, along with the number of threads is passed to the final timing analysis. After finding the longest path through the ribbon, incorporating the behavior of BUNDLE over the m threads produces an execution bound for the set of threads. Figure 6 outlines the general approach to timing analysis.

Fig. 5. Sub-Graphs and Bounds for G of r

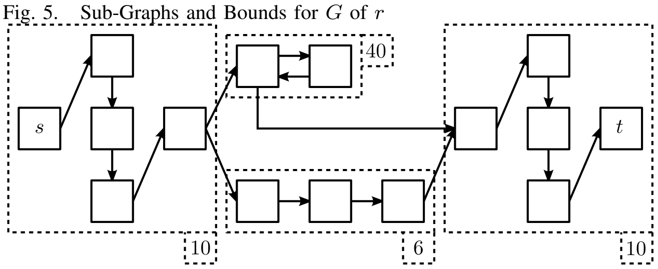
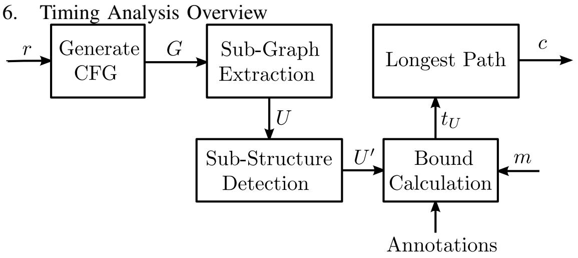


Fig. 6. Timing Analysis Overview



Two significant tasks remain to complete this work: to evaluate and compare. The evaluation will entail more effort due to the lack of tools. Performing an evaluation requires a new memory restricted threading library and CPU simulator with extensible cache configuration. Comparing with related works such as PREM [2] and MultiPREM [3] will emphasize the benefit of combining the three aforementioned disciplines.

References

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

[2] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for cots-based embedded systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, April 2011, pp. 269–279.

[3] A. Alhammad and R. Pellizzoni, "Time-predictable execution of multithreaded applications on multicore systems," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, March 2014, pp. 1–6.

[4] M. Schoeberl, W. Puffitsch, and B. Huber, "Towards time-predictable data caches for chip-multiprocessors," in *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, ser. SEUS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 180–191.

[5] R. Arnold, F. Mueller, D. Whalley, and M. Harmon, "Bounding worst-case instruction cache performance," *Real-Time Systems Symposium, 1994., Proceedings.*, pp. 172–181, Dec 1994.

[6] F. Mueller, "Static cache simulation and its applications," Ph.D. dissertation, Florida State University, 1995.

[7] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *IEEE Transactions on Computers*, vol. 47, no. 6, pp. 700–713, Jun. 1998.

[8] F. E. Allen, "Control flow analysis," *SIGPLAN Not.*, vol. 5, no. 7, pp. 1–19, Jul. 1970.

[9] H. S. Negi, T. Mitra, and A. Roychoudhury, "Accurate estimation of cache-related preemption delay," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '03. New York, NY, USA: ACM, 2003, pp. 201–206.

[10] S. Altmeyer and C. Maiza Burguière, "Cache-related preemption delay via useful cache blocks: Survey and redefinition," *Journal of Systems Architecture*, vol. 57, no. 7, pp. 707–719, Aug. 2011.

I/O contention aware mapping of multi-criticalities real-time applications over many-core architectures

Laure Abdallah and Mathieu Jan
CEA, LIST, Embedded Real Time Systems Laboratory
F-91191 Gif-sur-Yvette, France
Email: Firstname.Lastname@cea.fr

Jérôme Ermont and Christian Fraboul
IRIT INP-ENSEEIHT, Université de Toulouse
F-31000 Toulouse, France
Email: Firstname.Lastname@enseeiht.fr

***Abstract*—Many-core architectures are more promising hardware to design real-time systems than multi-core systems as they should enable an easier mastered integration of a higher number of applications, potentially of different level of criticalities. However, the worst-case behavior of the Network-on-Chip (NoC) for both inter-core and core-to-Input/Output[1] (I/O) communications of critical applications must be established. The mapping over the NoC of both critical and non-critical applications has an impact on the network contention these critical communications exhibit. So far, all existing mapping strategies have focused on inter-core communications. However, many-cores in embedded real-time systems can be integrated within backbone Ethernet networks, as they mostly provide Ethernet controllers as I/O interfaces. In this work, we first show that Ethernet packets can be dropped due to an internal congestion in the NoC, if these core-to-I/O communications are not taken into account while mapping applications. To solve this issue, we show on an avionic case study the benefits of the core-to-I/O contention-aware mapping strategy we propose.**

I. Introduction

The continuous need in increased computational power has fueled the on-going move to multi-core architectures in hard real-time systems. However, multi-core architectures are based on complex hardware mechanisms, such as for instance advanced branch predictors whose temporal behavior is difficult to master. Many-core architectures are instead based on simpler cores, so the timing predictability of cores are thus easier to analyze [10]. Besides, they should enable the safe simultaneous integration of both critical and non-critical applications [2]. Many-cores are thus promising hardware to host such a mix of real-time applications with different levels of criticalities. Note that we consider only two levels of criticalities in the remainder of this work: critical and non critical. The main challenge however lies in the ability to analyze the Worst-Case Traversal Time (WCTT) of critical flows exchanged within the Network-on-Chip (NoC). How hard real-time tasks that generate these critical flows, but also the non critical tasks, are mapped within cores, is therefore of utmost importance to control the contention over the NoC and thus the WCTT of flows. In the remainder of this paper, we simply say that we map flows over a NoC to avoid linking flows to tasks.

The efficiency of a mapping strategy over a NoC can be evaluated using different performance metrics. Hard-real time applications rely on the latency metric, as the goal is to minimize the WCTT of flows. Several contention aware mapping strategies (for instance [3], [11], [14]) have thus been proposed but for inter-core communications only. To the best of our knowledge, none consider communications between cores and with I/O interfaces, that we call core-to-I/O communications[1]. However, many-cores mainly provide DDR and Ethernet controllers only as I/O interfaces. For instance, Tilera [12] provides 3 Ethernet and 4 DDR controllers, while MPPA [5] from Kalray provides 8 Ethernet and 2 DDR controllers. Theses many-cores are thus more tailored to host embedded applications whose I/O data are exchanged using Ethernet packets. *In embedded real-time systems, many-cores can be used as processing elements within a backbone Ethernet network*, such as AFDX for the avionic domain or Ethernet AVB in the automotive field.

Mapping strategies for many-cores should therefore also take into consideration core-to-I/O communications, in addition to other core-to-core and core-to-memory communications. To demonstrate this strong requirement, the first contribution of this paper is to show that Ethernet packets can be dropped due to an NoC congestion, if I/O requirements are not taken into account when mapping applications. To this end, we rely on a case study from the avionic domain. It is made of a critical Full Authority Digital Engine (FADEC) application and a non-critical Health Monitoring (HM) application of the engine, used for recognizing incipient failure conditions. We thus propose an approach to map critical and non critical real-time applications over many-cores that reduces the WCTT of core-to-I/O communications. It is based on an existing strategy but in which we treat core-to-I/O communications as first class citizen. Our algorithm first assigns for each application to map a region within the NoC. Then, each task of an application is mapped within its region, so that the paths used by core-to-I/O communications from the Ethernet controller exhibit the lowest contention possible. We show for two variants of our case study that our algorithm successfully find a mapping that avoids Ethernet packets, whose payload are making the core-to-I/O communications, to be dropped. This demonstrates the benefits of our proposal compared to a state of the art mapping strategy that fails to do so.

II. Problem formulation

To illustrate the problem we address, we use an avionic case study made of a FADEC and an HM application. For the FADEC, 1270 bytes of sensors data from the engine are received by an Ethernet interface. These data are then divided and distributed to 6 tasks, noted t_{f0} to t_{f5}. These 6 tasks exchange 211 bytes of data between them. All these tasks also send 211 bytes of data to a last task noted t_{f6}. Then, t_{f6} stores 110 bytes within a DDR interface and sends 64 bytes of actuators data through an Ethernet interface. On the other hand,

[1] We use this term for both core communications from or to I/O interfaces.

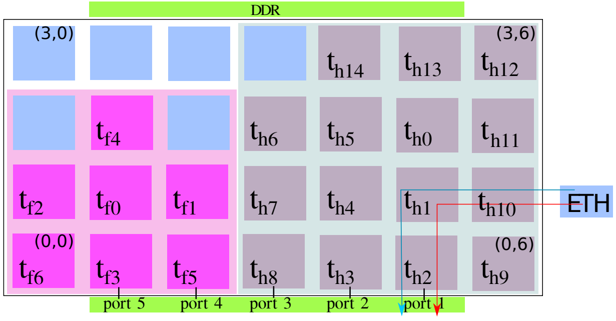


Fig. 1. Arbitrary mapping of FADEC and HM applications.

the HM application continuously receives through an Ethernet interface, a set of frames representing data to be processed in order to anticipate engine failures. The size of a frame is 130 KBytes and a set is made of 30 frames. When a set of frames is received, every two frames are assigned to a different task amongst 15 tasks, noted t_{h0} to t_{h14}. When the processing takes place, task t_{hi} also sends 112 bytes of data to t_{hi+1}, with $i \in [0, 14]$. Finally, all these tasks, finish their processing by storing their frames into the DDR.

Figure 1 shows an arbitrary mapping of these 2 applications over a 7×4 mesh NoC, shipped with a single giga-Ethernet interface. This Ethernet interface is thus shared between the two applications. This NoC configuration is not artificial as it can be seen as a subset of an initial larger $N \times N$ NoC that therefore leads to consider several instances of the problem we introduce in this section. A core of the NoC is identified by its (x,y) coordinates and we assume that (0,0) is located on the bottom left of the NoC. The square 3×3, whose left corner is located at (0,0) defines the regions where the tasks of the FADEC application are mapped. The square 4×4, located at (0,3), defines the region where the HM applications is mapped. Let us now describe the steps input I/O data received by an Ethernet interface go through in order to be used by a core of the NoC (blue and red arrows for respectively the HM and FADEC applications). We assume that Ethernet packets have a Maximum Transmit Unit (MTU) of 1500 bytes. We further assume that NoC packets can be made of up to 19 flow control digits (flits), as in the Tilera many-core. A flit is equal to 32 bits. The size of an Ethernet packet is thus generally several factors higher than the size of a NoC packet. Several NoC packets are therefore needed to transmit to a core an Ethernet payload, that must therefore be buffered within the Ethernet interface. Reaching a destination core can either be done directly or through an intermediate DDR controller. This choice is left to the user, and the last option is the case that we consider in this work. Note that I/O FADEC data are sent to the port 1 of the DDR, and can later be used from for instance either the port 4 or 5 of the DDR.

Flits of NoC packets are transmitted one by one by routers, i.e. in a pipeline way, by relying on wormhole switching strategy, with an dimension ordered XY routing policy and Round-Robin Arbitration (RRA) within routers. As routers have buffers of a few flits due to the area cost of memories in chips, flits of a same packet can thus be present on different routers. A NoC congestion occurs when a contention between two flows a at given router propagate backward due to the credit-based mechanism, preventing flits of other flows to also make progress. **In this work, we first consider the paths taken by flows originating from an Ethernet interface when reducing the contentions**, as we claim that many-core will be used as processing elements within a backbone Ethernet network. **If a NoC congestion occurs on one of these paths, the estimated WCTT of a flow can be higher than the arrival delay of the next incoming Ethernet packet**. In that case, this Ethernet packet may be dropped due to the lack of space in the Ethernet buffer. Previous Ethernet packets could indeed be stored in this buffer while waiting associated NoC flows can progress towards the DDR interface. Note that in this work we neglect the time a DDR request takes, however this delay would only increase the global one.

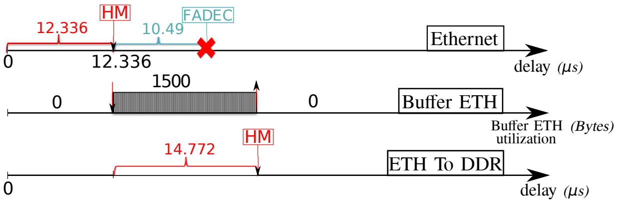


Fig. 2. Timeline of Ethernet and NoC packets showing that the FADEC Ethernet packet is dropped when the mapping of Figure 1 is used.

At both the Ethernet and NoC levels (and up to the DDR only), the Figure 2 shows the timeline of a frame from the HM application followed by one from the FADEC application. When the HM frame with payload size of 1500 bytes arrives at the single giga-Ethernet interface, i.e. after 12.336 μs (link traversal), it is stored into the interface buffer. Thus, if we consider that the Ethernet buffer size is 2 KB, as in Tilera, the buffer can only receive frames with 500 bytes of payload. Then FADEC frame can only be stored if all the HM frame has been transmitted to the DDR, as its size is 1270 bytes. The transmission of HM frame at the NoC level will use 20 packets of 19 flits and one packet of 15 flits. The WCTT of each HM packets on the NoC takes 701.5 ns. This value is obtained by reusing an existing method that gives the tightest WCTT values over RRA-based NoC [1]. Other strategies will therefore lead to even higher WCTT values and worsen the situation. We define the worst-case scenario as when each of these NoC packets will be blocked at each router by all NoC flows that can be encountered. So, as the HM frame is decomposed into 21 NOC packets, the global transmission delay of an HM frame to the DDR through the NoC will take $t_1 = 14.772$ μs. However, the transmission of the FADEC frame on Ethernet takes $t_2 = 10.49$ μs (transmission of 1270 bytes of payload at 1Gb/s). As the Ethernet buffer still contains the HM frame when the FADEC frame arrives, as $t_1 > t_2$, then FADEC frame is dropped. Therefore, the mapping proposed in Figure 1 does not take into account the I/O requirements leading to losing frames. The goal of the paper is then to propose a mapping approach considering the I/O transmission on the NoC.

III. Limitations of Existing Work

However, to the best of our knowledge, no mapping strategies for many-cores take into account core-to-I/O communications. We thus briefly review in this section existing contention-aware mapping strategies for core-to-core communications and discuss their limitations with respect to our problem.

Different strategies exist to reduce the number of contention flows on the path of the transmitted flows ([3], [11], [14], [13]) using minimization functions. However, all these approaches consider only the mapping of a single application.

Conversely, [6] considers the mapping of several applications by arbitrarily dividing the NoC into clusters. Each cluster is dedicated to an application. Then, within each cluster a congestion-aware mapping heuristic, similar to one in [14], minimizes the bandwidth utilization of NoC links. Authors of [4] enhance the definition of clusters for applications by making them near convex regions. Generating non-contiguous regions is avoided, thus reducing the congestion that can occur between applications, called the external congestion. But, due to the first core selection policy when building regions, mapping of an application over a fragmented region is possible [9]. In order to solve this problem, [9] propose to select the core having the most available neighbors (up to 4) to avoid region fragmentation and thus decrease both internal and external congestions. This solution, called CoNA, only considers direct neighbors when selecting the first core and then still lead to fragmentation of areas [7]. Smart Hill Climbing (SHiC) approach [7] considers a new metric called square-factor (SF) for selecting the first core when building regions. The SF of a core is the maximal size of the square area in which that core can be put in, to which the number of free cores around this square is added. The first core is then the one having a SF greater or equal to the size of the application to be mapped, i.e. the number of cores that are needed assuming a core can only execute a single task. [8] adapts SHiC so that contiguous regions as used to map critical applications, in order to reduce contentions, while non-critical applications are mapped over non-contiguous regions to increase the system throughput.

SHiC is the best congestion-aware mapping approach which is the closest related work to ours. The mapping done in Figure 1 has been in fact obtained using SHiC. As we can see, SHiC, and also all others existing mapping strategies, does not consider where I/O interfaces are located within the NoC when mapping applications. The NoC core-to-I/O flows may therefore suffer from external congestions, if the applications are not mapped close to the I/O interfaces they use. However, these interfaces could be shared between several applications. Besides, the internal mapping of applications also influence the WCTT of these core-to-I/O flows. Mapping the most communicating task to the first selected core, as most existing strategies do, may no longer be appropriate. The number of contentions core-to-I/O flows experiences should instead be reduced, so that their WCTT are decreased and avoid dropping incoming I/O packet. Finally, SHiC defines strict contiguous regions preventing the mapping of a set of applications whose total size is equal to the size of NoC.

IV. Proposal overview

To overcome these limitations, our approach also relies on a two steps process to map applications of different levels of criticalities. In this paper, instead of giving the mapping algorithms, we propose to describe our approach using on overview of these two steps. We currently assume a single Ethernet controller for the I/O interface, a single critical application amongst a set of non-critical applications and a single core-to-I/O flow per application, called core-to-Ethernet.

The first step of our mapping process, called the *external mapping*, assigns to each application a region and determine its shape. Our second step then maps the tasks of each application within its assigned region. When the sum of the size of each application is equal to the size of the NoC, i.e there is no free cores available, the external mapping starts with the critical application and assigns to it a region next to the Ethernet interface. In the other case, the external mapping consider the non-critical applications first. It starts to map them from the opposite side where the Ethernet interface is located. The critical application is thus the last one to be considered, in order to gather remaining free cores within its corresponding region. These cores will be used to provide more laxity in the second step of our mapping process when mapping the critical application. When assigning a region to an application, we consider the minimal rectangular shape that corresponds to its size and favors biggest shapes in next assignment of regions.

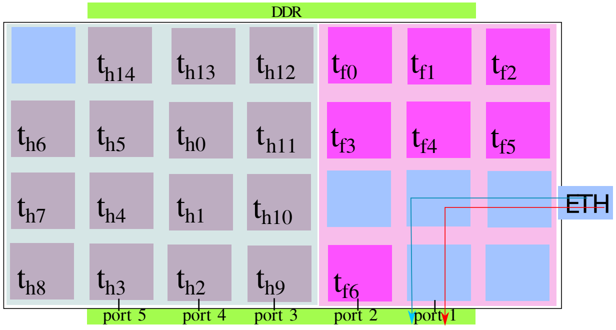


Fig. 3. Mapping of the FADEC and HM applications using our approach.

The second step of our mapping process is called the *internal mapping*. When mapping applications, its goal is to reduce the WCTT of NoC core-to-I/O flows for both the critical and non-critical applications. To this end, the internal mapping reduces the number of contentions on the paths taken by these flows according to different specific rules. On the example of Figure 3, the available free cores within the region of the critical application are put in priority on the path taken by core-to-I/O flow. Note that for application that are not mapped on the paths taken by core-to-I/O flows, we rely on the SHiC strategy.

Figure 3 shows the final mapping that our approach generates for our case study. Remember that it is composed of 28 cores, as it is a 7×4 mesh NoC. The FADEC application requires 7 cores, while the HM application requires 15 cores. The total size of both applications is thus 22, leaving initially 6 free cores. The external mapping therefore starts by assigning a region to the HM application. To this end, a 4×4 square region is defined and located at (0,0). One free core is thus lost and left unused when defining this region. The FADEC application is then assigned a 3×4 rectangular region, located at (0,4), and that integrates the 5 remaining free cores. The internal mapping uses 4 free cores, amongst the 5 available in its region, in a 2×2 square area next to the Ethernet controller.

V. Preliminary evaluation

First, let us assume that the internal mapping maps the applications by following the SHiC strategy. Compared to the mapping shown by Figure 1, the mapping of applications is thus simply permuted. For this mapping, the WCTT for reaching the DDR by a NoC packet of the core-to-Ethernet flow of the HM application is 610 ns. This leads to a WCTT for the core-to-Ethernet flow of 12.8 μs. This delay is thus reduced compared to the one shown by Figure 2. However, it is still higher than what is required to avoid dropping the the FADEC Ethernet packet. This demonstrates the need for a

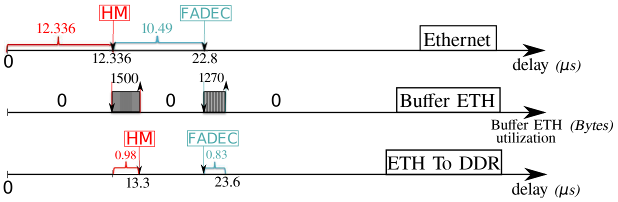


Fig. 4. Timeline of Ethernet and NoC packets showing that the FADEC Ethernet packet is no longer dropped when mapped using our approach.

strategy in the internal mapping that further reduces the WCTT of the NoC core-to-I/O flows of non-critical applications, in addition to the critical one.

Let us now assume that our internal mapping takes advantage of the free cores gathered by our external mapping. This corresponds to the mapping shown by Figure 3. The timeline of Figure 4 shows that the FADEC Ethernet packet is no longer dropped in that case. The free cores are indeed located on the path taken by the Ethernet-to-core NoC flows of the FADEC and HM applications. These flows are thus no longer blocked while progressing towards the DDR. The WCTT of NoC core-to-Ethernet packets (made of 19 flits) from the HM application is thus reduced to $47.15\ ns$. The global WCTT of the corresponding NoC flow of the HM application is then $0.98\ \mu$s thanks to our approach where the internal mapping reduces the number of blocking flows with the core-to-Ethernet flow. The HM Ethernet packet is therefore removed from the Ethernet buffer before the FADEC Ethernet packet arrives to the Ethernet interface. The global WCTT of the NoC core-to-Ethernet flow from the FADEC application is $0.83\ \mu$s.

We also evaluated our approach when all the cores of the NoC are used. To this end, we added two tasks t_{f7} and t_{f8} to the FADEC application. These additional tasks have the same characteristics as the tasks t_{f0} to t_{f5}. For the HM application, we reduced the number of tasks from 15 to 12 tasks. We assume a reduced 7×3 mesh NoC. Compared to the mapping shown by Figure 1, SHiC maps t_{f7} and t_{f8} at respectively (2,0) and (2,2). In this case, the same timeline as the one of Figure 2 is obtained. The FADEC Ethernet frame is thus still dropped. When however considering our approach, the external mapping assigns to the FADEC application a 3×3 square region next to the Ethernet interface. The internal mapping of the FADEC application leads to a WCTT of $483.5\ ns$ for the NoC core-to-Ethernet packets of the HM application. This corresponds to a global WCTT of $10.15\ \mu$s for reaching the DDR. The FADEC frame can thus reaches the Ethernet interface after the removal of the HM Ethernet packet from the Ethernet buffer. This example shows that our approach seems promising even if all the core of the NoC are used.

VI. Conclusion and Future Work

Existing contention-aware mapping strategies aim to minimize the inter-core congestion without taking into account requirements of I/O communications of applications. However, a NoC is mostly connected to other external systems through several Ethernet interfaces. The WCTT of NoC core-to-Ethernet flows depends on the congestions generated by the mapping of both critical and non-critical applications.

In this paper, we first show on an avionic case study that the solution generated by a state of the art contention-aware mapping strategies even lead to drop Ethernet packets used by a critical application, when mapped together with a non-critical application. We then show on the same case study that a two steps mapping strategy solve this issue. An external mapping step assigns the critical application near the considered Ethernet interface and reclaim free cores for the definition of the region where the critical application will be mapped in. Within this region, an internal mapping step reduces the contention both NoC core-to-Ethernet critical and non-critical flows experience. Work underway shows that our approach can be successfully applied over other case studies. Next step is the generalization and formalization of the internal mapping rules.

Further work includes the development of a software tool implementing our mapping strategy as well as consider all types of flow in our mapping strategy. We are also interested in generalizing our algorithm by supporting additional core-to-I/O flows, several critical applications and I/O interfaces.

References

[1] L. Abdallah, M. Jan, J. Ermont, and C. Fraboul. Wormhole networks properties and their use for optimizing worst case delay analysis of many-cores. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 59–68, Siegen, Germany, June 2015.

[2] A. Burns, J. Harbin, and L. S. Indrusiak. A wormhole noc protocol for mixed criticality systems. In *Proc. of the IEEE 35th Real-Time Systems Symposium, RTSS*, pages 184–195, Rome, Italy, December 2014.

[3] C.-L. Chou and R. Marculescu. Contention-aware application mapping for network-on-chip communication architectures. In *IEEE Intl. Conf. on Computer Design (ICCD)*, pages 164–169, 2008.

[4] C.-L. Chou, U. Y. Ogras, and R. Marculescu. Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1866–1879, 2008.

[5] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager. Time-critical computing on a single-chip massively parallel processor. In *Proc. of the Conf. on Design, Automation & Test in Europe (DATE'14)*, pages 97:1–97:6, 2014.

[6] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes. Dynamic task mapping for mpsocs. *Design & Test of Computers*, 27(5):26–35, 2010.

[7] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila. Smart hill climbing for agile dynamic mapping in many-core systems. In *Proc. of the 50th Annual Design Automation Conference*, page 39, 2013.

[8] M. Fattah, A.-M. Rahmani, T. C. Xu, A. Kanduri, P. Liljeberg, J. Plosila, and H. Tenhunen. Mixed-criticality run-time task mapping for noc-based many-core systems. In *22nd Euromicro Intl. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*, pages 458–465. IEEE, 2014.

[9] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila. Cona: Dynamic application mapping for congestion reduction in many-core systems. In *30th Intl. Conf. on Computer Design (ICCD)*, pages 364–370, 2012.

[10] V. Nélis, P. M. Yomsi, L. M. Pinho, J. C. Fonseca, M. Bertogna, E. Quiñones, R. Vargas, and A. Marongiu. The Challenge of Time-Predictability in Modern Many-Core Architectures. In *14th Intl. Workshop on Worst-Case Execution Time Analysis*, pages 63–72, Madrid, Spain, July 2014.

[11] A. Racu and L. S. Indrusiak. Using genetic algorithms to map hard real-time on noc-based systems. In *7th Intl. Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8, 2012.

[12] Tilera corporation. *Tile processor user architecture manual*, Nov. 2011. UG101.

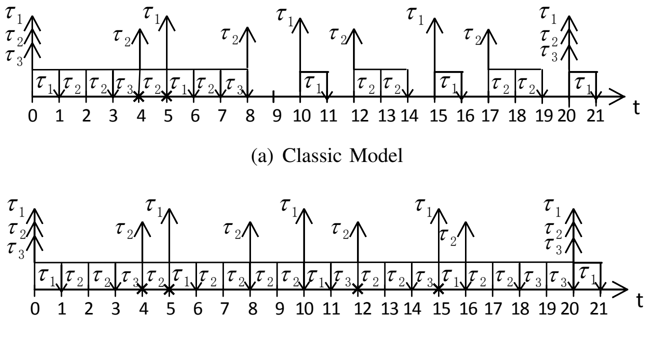
[13] B. Yang, L. Guang, T. Säntti, and J. Plosila. Tree-model based contention-aware task mapping on many-core networks-on-chip. *Communications in Information Science and Management Engineering*, 2012.

[14] C. Zimmer and F. Mueller. Low contention mapping of real-time tasks onto tilepro 64 core processors. In *18th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 131–140, 2012.

Memory-aware Response Time Analysis for P-FRP Tasks

Xingliang Zou, Albert M. K. Cheng
Department of Computer Science
University of Houston
Houston, TX, 77004, USA
Email: xzou@uh.edu, cheng@cs.uh.edu

***Abstract*—Functional Reactive Programming (FRP) is playing and potentially going to play a more important role in real-time systems. Priority-based (preemptive) FRP (P-FRP), a variant of FRP with more real-time characteristics, demands more research in its scheduling and timing analysis. In a P-FRP system, similar to a classic preemptive system, a higher-priority task can preempt a lower-priority one and make the latter aborted. The lower-priority task will restart after higher-priority tasks complete their execution. Unlike the classic preemptive model, when a task aborts, all the changes made by the task are discarded (abort and restart). In previous studies, the value of Worst Case Execution Time (WCET) of a task is used for all its restarted tasks. However, in practice the restarted tasks likely consume less time than the WCET when considering the memory effect such as cache-hit in loading code and data. In this work, we use different task execution time for restarted tasks when conducting schedulability and response time analysis for P-FRP tasks.**



(a) Classic Model

(b) P-FRP Model

Fig. 1: Schedules of fixed priority task set ($C_1 = 1, C_2 = 2, C_3 = 2, T_1 = 5, T_2 = 4, T_3 = 20$)

I. Introduction

There are two distinct types of programming paradigms in computer programming: imperative and functional. Functional Programming has a distinct difference from imperative programming in that it is immune to side-effects caused by using states and mutable data. Since the formal system of λ-calculus (lambda-calculus) was first devised by Church [1] and Kleene [11], many functional programming languages have been invented: LISP, Ocaml, Haskell, Scheme, Erlang, F#, Atom, Scala and so on. Haskell and Erlang have been studied and commercially developed intensively. Scala is recently adopted by companies such as Linkedin, Twitter and Walmart [7]. Functional Reactive Programming (FRP) [16] is a framework for constructing reactive applications using the building blocks of functional programming.

In real-time systems, the correctness of a program is measured by its logical output as well as its ability to complete within certain time limits. FRP is demonstrated to be effective in modeling and building reactive systems such as graphics, robotic and vision applications. However, another significant feature of real-time systems, priority, is not considered in FRP. To address this problem, the P-FRP [10] model has been proposed as a variant of the FRP model. P-FRP maintains both the type-safety and the state-less execution paradigm of FRP, and supports priorities assigned to different tasks while not requiring the use of synchronization mechanisms between tasks in the system. It has the potential to transform the building of more and more complicated Cyber Physical Systems (CPSs). Christoffersen and Cheng [8] presented an impact of P-FRP in building controllers in automobile anti-lock brake systems.

In order to maintain the state-less paradigm of the FRP model, unlike the classic preemptive scheduling (shortened to classic model) (Fig.1.(a)), P-FRP uses an Abort and Restart (Fig.1.(b)) semantics where if a lower-priority task is interrupted by a higher-priority task, it has to restart from the beginning when it is resumed. Here we consider a typical task life cycle without being interrupted (*cold started* task): (1) code is loaded from hard drive and data is loaded from external memory; (2) computation is done by processor(s); (3) results are committed to external memory. In the P-FRP model, the time spent in phase (2) and (3) is wasted when a task is aborted, however, since the existence of memory hierarchy, the time spent in phase (1) can be less when a task is restarted, for example, the task code is still in cache and does not need to be read from slow external memory again. This memory effect is not considered in previous studies of P-FRP systems. In this paper, we present our preliminary memory-aware P-FRP task response time analysis and experimental results.

II. System Model and Related Works

We consider a hard real-time uniprocessor system with hierarchical memory components. n-task system Γ_n =

*The work is supported in part by the United States National Science Foundation (NSF) under awards No. 0720856 and No. 1219082.

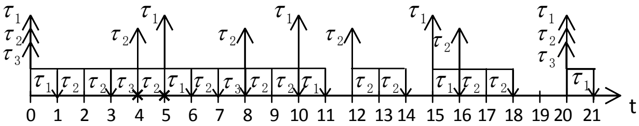
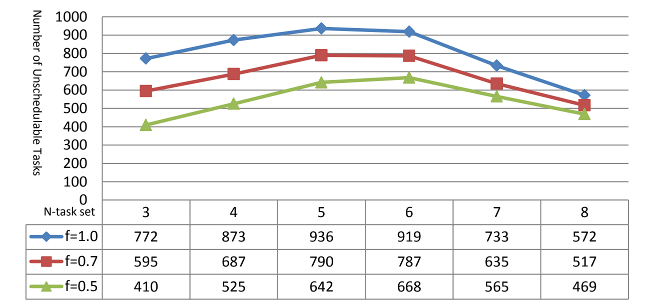

τ_1
τ_2
τ_3
τ_1 τ_2 τ_2 τ_3 τ_2 τ_1 τ_2 τ_3 τ_2 τ_2 τ_1 τ_2 τ_2 τ_1 τ_2 τ_2 τ_1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
t

Fig. 2: Memory-aware P-FRP scheduling of fixed priority task set ($C_1^1 = 1, C_1^2 = 1; C_2^1 = 2, C_2^2 = 1; C_3^1 = 2, C_3^2 = 1; T_1 = 5, T_2 = 4, T_3 = 20$)

$\{\tau_1, \tau_2, ..., \tau_n\}$ of periodic tasks with fixed priorities is scheduled in the P-FRP abort and restart model. Task τ_i is assigned a unique fixed priority i ($1 \leq i \leq n$), where 1 is the highest priority and n is the lowest. An instance or invocation of a periodic task is called a job. J_i^k refers to the k-th job of τ_i. Each periodic task τ_i is characterized by a constant arrival time period T_i between two successive jobs of the task, a relative deadline $D_i(D_i \leq T_i)$, and *two* computation times: C_i^1 and C_i^2, the execution time of the task in *cold started* and *restarted* modes respectively. The Response Time (RT) of a job is the time between the release of a job and its completion. The response time of a task in a given priority assignment is the largest response time among those of all its jobs. A task is referred to as schedulable according to a given scheduling policy if its response time under the given scheduling policy is less than or equal to its deadline. A task set is referred to as schedulable according to a given scheduling policy if all of the tasks in the task set are schedulable under the given scheduling policy.

Jiang *et al* presented their research on P-FRP task schedulability analysis in [13][14] to find tighter feasibility intervals. Belwal and Cheng have shown in [3] that RM is not optimal in P-FRP systems with synchronous release and it is even unknown if there exists an optimal one other than an exhaustive test over all possible priority assignment algorithms. Belwal and Cheng [2] presented a utilization-based analysis that the current schedulability condition only holds true with the utilization bound of $1/n$ under certain restrictions on periods and release scenarios. Wong *et al* [5] conducted research on other priority assignment algorithms: Utilisation Monotonic (UM), Execution-time Monotonic (EM), and a combination of UM and EM, Execution-time-toward-Utilisation Monotonic (EUM) priority assignment algorithm. By comparing with an Exhaustive Search schema, they confirmed that none of RM, DM, EM or EUM is optimal for the P-FRP model. Zhou *et al* [9] presented their research on WCRT and schedulability analysis for real-time software transactional memory-lazy conflict detection for P-FRP tasks. Wong *et al* [6] proposed the Deferred Abort model to reduce the number of preemptions in scheduling P-FRP tasks. Zou *et al* [12] proposed a non-work-conserving Deferred Start model to eliminate preemptions in scheduling P-FRP tasks. However, none of these researches considers the different execution time of *cold* started tasks and restarted tasks. On the other hand, Kazemi and Cheng [15] studied the P-FRP task execution time on a scratchpad memory-based platform, and showed that the task execution time changes because of the memory hierarchy.



N-task set	3	4	5	6	7	8
f=1.0	772	873	936	919	733	572
f=0.7	595	687	790	787	635	517
f=0.5	410	525	642	668	565	469

Fig. 3: Number of Unschedulable Task Sets

III. OUR WORK

A. Memory-aware P-FRP model

In Fig.1, we schedule three tasks with a given priority assignment under the classic model and the P-FRP model. If the execution time of a task in *cold* start and restart cases are different, we will have a memory-aware P-FRP model scheduling as Fig.2 shows.

In Fig.2, the second job of τ_2, J_2^2, is preempted at time point 5 since it requires 2 time units of *cold start execution time* (C_2^1) and executes only 1 time unit when the second job of the highest priority task τ_1 is scheduled to execution. And when J_2^2 is resumed at time point 6, it needs only 1 time unit of *restart execution time* (C_2^2) and hence finishes execution at time point 7. For the job J_2^3 that arrives at time point 8, it requires 2 time units of *cold start execution time* and finishes execution at time point 10. The similar scheduling applies to task τ_3. We can see, under the same priority assignment and scheduling policy, compared to Fig.1(b) of P-FRP scheduling, in time interval of [0, 20) the number of preemptions is reduced from 4 to 2, the CPU idle time unit is increased from 0 to 4. Also, the response time of τ_2 is reduced from 4 to 3, and the response time of τ_3 is reduced from 20 to 8. Thus the memory-aware P-FRP scheduling saves CPU time and potentially is able to schedule more tasks.

B. Experiment and Result

The experiments are designed and conducted on a desktop computer with a CPU of i3-4130 3.4GHz, 8 GB memory and *Ubuntu 14.04.3 LTS 64-bit Desktop* operating system. We generate task sets and run the P-FRP scheduling with and without considering memory effect. The task sets we generated have 3, 4, ..., 8 tasks respectively; the periods are randomly generated in range of $[15, 75]$; the total utilization of a task set is 0.6. We use the *UUniFast* algorithm [4] to generate n ($n = 3, 4, .., 8$) utilization factors U_i ($1 \leq i \leq n$) in a descending order such that the total utilization $U = \sum_{i=1}^{n} U_i$ equal to the given value, 0.6 in this experiment. We then shuffle those U_i to a random order for the consideration of generalization. The computation time of each task is computed as $C_i^1 = U_i * T_i$, $C_i^2 = C_i^1 * f$, where f is a factor that shows the difference of task execution time in *cold* start and restart. f is 0.7 and 0.5 in our experiments. As comparison, we also run the task sets in original P-FRP model, which is equivalent to *f=1.0*. For each n-task set, we generate 1000 task sets.

Fig.3 shows the number of unschedulable task sets with different f for the n-task sets ($n = 3, 4, .., 8$) we generated. The case of *f=0.7* has 22.9%, 21.3%, 15.6%, 14.4%, 13.4% and 9.6% less unschedulable task sets compared to the original P-FRP scheduling. The case of *f=0.5* has 46.9%, 39.9%, 31.4%, 27.3%, 22.9% and 18.0% less unschedulable task sets compared to the original P-FRP scheduling. Thus the memory effect of restarted P-FRP tasks must not be ignored.

IV. Conclusion and Future Work

We have proposed our preliminary research of memory-aware P-FRP model. Simulation results show the schedulability and response time improvement when considering the different execution time of a task in *cold* start and restart cases. Our ongoing research is to present more theoretical response time analysis and priority assignment research in the memory-aware P-FRP task scheduling. And since the execution time difference is likely related to data placement/locality, we will address this difference in our multi-core P-FRP task scheduling research too.

References

[1] A. Church. An unsolvable problem of elementary number theory. American Journal of Mathematics 58, pp.345-363, 1936.

[2] C. Belwal, A. M. K. Cheng. A Utilization based Sufficient Condition for P-FRP. 9th IEEE/IFIP Int'l Conf. on Embedded and Ubiquitous Computing (EUC), 2011, pp.237-242.

[3] C. Belwal, A.M.K. Cheng. On priority assignment in P-FRP. RTAS 2010 WiP Session.

[4] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. Real-Time System, 30(1-2):129-154, 2005.

[5] H. C. Wong, A. Burns. Improved Priority Assignment for the Abortand-Restart (AR) Model. Technical Report YCS-2013-481, University of York, Department of Computer Science, 2013.

[6] H. C. Wong, A. Burns. Priority-based Functional Reactive Programming (P-FRP) using Deferred Abort. RTCSA 2015.

[7] https://typesafe.com/company/casestudies, 09/24/2014.

[8] K. R. Christoffersen, A. M. K. Cheng. Model-based design: Antilock brake system with priority-based functional reactive programming. RTSS 2013 WiP Session.

[9] Q. Zhou, Y. Li, X. Zou, A. M. K. Cheng, Y. Jiang. Worst Case Response Time and Schedulability Analysis for Real-Time Software Transactional Memory-Lazy Conflict Detection (STM-LCD). DPRTCPS workshop, 2015.

[10] R. Kaiabachev, W. Taha, A. Zhu. E-FRP with priorities. ACM EMSOFT 2007.

[11] S. Kleene. A theory of positive integers in formal logic. American Journal of Mathematics 57, pp.153-173 and 219-244, 1935.

[12] X. Zou, A. M. K. Cheng, Y. Jiang. A Non-Work-Conserving Model for P-FRP Fixed Priority Task Scheduling. RTSS 2015 WiP session.

[13] Y. Jiang, A. M. K. Cheng, X. Zou. Schedulability Analysis for Real-Time P-FRP Tasks Under Fixed Priority Scheduling. RTCSA 2015.

[14] Y. Jiang, Q. Zhou, X. Zou, A. M. K. Cheng, X. Zou. Minimal Schedulability Testing Interval for Real-Time Periodic Tasks with Arbitrary Release Offsets. IEEE ICESS 2014.

[15] Z. Kazemi, A. M. K. Cheng. A Scratchpad Memory-Based Execution Platform for Functional Reactive System and its Static Timing Analysis. RTAS 2015 WiP session.

[16] Z. Wan, P. Hudak. Functional reactive programming from first principles. ACM SIGPLAN PLDI 2000.

Cache Persistence Aware Response Time Analysis for Fixed Priority Preemptive Systems

Syed Aftab Rashid, Geoffrey Nelissen, Eduardo Tovar
CISTER/INESC TEC, ISEP
Polytechnic Institute of Porto, Portugal

***Abstract*—A task can be preempted by several jobs of a higher priority task during its response time. Assuming the worst-case memory demand for each of these jobs leads to pessimistic worst-case response time (WCRT) estimations. Indeed, there is a high chance that a big portion of the instructions and data associated with the preempting task τ_j, are still available in the cache when τ_j releases its next jobs. We call this content "persistent cache blocks" (PCBs). Accounting for PCBs in the memory demand of the preempting task allows to significantly reduce the pessimism on the total memory demand considered by the WCRT analysis. In this work, we propose a refined WCRT analysis for fixed priority preemptive systems considering (i) the effect of PCBs on the memory demand of the preempting task, and (ii) accounting for the number of PCBs that can be evicted by the preempted tasks between two successive job releases of the preempting tasks.**

I. Introduction

The existing gap between the processor and main memory operating speeds necessitates the use of intermediate cache memories to accelerate the average case access time to instructions and data that must be executed or treated on the processor. The introduction of cache memories in modern computing platforms is the cause of big variations in the execution time of each instruction depending on whether the instruction and the data it treats are already loaded in the cache (cache hit) or not (cache miss).

These last years, a lot of attention has been placed on the analysis of the impact of preemptions on the worst-case execution time (WCET) and worst-case response time (WCRT) of tasks in systems where preemptions are allowed. Indeed, the preempted tasks may suffer additional cache misses if memory blocks are evicted from the cache during the execution of the preempting tasks. These evictions cause extra accesses to the main memory, which result in additional delays in the task execution. This extra-cost is usually referred to as cache-related preemption delays (CRPDs).

Over the years, different approaches have been proposed to counter the effect of preemptions. Some (e.g., [1], [2]) use non- or limited preemption scheduling schemes to eliminate or reduce the number of preemptions. Others [3]–[9] use information about the task memory access pattern to bound and incorporate the preemption costs into the WCET and the WCRT analyses. In this work, we focus on the latter and propose a method, complementary to [3]–[9], to bound the memory overhead during the task response time.

Several approaches have been proposed in the literature to compute accurate bounds on CRPDs. They are based on the study of memory access patterns of the preempted task [4], the preempting tasks [3], [5], or both [5]–[9]. However, all these approaches still result in pessimistic WCRT bounds due to the fact that they only consider the effect of preemptions on the memory demand of the preempted task but do not consider the variation in the memory demand of the preempting tasks. They all assume that every job of a high priority task τ_j preempting a low priority task τ_i will ask for its maximum memory demand, i.e., its worst-case memory demand in isolation. Although true for the first job released by the preempting task τ_j, subsequent jobs of τ_j may re-use most of the data and instructions that were already loaded in the cache during the execution of its previous jobs. Noticeably, the memory blocks loaded by a job $J_{j,k}$ of τ_j remain in the cache until the execution of the next job $J_{j,k+1}$ of τ_j unless evicted by any other task executing between the completion of $J_{j,k}$ and the beginning of the execution of $J_{j,k+1}$.

Therefore, in order to propose tighter bounds on the memory overhead, and hence on the WCRT of each task τ_i executing in a preemptive system, we (i) model the impact of persistent cache contents associated with each preempting tasks on the WCRT of the preempted task, and (ii) analyze the effect of the preempted task cache accesses on the memory demand of the preempting tasks.

II. System Model

This work targets single-core platforms with a single level (L1) data/instruction cache. The cache is assumed to be direct-mapped, that is, each memory block in the main memory can be mapped to only one block in the cache.

We consider sporadic tasks with constrained deadlines where each task has a fixed priority. Any priority assignment scheme (e.g., Rate Monotonic [10]) is acceptable. We also assume that the tasks are independent and do not suspend themselves during their execution. A task τ_i is defined by a triplet (C_i, T_i, D_i); where C_i is the WCET of τ_i, T_i is its minimum inter-arrival time and D_i is the relative deadline of each instance (or job) of τ_i. We assume that the tasks have constrained deadlines, i.e., $D_i \leq T_i$. In this work, we further decompose each task WCET into two terms, namely, the worst-case processing demand P_i and the worst-case memory demand MD_i. P_i denotes the worse case execution time of τ_i considering that every memory access is a cache hit. Consequently, it only accounts for execution requirements of the task and does not include the time needed to fetch data and instructions from the main memory. MD_i is the worst-case memory demand of any job of task τ_i, that is, it is the maximum amount of time during which any job of τ_i is performing memory operations. Because the worst-case processing demand and the worst-case memory demand may not necessarily be experienced on the same execution path of τ_i, it results that $C_i \leq P_i + MD_i$. The WCRT of task τ_i is denoted by R_i and is defined as the longest amount of time between the arrival and the completion of any of its jobs. A task τ_i is said to be schedulable if $R_i \leq D_i$. Similarly, a task set is schedulable if all of its tasks are schedulable.

In this work, we consider that preemption costs only refer to additional cache reloads due to those preemptions. Other overheads due to context switches, scheduler invocations and pipeline flushes are assumed to be included in the WCET.

For notational convenience, we define the following task sets:

- $hp(i)$: the set of tasks with a priority higher than that of τ_i.
- $hep(i)$: the set of tasks with priorities higher than or equal to that of τ_i.
- $\mathit{aff}(i,j)$: the set of tasks with priorities higher than or equal to the priority τ_i but strictly lower than that of τ_j. This set contains the intermediate priority tasks, which may affect the response time of τ_i but may also be preempted by τ_j.

III. State of the Art

As already explained in the introduction, when a task τ_i is preempted by a higher priority task τ_j, it is likely that τ_j will evict memory blocks of τ_i from the cache. On resumption, τ_i might consequently require to reload cache blocks from the main memory along with its normal memory requirements. This CRPD caused by τ_j on τ_i is denoted by $\gamma_{i,j}$. Several methods have been proposed in the literature to compute $\gamma_{i,j}$. In one of the earlier works, Lee et al. [4] introduced the concept of *useful cache block* (UCB). As defined in [9], *"a memory block m is called a useful cache block (UCB) at program point P, if it is cached at P and will be reused at program point Q that may be reached from P without eviction of m"*. Lee et al. [4] used the maximum number of UCBs among all the tasks in $\mathit{aff}(i,j)$ to upper bound the preemption cost $\gamma_{i,j}$. Busquets et al. [3] and Tomiyama et al. [5] rather used the notion of *evicting cache block* (ECB), i.e, any cache block accessed during the execution of the task and which can then evict the memory block cached by another task, to upper bound the preemption cost that can be caused by each preempting task. Other approaches by Tan and Mooney [7], Staschulat et al. [6] and Altmeyer et al. [8] used both the UCBs of the preempted tasks and ECBs of the preempting tasks in order to come up with more precise bounds on the preemption cost. Notably, the ECB and UCB-union and the multi-set approaches presented in [8] and [9] dominate all the existing approaches for CRPD calculation. We first detail the ECB-union approach and then the UCB-union multi-set. Readers are referred to [9] for the description of UCB-union and ECB-union multi-set approaches.

The ECB-union approach [8] uses the ECBs of all tasks in $hep(j)$ maximized over the UCBs of tasks in $\mathit{aff}(i,j)$ to calculate the preemption cost $\gamma_{i,j}$. The resulting value for the preemption cost, denoted as $\gamma^{ecb}_{i,j}$, is given by

$$\gamma^{ecb}_{i,j} = d_{mem} \times \max_{\forall k \in \mathit{aff}(i,j)} \left(\left| UCB_k \cap \Big(\bigcup_{\forall h \in hep(j)} ECB_h \Big) \right| \right) \tag{1}$$

where d_{mem} is the time required to reload one memory block from the main memory to the cache, and UCB_k and ECB_j are the sets of UCBs and ECBs of task τ_k and τ_j, respectively. The preemption cost can then be accounted for in the WCRT analysis using the following formulation:

$$R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \times (C_j + \gamma^{ecb}_{i,j}) \tag{2}$$

When combined, the ECB and UCB-union approaches provide a reasonably precise upper bound on the preemption cost. However, it can also lead to overestimations in different situations as shown in [9]. To further reduce the pessimism associated to the ECB and UCB-union approaches, Altmeyer et al. [9] proposed two new solutions, namely, the UCB-union multi-set and the ECB-union multi-set approaches. These multi-set versions of the UCB-union and ECB-union approaches additionally take into account the maximum number of jobs $E_j(R_i) \overset{\text{def}}{=} \left\lceil \frac{R_i}{T_j} \right\rceil$ that each higher priority task τ_j can release during the response time of τ_i. Under that framework, the WCRT equation becomes:

$$R_i^{k+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^k}{T_j} \right\rceil \times C_j + \sum_{\forall j \in hp(i)} \gamma^{mul}_{i,j} \tag{3}$$

where $\gamma^{mul}_{i,j}$ accounts for the total preemption cost that can be caused by all the jobs of τ_j released during the response time of τ_i. Using the UCB-union multi-set approach $\gamma^{mul}_{i,j}$ is upper bounded by $\gamma^{ucb-m}_{i,j}$ defined as follows:

$$\gamma^{ucb-m}_{i,j} = d_{mem} \times \left| M^{ucb}_{i,j} \cap M^{ecb}_{i,j} \right| \tag{4}$$

where $M^{ucb}_{i,j}$ and $M^{ecb}_{i,j}$ are multi-sets defined as

$$M^{ucb}_{i,j} = \bigcup_{\forall k \in \mathit{aff}(i,j)} \left(\bigcup_{E_j(R_k)E_k(R_i)} UCB_k \right) \tag{5}$$

and

$$M^{ecb}_{i,j} = \bigcup_{E_j(R_i)} ECB_j \tag{6}$$

Note that the ECB-union multi-set approach dominates the ECB-union approach [8] whereas the UCB-union multi-set approach dominates the UCB-union approach [7]. Yet, it is shown in [9] that the ECB-union and UCB-union multi-set approaches are incomparable.

For a more detailed description on the formulation of Equations (2) to (6), the reader is referred to [9].

IV. Motivational Example

As presented in the previous section, the impact of a high priority task τ_j on the WCRT of lower priority task τ_i can be estimated in a fairly accurate manner by analyzing the mapping of UCBs and ECBs in the cache. The impact of τ_i on the memory demand of τ_j is however ignored during the WCRT analysis of τ_i. Yet, high priority tasks may often execute more than one job during the response time of a lower priority task. Therefore, to accurately estimate the WCRT of a low priority task τ_i, one must consider the impact of the preempted tasks on the memory demand of each job released by the preempting tasks. In the literature, this is dealt with by assuming that the memory demand for each job of a high priority task τ_j executing within the response time of a low priority task τ_i is always maximum, i.e, equal to the maximum memory demand MD_j. As a result, the total memory overhead MO_i that must be accounted by τ_i during its WCRT is upper bounded by the following equation derived in [11].

$$MO_i = MD_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times (MD_j + \gamma_{i,j}) \qquad (7)$$

There is a significant level of pessimism involved in Equation (7) as we will demonstrate using the example given below.

Example 1. *Consider the two tasks τ_1 and τ_2 (where τ_1 has a higher priority than τ_2) presented in Fig. 1. We assume that the time d_{mem} needed to access the main memory and load a memory block to the cache is equal to 1 time unit, and that the memory demand of τ_1 and τ_2 are $MD_1 = 6$ and $MD_2 = 8$[1], respectively. We also assume that the memory block $\{9\}$ accessed by τ_1 contains some data that must be reloaded at the beginning of each of its job's execution. Fig. 1 depicts a possible schedule together with the evolution of the cache content over time. The memory blocks that must be loaded/reloaded from the main memory after each preemption or resumption are shown on a grey background.*

Initially, the cache is empty and τ_2 loads all its ECBs from the main memory as soon as it starts to execute. When τ_1 preempts τ_2 for the first time, it loads $MD_1 = 6$ memory blocks into the cache. Since there is an overlap between the ECBs of τ_1 and the UCBs of τ_2, τ_1 evicts some of the useful cache blocks of τ_2. When τ_2 resumes its execution, it has to reload $\gamma_{2,1} = 2$ cache blocks from the main memory. As the second job of τ_1 preempts τ_2, one can notice that its memory demand is no longer equal to MD_1. In fact, most of the memory blocks needed by τ_1 are still in the cache. As a consequence, τ_1 must only reload the memory blocks $\{5, 6\}$ which have been evicted by τ_2, as well as the memory block $\{9\}$ which must be reloaded at each new job execution. The same scenario happens for all the jobs released by τ_1 at the exception of the first one. Therefore, the actual memory demand for the second and third job of τ_1 is much less (i.e., 3) than $MD_1 = 6$.

In the presented example, the memory blocks $\{5, 6, 7, 8, 10\}$ are called *persistent cache blocks* (PCBs) as they are never evicted from the cache when τ_1 executes in isolation. A PCB is therefore a memory block that remains cached during the entire execution of a task unless evicted by another task executing on the same processor. The cache block $\{9\}$ however is called *non-persistent cache block* (nPCB) as it must be reloaded at the beginning of each job execution. nPCBs may be cache blocks that are shared by several memory blocks of the same task, or simply some data that must be reloaded before each job execution of a task. One must note that PCBs and nPCBs are different from the notions of UCBs and ECBs in the sense that it does not matter if they are referenced more than once during a single execution of a task. However, a PCB must never be evicted from the cache by the task itself once it is fetched from the main memory.

The state-of-the-art does not consider PCBs while calculating the memory overhead suffered by a task τ_i in case of preemptions. This results in pessimistic memory overhead evaluations and hence pessimistic WCRT computations. This can easily be shown using the example of Fig. 1. If τ_2's memory overhead is computed using Eq. (7), one would get:

$$MO_2 = MD_2 + 3 \times MD_1 + 3 \times \gamma_{2,1} = 8 + 3 \times 6 + 3 \times 2 = 32$$

Equation (7) considers the worst-case memory demand, i.e., MD_1 for each job of τ_1 that executes during the response time of τ_2. As we have shown in Example 1, the actual memory demand of the second and third job of τ_1 is in fact much less. Considering the PCBs of τ_1 while calculating the memory overhead MO_2, the resulting value is given as:

$$\begin{aligned} MO_2 &= MD_2 + MD_1 + 2 \times (MD_1 - |PCB_1| \times d_{mem}) \\ &\quad + 3 \times \gamma_{2,1} \\ &= 8 + 6 + 2 \times (6 - 5 \times 1) + 3 \times 2 = 22 \end{aligned}$$

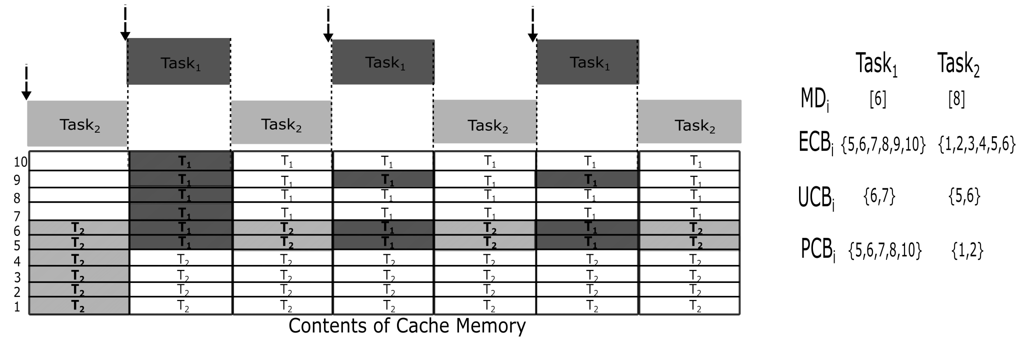
This simple example demonstrates why it is important to account for PCBs when calculating the memory demand and hence the WCRT of a task.

V. WCRT Analysis using memory overhead cost of high Priority Tasks

Two interesting properties can be observed in the example of Section IV:

1) The tasks with a high number of PCBs will have a lower memory demand after the execution of their first job than their worst-case memory demand in isolation. Therefore, we define MD_i^r as the worst-case memory demand over all the jobs of τ_i except the first one.
2) The PCBs of a task τ_j can be evicted due to the execution of lower and high priority tasks (i.e., tasks in $aff(i, j) \cup hp(j)$) between the arrivals of two successive jobs of τ_j. This requires to consider the effect of the tasks in $aff(i, j) \cup hp(j)$ on the memory demand of τ_j during the WCRT of τ_i. This extra memory demand caused by the eviction of the PCBs of τ_j by the tasks in $aff(i, j) \cup hp(j)$ is denoted by $\rho_{j,i}$.

[1] Note that because the same cache block may be used by several memory blocks of the same task τ_i, the worst-case memory demand MD_i of τ_i may be larger than the number of ECBs of τ_i multiplied by d_{mem}.



	Task$_1$	Task$_2$
MD$_i$	[6]	[8]
ECB$_i$	{5,6,7,8,9,10}	{1,2,3,4,5,6}
UCB$_i$	{6,7}	{5,6}
PCB$_i$	{5,6,7,8,10}	{1,2}

Fig. 1. Task schedule and cache content for Example 1.

$\rho_{j,i}$ can be computed using a similar formulation to the ECB-union approach described in Section III (see Eq. (2)). First, we note that every task $\tau_k \in \mathit{aff}(i,j) \cup hp(j)$ can execute between the releases of two successive jobs of τ_j. Second, we consider the fact that each task τ_k may need to load new content in all its ECBs at any time of its execution. Since we are interested in upper bounding the number of PCBs of τ_j that can be evicted by the tasks in $\mathit{aff}(i,j) \cup hp(j)$, we therefore check how many PCBs of τ_j intersect with the ECBs of the tasks in $\mathit{aff}(i,j) \cup hp(j)$. Consequently, the memory overhead $\rho_{j,i}$ is given by:

$$\rho_{j,i} = d_{mem} \times \left| PCB_j \cap \Big(\bigcup_{\forall k \in \mathit{aff}(i,j) \cup hp(j)} ECB_k \Big) \right| \tag{8}$$

Considering the two properties identified at the beginning of this section, we present a more elaborate formulation of the WCRT equation presented in Section III (see Eq. (3)):

$$\begin{aligned} R_i =& P_i + MD_i + \sum_{\forall j \in hp(i)} (P_j + MD_j) + \sum_{\forall j \in hp(i)} \gamma_{i,j}^{mul} \\ &+ \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} - 1 \right\rceil \times \left(P_j + MD_j^r + \rho_{j,i} \right) \end{aligned} \tag{9}$$

In this equation, we separately account for the processing and memory demand of each task, i.e., P_i and MD_i. Similarly, so as incorporate the effect of both MD_i and MD_i^r, we separate the execution of the first job of each preempting task from the execution of their next jobs. While the first job of each task τ_j in $hp(i)$ has a worst-case memory demand MD_j, all the other jobs have a worst-case memory demand $MD_j^r + \rho_{j,i}$, where $\rho_{j,i}$ is calculated using Equation (8). The CRPD $\gamma_{i,j}^{mul}$ is calculated using the multi-set approach given by Eq. (4). Note that in cases where PCBs are also UCBs, $\gamma_{i,j}^{mul}$ and $\rho_{j,i}$ may account twice for the same block evictions. Yet, Eq. (9) improves over the state-of-the-art (i.e., Eq. (3)) as long as C_j is larger than $(P_j + MD_j^r + \rho_{j,i})$.

VI. Conclusion

This work proposes a method to calculate the memory overhead of high priority tasks executing during the response time of a low priority task. In order to bound this overhead, we identified the existence of persistent cache blocks associated with each task. We showed with an example that, due to existence of PCBs, the memory demand of a task can significantly vary over time. We also presented an approach, complementary to [9], to upper bound the number of PCBs of a preempting task that can be evicted by the execution of the preempted tasks. Finally, we reformulated the WCRT analysis so as to consider the effect of the PCBs and the memory demand overhead. In future, we plan to extend our approach to set associative caches. We also aim to present a less pessimistic multi-set approach for memory demand overhead calculation. We further plan to generate results for the proposed WCRT analysis using available benchmarks.

Acknowledgments. This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0003/2012 - JU grant nr. 333053 (CONCERTO) and ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

References

[1] G. C. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. a survey," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 1, pp. 3–15, 2013.

[2] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *RTSS'91*. IEEE, 1991, pp. 129–139.

[3] J. V. Busquets-Mataix, J. J. Serrano, R. Ors, P. Gil, and A. Wellings, "Adding instruction cache effect to schedulability analysis of preemptive real-time systems," in *RTAS'96*. IEEE, 1996, pp. 204–212.

[4] C.-G. Lee, J. Hahn, Y.-M. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, "Analysis of cache-related preemption delay in fixed-priority preemptive scheduling," *Computers, IEEE Transactions on*, vol. 47, no. 6, pp. 700–713, 1998.

[5] H. Tomiyama and N. D. Dutt, "Program path analysis to bound cache-related preemption delay in preemptive real-time systems," in *Proceedings of the eighth international workshop on Hardware/software codesign*. ACM, 2000, pp. 67–71.

[6] J. Staschulat, S. Schliecker, and R. Ernst, "Scheduling analysis of real-time systems with precise modeling of cache related preemption delay," in *ECRTS'05*. IEEE, 2005, pp. 41–48.

[7] Y. Tan and V. Mooney, "Timing analysis for preemptive multitasking real-time systems with caches," *ACM (TECS)*, vol. 6, no. 1, p. 7, 2007.

[8] S. Altmeyer, R. Davis, C. Maiza *et al.*, "Cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," in *RTSS'11*. IEEE, 2011, pp. 261–271.

[9] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

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

[11] S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke, "A generic and compositional framework for multicore response time analysis," in *RTNS'15*. ACM, 2015, pp. 129–138.

An Optimizing Framework for Real-time Scheduling

Sakthivel Manikandan Sundharam, Sebastian Altmeyer, Nicolas Navet
University of Luxembourg
FSTC/Lassy
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg
firstname.lastname@uni.lu

***Abstract*—Scheduling is crucial in real-time applications. For any real-time system, the desired scheduling policy can be selected based on the scheduling problem itself and the underlying system constraints. This paper discusses a novel optimization framework which automates the selection and configuration of the scheduling policy. The objective is to let designer state the permissible timing behavior of the system in a declarative manner. The system synthesis step involving both analysis and optimization then generates a scheduling solution which at runtime is enforced by the execution environment.**

I. Introduction

Real-time scheduling is now a mature and well established research field. Many scheduling policies and results have been proposed and derived over the last four decades providing efficient scheduling solutions for most hardware platforms and application-level needs. Tools and frameworks have been developed implementing these scheduling algorithms and their analyses [1]. To the best of our knowledge, apart from few early works in that direction for specific execution platforms (e.g. [8]), none of these frameworks target the automatic configuration and selection of the best suited policy and parameters in a systematic manner.

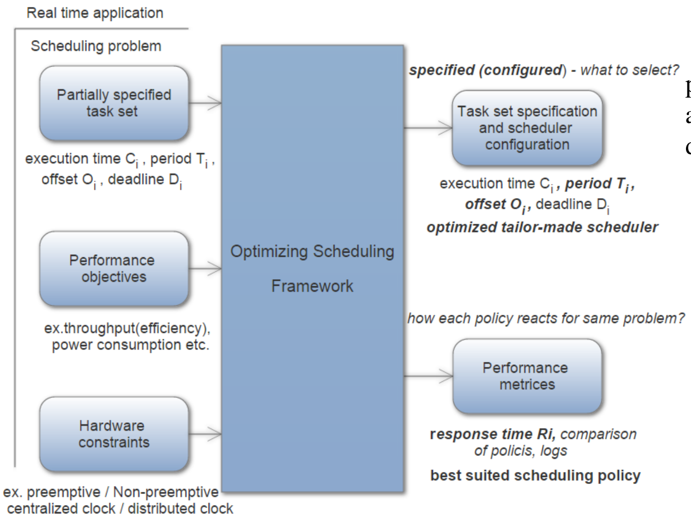

Real time application
Scheduling problem
Partially specified task set
execution time C_i , period T_i , offset O_i , deadline D_i
Performance objectives
ex.throughput(efficiency), power consumption etc.
Hardware constraints
ex. preemptive / Non-preemptive centralized clock / distributed clock
Optimizing Scheduling Framework
specified (configured) - what to select?
Task set specification and scheduler configuration
execution time C_i , period T_i, offset O_i, deadline D_i
optimized tailor-made scheduler
how each policy reacts for same problem?
Performance metrices
response time Ri, comparison of policis, logs
best suited scheduling policy

Fig. 1. Inputs/Outputs of the Framework

In this paper, we present an optimizing framework that selects the best suited scheduling configuration for a partially specified task set and the given system constraints. In Figure 1, we illustrate the structure of the framework. The inputs to the framework are:

- a partially specified task set (see Section II),
- the performance objectives, and
- the hardware constraints.

The framework performs the selection of the policy, optimization of the scheduling parameters, and outputs

- the complete task set specification and scheduling configuration,
- the performance metrics of the different scheduling algorithms.

This work is motivated by the ongoing research on timing-augmented Model-Based Design [7] at the University of Luxembourg. Our aim is to develop the framework such that the system designer only focuses on the high-level timing behavior of the system, where the implementation choices of the low-level timing behavior are taken care of by the framework. The framework fits in the early design phases as a device to automate system synthesis and hide away from the designer the complexity of the underlying runtime environments.

II. Defining The Framework

The inputs is the high-level description of the scheduling problem, whereas the output is the scheduling solution with all the required configuration parameters. In this section, we define inputs and outputs of our optimizing framework.

Partially specified task set:

We assume that the application is composed of n tasks $\{\tau_1, \ldots \tau_n\}$. The task set is defined partially to reflect the freedom in the selection of certain parameters. Each task τ_i is specified by a tuple

$$\tau_i \colon (C_i, T_i, D_i),$$

a) where C_i is the worst-case execution time and is assumed to be given as single value,
b) the execution period T_i is potentially defined as a range of permissible values,
c) the deadline relative to the release time of the task, denoted by D_i, is given as single value. A range of values for the deadline would be futile, as the run-time environment must ensure the system is feasible with respect to the most stringent deadline.

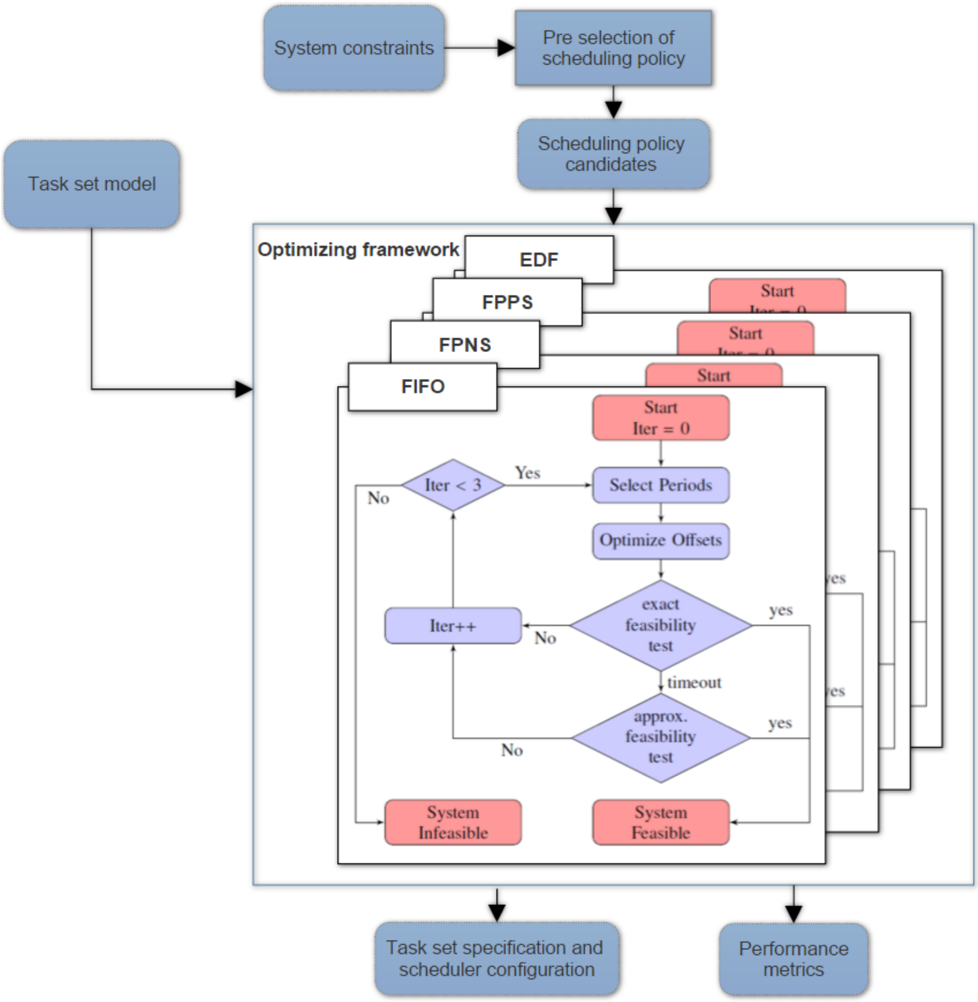


Fig. 2. Flow of the scheduler synthesis framework

Performance objectives:

Examples for performance objectives are throughput (efficiency), power consumption, predictability constraints such as requirement of uninterrupted execution for a task, minimal activation or end-of-execution jitters, etc. These objectives are achieved for example by minimizing the periods within the allowed range in order to reduce the power consumption. If for instance throughput is to be maximized, the frequency of execution is increased.

Hardware constraints:

The selected hardware further constraints the choices of the framework. Typical hardware, and more generally constraints of the run-time environment, are number of processing cores, preemptiveness, type of the system clock (e.g., global clock or distributed clock in the system). These inputs are accounted by the framework in the derivation of the scheduling solution.

Scheduling configuration:

The main outcome of the framework is the scheduling solution, that is the complete specification of the task set and the scheduling configuration. This scheduling configuration covers all low-level configuration parameters and no further input is needed to execute the application on the target system. In addition to the policy, scheduling parameters include periods, offsets and possibly deadlines and priorities.

Performance metrics:

The performance of the candidate algorithm is evaluated by selected performance metrics. Typical performance metrics are schedulability (a task set is schedulable or not under a policy), numerical values of the response times and jitters, power-consumption, or the ability of the system to grow further measured for instance by the minimum slack time.

III. Scheduler Synthesis

In this section, we explain the core of the framework, i.e., the scheduler synthesis. In contrast to other approaches towards scheduler synthesis [3], we do not generate new or non-standard scheduling policies. Instead, we focus on the selection of the most appropriate scheduling policy (including parameter optimization) amongst a set of well-studied and widely-implemented real-time scheduling policies. In Figure II, we illustrate the general steps of our framework.

In a first step, the framework performs a pre-selection of the scheduling policies on the basis of the system constraints and the hardware to execute the application. All policies that violate some of the requirements are excluded at this step. Policies that are compliant with the requirements under some side-constraints are considered with those side-constraints. This step results in the set of candidate policies which are subject to the actual parameter optimization.

The next step, the parameter optimization is then highly specific to the policy, and thus has to be performed for each policy individually. Also, the type of parameters to be optimized differ. However, we can build on a large variety of existing methods and techniques. For the selection of the periods, for instance, we can use a recent work by Nasri et al. [6], offsets in case of offset-aware policies can be optimized using [5] and for the selection of priorities, we have optimality results such as [2].

Real-time scheduling problems are in most contexts NP-hard. Due to the computational complexity of the problems, an optimal scheduling solution cannot be guaranteed. However, the candidate optimization techniques and heuristic algorithms have proven to be robust and lead to satisfactory solutions in many application domains (e.g., [5, 6, 8]).

IV. Illustrating Example

We illustrate our approach using the following task set Γ:

	C_i	T_i	D_i	constraints	objective
τ_1	1	[4 : 5]	4		reduce period
τ_2	2	[4 : 8]	8	non-preemptive	reduce period
τ_3	6	[15 : 24]	24	-	

Constraints and scheduling policies

A side constraint besides meeting deadlines is that task τ_2 has to be executed non-preemptively and the objective is to minimize the values of the periods of τ_1 and τ_2 (*i.e.*, increase frequency to achieve a better control of the system). To simplify the example, we restrain ourselves to a limited number of well-known scheduling policies: earliest deadline first (EDF), both preemptively and non-preemptively (EDF-NP), fixed-priority preemptive scheduling (FPP), fixed-priority non-preemptive scheduling (FPNP), and FIFO:

EDF	EDF-NP	FPP	FPNP	FIFO

The policy selection identifies that all policies can indeed satisfy the system constraints, but in case of the preemptive policies, i.e., EDF and FPP, further constraints are required to ensure the non-preemptive execution of τ_2:

EDF	EDF-NP	FPP	FPNP	FIFO
✓ if $D_2 \leq D_i$	✓	✓ if $pr_2 = 1$	✓	✓

All non-preemptive policies fail since the execution time of τ_3 exceeds even the largest permissible period of τ_1. Hence, the search has to concentrate only on the two remaining policies EDF and FPP (both with the appropriate side constraints).

EDF scheduling

Using EDF we are able to achieve a processor utilization of 1 and execute tasks τ_1 and τ_2 with the smallest possible periods. This is the optimal result and it will be selected as the scheduling solution with the following parameters for the task set:

	C_i	T_i	D_i
τ_1	1	4	4
τ_2	2	4	4
τ_3	5	20	20

For this particular scheduling problem — as well as in many other cases — EDF is the optimal solution. This situation changes when more complex side and system constraints are in place, or when we consider realistic scheduling overheads. The cache-related preemption delays (CRPDs) constitute an example of such overheads that, in this particular case, penalize preemptions. As a result, the advantages of EDF scheduling over FPPS often becomes negligible under CRPD overheads [4], As future work, we plan to include the modeling of these overheads, which will lead to other, less trivial optimal solutions.

V. Conclusions and Discussions

We are developing an optimizing framework that considers as inputs a partially specified task set, the performance objectives of the system and the constraints of the run-time environment, importantly the hardware support. The framework synthesizes the scheduling solution that best meet the requirements. Our framework currently includes a number of basic real-time scheduling policies and ongoing work is devoted to enrich the set of available policies with customized scheduler that make optimal use of underlying execution hardware and improvements in system behavior. This work is a contribution towards a more automated design process building on the wide set of techniques and results developed within the real-time system community.

The applicability and precision of the framework is determined by the optimization algorithms and schedulability analyses. But these algorithms and heuristic techniques are limited in precision. Consequently, for some scheduling problems, the framework cannot guarantee optimality. A future work is to develop techniques such as lower bounds to estimate how far is a solution computed with the framework from the optimal solution.

Acknowledgment

This research is supported by FNR (Fonds National de la Recherche), the Luxembourg National Research Fund (AFR Grant n°10053122).

References

[1] K. Altisen, G. Gossler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS)*, Dec. 1999.

[2] N. C. Audsley. On priority asignment in fixed priority scheduling. *Inf. Process. Lett.*, 79(1):39–44, May 2001.

[3] M. Grenier and N. Navet. Fine tuning MAC level protocols for optimized real-time QoS. *IEEE Transactions on Industrial Informatics, special issue on Industrial Communication Systems*, 4(1), 2008.

[4] W. Lunniss, S. Altmeyer, and R. I. Davis. A comparison between fixed priority and edf scheduling accounting for cache related pre-emption delays. *Leibniz Transactions on Embedded Systems*, 1(1), 2014.

[5] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. Multisource software on multicore automotive ecus combining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942, Oct 2012.

[6] M. Nasri and G. Fohler. An efficient method for assigning harmonic periods to hard real-time tasks with period ranges. In *27th Euromicro Conference on Real-Time Systems (ECRTS 2015)*, pages 149–159. IEEE, 2015.

[7] N. Navet, L. Fejoz, L. Havet, and S. Altmeyer. Lean model-driven development through model-interpretation: the CPAL design flow. In *Embedded Real-Time Software and Systems (ERTSS2016)*, January 2016.

[8] N. Navet and J. Migge. Fine tuning the scheduling of tasks through a genetic algorithm: Application to Posix1003.1b compliant OS. *IEE Proceedings Software*, 150(1):13–24, 2003.

Preliminary Performance Evaluation of HEF Scheduling Algorithm

Carlos A. Rincón [†*] and Albert M. K. Cheng[*]
[†]Networking and Telematics Academic Unit, Universidad del Zulia, Maracaibo, Venezuela. Email: crincon@fec.luz.edu.ve
[*]Real-time Systems Laboratory, University of Houston, Houston, USA. Email: cheng@cs.uh.edu

***Abstract*—The purpose of this paper is to analyze the performance of the Highest Entropy First (HEF) scheduling algorithm for real-time tasks. We generate multiple task sets using the Seoul National University (SNU) real-time benchmark. The tasks were implemented on WindRiver Workbench 3.3 to estimate the WCET. A linear programming solution was implemented to set the period of the tasks aiming to maximize the utilization of the system based on a predefined hyper-period. We measure the performance of HEF scheduling algorithm using as parameters the number of context switches and the deadline-miss ratio. As preliminary result we show that the number of context switches is directly proportional to the number of tasks in a task set. The deadline-miss ratio for all the studied cases was 0%, because the utilization for all the task sets was at most 1 ($U \leq 1$).**

Keywords—Highest Entropy First; real-time systems; scheduling; performance.

I. Introduction

In recent years the use of entropy as a parameter has been proposed as a new approach to schedule real-time tasks [1]. In 2015, Rincon and Cheng [2] presented the mathematical background to measure the entropy of a task set in a real-time system as well as the design, feasibility analysis and implementation of the highest entropy first (HEF) algorithm to schedule real-time tasks in uni-processors.

The HEF algorithm is a new dynamic priority technique to schedule real-time tasks that tries to minimize the uncertainty (based on the probability of the execution of a task during the hyper-period) of the scheduling problem by executing the task with the highest entropy first without missing any deadline.

The purpose of the research is to measure the performance of the highest entropy first scheduling algorithm in order to have a guideline about the behavior of the studied algorithm under certain conditions.

The contributions of this paper are:

- Generate multiple task sets by implementing the programs from the SNU real-time benchmark [3] in Wind River Workbench 3.3 [4] to calculate the WCET and generating the periods by using a linear programming solution aiming to maximize the utilization of the system based on a predefined hyper-period.
- Measure the performance of HEF algorithm to schedule real-time tasks using as metrics the number of context switches and deadline-miss ratio.

The rest of the paper is organized as follows. In the next section, we describe the related work about using entropy as a parameter to schedule real-time tasks. In section 3, we present the design of the HEF scheduling algorithm. Section 4 presents the methodology used to generate the task set for the performance evaluation. Section 5 presents the performance evaluation of the HEF algorithm running the generated task set. We give our conclusions and future work in section 6.

II. Related Work

A. Entropy as a Parameter for Real-time Scheduling

Entropy is defined as the product of the information generated by an event x and the probability of occurrence of that event (p_x*I_x) [5]. Considering a periodic task system with implicit deadlines where the worst case execution time = C_i, period = T_i, hyper-period ($hperiod$) = least common multiple of the periods and applying the information-theoretic concepts, we define the following parameters:

Entropy of a Single Time Unit from a Scheduling Diagram H_{SU}: we define the information generated by a single time unit of the scheduling diagram (I_s) as $log_2(1/P_s)$, where P_s is the probability of a single time unit = $1/hperiod$. The entropy of a single time unit = $P_s * I_s = log_2(hperiod)/hperiod$ bits.

Total Entropy of a Task H_{Task}: is defined as the product of the number of single time units on the scheduling diagram used by a task (number of task instances times C_i) and the entropy of a single time unit (H_{SU}).

$$H_{Task} = \frac{hperiod}{T_i} * C_i * H_{SU} = log_2(hperiod) * \frac{C_i}{T_i} bits \quad (1)$$

Total Normalized Entropy of a Task NH_{Task}: is defined as the total entropy of a task (H_{Task}) divided by its computation time (C_i). This parameter is critical for using entropy in real-time systems because it prioritizes the scheduling based on the task deadlines.

Total Entropy of the System H_{Sys}: is the summation of the entropies of all m tasks (with m=number of tasks).

$$H_{Sys} = \sum_{i=1}^{m} H_{Task_i} = log_2(hperiod) * \sum_{i=1}^{m} \frac{C_i}{T_i} bits \quad (2)$$

Relationship between H_{Sys} and Utilization: Based on Shannon's information theory [5], we know that the maximum value of the entropy (H_x) = log_2(number of possible cases=$hperiod$). Then $H_{Sys} \leq log_2(hperiod)$. Based on this inequality, we have:

$$log_2(hperiod) * \sum_{i=1}^{m} \frac{C_i}{T_i} \leq log_2(hperiod) \quad (3)$$

This inequality is true only if $U = \sum_{i=1}^{m} C_i/T_i \leq 1$.

[*]Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

III. The Highest Entropy First scheduling algorithm

The studied algorithm is a dynamic priority scheduler that uses the normalized entropy of the task (NH_{Task}) and the total entropy of the task (H_{Task}) to decide which task to run first. Basically, every time the scheduler needs to decide which task to run, it will choose the task with the highest entropy (normalized and total), in order to minimize the complexity of the scheduling problem.

A. *Algorithm's Design*

The proposed scheduling algorithm based on entropy has the following steps:

1) Determine the schedulability of the given task set using the relationship between entropy and utilization proposed in equation (3).
2) Calculate the normalized and total entropy for each task.
3) Select the task to be executed using the following criteria:
   - Select the task with the highest normalized entropy.
   - If two or more tasks have the highest normalized entropy, then select the task with the highest total remaining entropy.
   - If two or more tasks have the highest total remaining entropy and one of these tasks is the one running, then select the task that is running (to minimize preemption), else select the task based on its process identifier (PID).
4) Update the values of T_i and C_i for all tasks.
5) Go to step 2 until time = $hperiod$.

IV. Generating the Task sets

In order to generate the tasks sets to measure the performance of the HEF scheduling algorithm we selected 5 programs from the SNU Real-time benchmark (sqrt.c, fibcall.c, crc.c, minver.c and select.c). Table I shows a description of each selected program.

TABLE I: Selected programs from the SNU real-time benchmark

Task Number	SNU Program	Description
1	sqrt.c	Square root,function implemented by Taylor series
2	fibcall.c	Summing the Fibonacci series
3	crc.c	A demonstration for CRC (Cyclic Redundancy Check) operation
4	minver.c	Matrix inversion for 3x3 floating point matrix
5	select.c	A function to select the Nth largest number in the floating point array size 20

After selecting the programs we implemented them on a server with an Intel i7-3770 processor running at 3.4 GHz, with 16 GB of RAM and 2 TB hard drive using Wind River Workbench 3.3 to calculate the worst case execution time (WCET). We run each program 100 times to average the results. Table II shows the average WCET for the selected programs.

A. *Task sets*

To measure the performance of the HEF scheduling algorithm we decided to use as independent variable the number of tasks in the task set. We created 4 task sets with 2, 3, 4, and 5 tasks respectively. For each task set we use 100 ms as the hyperperiod and applied a linear programming solution to calculate the periods for each task (aiming to maximize the utilization of the system). For the deadlines, we implemented a system with implicit deadlines. Tables III and IV show the generated task sets.

TABLE II: WCET for the selected tasks

Task Number	WCET	ROUND WCET
1	13.34 ms	14 ms
2	7.32 ms	8 ms
3	13.54 ms	14 ms
4	16.41 ms	17 ms
5	25.73 ms	26 ms

TABLE III: Task sets 1 and 2

Task Number	C_i	T_i
1	14 ms	50 ms
2	8 ms	12 ms

Task Number	C_i	T_i
1	14 ms	100 ms
2	8 ms	50 ms
3	14 ms	20 ms

TABLE IV: Task sets 3 and 4

Task Number	C_i	T_i
1	14 ms	100 ms
2	8 ms	34 ms
3	14 ms	50 ms
4	17 ms	50 ms

Task Number	C_i	T_i
1	14 ms	100 ms
2	8 ms	100 ms
3	14 ms	100 ms
4	17 ms	50 ms
5	26 ms	100 ms

The utilization of the generated task sets are: Task set 1 = 0.946666667, Task set 2 = 1, Task set 3 = 0.995294118 and Task set 4 =0.96.

V. Preliminary Performance Evaluation of HEF

With the generated task sets, we run the HEF scheduling algorithm to measure its performance. We calculated the number of context switches and the deadline-miss ratio for each task set. The results are shown in figure 1.

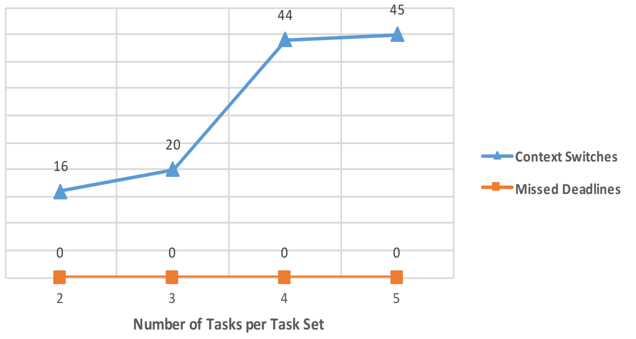


Fig. 1: Number of context switches and deadline-miss ratio for HEF

For task set 1 (2 tasks) the number of context switches is 16 and the deadline-miss ratio is 0%, for task set 2 (3 tasks) the number of context switches is 20 and the deadline-miss ratio is 0%, for task set 3 (4 tasks), the number of context switches is 44 and the deadline-miss ratio is 0% and for task set 4 (5 tasks), the number of context switches is 45 and the deadline-miss ratio is 0%.

These results show that when the number of tasks in the task set increases, the number of context switches increases. The obtained deadline-miss ratio (0 missed deadlines) for all the tasks sets is a consequence of the utilization values (less or equal than 1 for all task sets).

VI. Conclusions and Future Work

In this paper, we have proposed a methodology to generate task sets using the programs from the SNU real-time benchmark (implemented on Wind River Workbench). We calculated the periods for the tasks in the task set using a linear programming solution to maximize the utilization of the system.

The results from the preliminary performance evaluation of the HEF scheduling algorithm show that the number of context switches is directly proportional to the number of tasks in the task set. For the deadline-miss ratio, further analysis must be made to confirm that it depends on the utilization of the system ($U \leq 1$ = no deadline misses).

The HEF algorithm has some similarities with the earliest deadline first algorithm [6] (EDF) because selecting the task with the lowest absolute deadline is the same as selecting the task with highest normalized entropy ($NH_{Task} = log_2(hperiod) * \frac{1}{T_i}$). However when two or more tasks have the same absolute deadline, HEF will select the task that adds more complexity to the scheduling problem using as a parameter the total remaining entropy of the task. Therefore we propose as future work to compare the performance of HEF against EDF using the task sets generated by the methodology proposed in this paper.

References

[1] R. Sharma and Nitin, “Entropy, a new dynamics governing parameter in real time distributed system: a simulation study,” *IJPEDS*, vol. 29, no. 6, pp. 562–586, 2014.

[2] C. A. Rincon and A. M. Cheng, “Using entropy as a parameter to schedule real-time tasks,” in *Real-Time Systems Symposium. WiP Session, 2015 IEEE*, Dec 2015, pp. 375–375.

[3] “Snu real-time benchmark suite,” http://archi.snu.ac.kr/realtime/benchmark.

[4] WindRiver, “Wind river workbench,” http://www.windriver.com.

[5] C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[6] C. Liu and J. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

Using Linked List in Exact Schedulability Tests for Fixed Priority Scheduling

Jiaming Lv*, Xingliang Zou§, Albert M. K. Cheng§, and Yu Jiang+*
*School of Computer Science and Technology, Heilongjiang University, Harbin, Heilongjiang 150080, China
Email: 962831141@qq.com, jiangyu@hlju.edu.cn
§Department of Computer Science, University of Houston, Houston, TX 77004, USA
Email: xzou@uh.edu, cheng@cs.uh.edu

Abstract—**Efficient exact schedulability tests are one of important considerations of both research motivation and practice stage. In this paper, we investigate the exact response-time schedulability tests for fixed priority preemptive systems. The linked list is introduced to represent the simulated schedule of a given task set. Each node in the linked list represents a busy period. In addition, the memory space needed for the linked list is managed in the user space. Experiments show that the linked list-based exact test outperforms the current best exact response-time test and the hyperplanes exact tests (HET) in the case of task periods spanning no more than three orders of magnitude.**

I. Introduction

Real-time systems are playing a crucial role in our daily lives and in industry production, and fixed priority preemptive scheduling is widely supported by most commercial real-time operating systems [6].

In the context of fixed priority preemptive real-time systems, it is known that for periodic/sporadic tasks that comply with a restrictive system model and that have implicit deadlines the *Rate-Monotonic* (RM) scheduling is optimal, i.e., if a feasible scheduling exists for some task set then the RM scheduling is feasible for that task set [11], [14]. RM means that the priority of each task is assigned inversely proportional to its period (i.e., minimum inter-arrival time between jobs of the task). It is also known that when these tasks are released simultaneously (i.e., sharing a common release time) the time required by the first job of each task defines its response time [11], [14]. Therefore, it needs only to make response time analysis or conduct exact schedulability test within a time length no more than the maximum task period, and these tests are thus known to be pseudo-polynomial in time complexity [8], [9], [1].

Although the response time computation for RM schedules of implicit-deadline task-systems has been proved to be an NP-hard problem [7], the scale of many commercial systems is such that pseudo-polynomial exact tests can be used, and to achieve more efficient exact tests for use such as online response time analysis (RTA) is one of important considerations of both research motivation and practice stage. A significant research effort has been dedicated to improve the performance of exact response-time tests [8], [1], [12], [2], [6], [4], [13], such as finding good initial values, and to the best of our knowledge the authors of [6] presented the current best response-time test with better initial values.

In this paper, we investigate exact response-time schedulability tests of the RM scheduling in an n-task real-time system. For concision we use the *Burns Standard Notation* [5], such as the number of tasks n, for $1 \leq i \leq n$, the priority P_i, the worst-case execution time C_i, the relative deadline D_i, the period T_i, the worst-case response time R_i, and the utilization U_i, for a task τ_i.

The innovative aspect of our solution is that we use a linked list for representing the schedule in the exact response-time test, referred to as the *LList-based exact test*, for calculating the worst-case response time. The time complexity of the LList-based exact test is polynomial-time $O(N)$ where N is the total number of jobs within the time length T_n, while the total number of nodes used in the linked list is no more than $N - n + 1$ in the worst case. Our experiments show that the LList-based exact test outperforms the current best exact RTA test [6] and the hyperplanes exact tests (HET) [2] in the case of task periods spanning no more than three orders of magnitude, and the needed memory space is also affordable.

II. Our Method

We calculate the response time of each task set by simulating its schedule within a time length T_n. In our method, the schedule of a task set is represented by a linked list, and a *busy period* [10] in the schedule is represented by a linked list node. Each list node has three fields: the starting time of the busy period, the end time of the busy period, and the pointer to the next node. The simulation is performed task per task in the priority order, from 1 to n, and, when the starting time or the end time of a *priority level-i busy period* is the same as that of a *priority level-j busy period* where $j < i$, then the two nodes are merged into one node to represent a longer busy period.

Another key factor for improving the efficiency of the LList-based exact test is that before the simulation a memory array is allocated as a whole and then the following operations of memory allocation and recycle for each node are performed in the user space instead of in the operating system space.

This work is sponsored in part by the State Scholarship Fund of China under award No. 201308230034, the US National Science Foundation under award Nos. 0720856 and 1219082, the Funds of Heilongjiang Education Office of China under award Nos. 12541627 and GJZ201301027, and the Heilongjiang Student's Platform for Innovation and Entrepreneurship Training Program of China under award No. 2015102121008.

[+] Corresponding author.

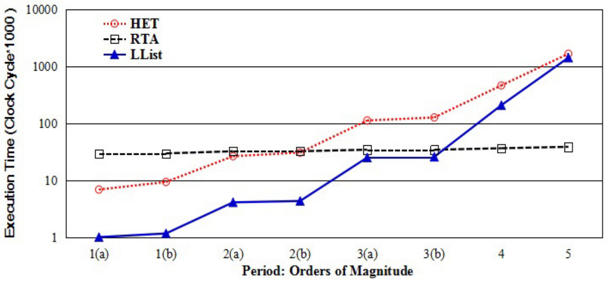


Fig. 1. Average execution time required by the HET, RTA, and LList-based exact schedulability tests versus number of orders of magnitude range of task periods. The range of periods starts from 10 and 10^4 for (a) and (b), respectively. Note both x and y-axes are logarithmic scales.

TABLE I
HET, RTA, AND LLIST-BASED ALGORITHMS, EXECUTION TIME IN COUNTER CLOCK CYCLES × 1,000

	Orders of magnitude spanning tasks periods						
Algorithm	1(a)	1(b)	2(a)	2(b)	3(a)	3(b)	4
HET	6.9	9.4	26.6	30.8	112.8	127.4	464.9
RTA	29.56	29.82	32.47	32.48	34.63	34.49	36.67
LList	1.01	1.17	4.12	4.35	25.19	25.94	208.9
RTA/LList	29.3	25.5	7.9	7.5	1.4	1.3	0.2

For an n-task set, the total number of jobs, N, in the time interval [0, T_n) is $N = \sum_{i=1}^{n}(T_n/T_i)$ and only these jobs are simulated in the LList-based exact test. This number is sensitive to the span and the distribution of task periods. Since one busy period in the schedule is represented with merely one node and the simulation is performed task per task in the task priority order, the total running time is mainly determined by the time of operating linked list nodes, and thus the time complexity of the LList-based exact test is related to the number of nodes used in the simulation. Particularly, as long as task periods span only one order of magnitude, only $O(n)$ time are needed for simulating an n-task set, regardless of the length of the maximum period of the task set as well as the total utilization. In the above mentioned case, the total number of list nodes used in simulation is very small, and this is the root cause why the LList-based test is performing better.

III. Experiment and Preliminary Results

For comparison, we use the same parameters as those in [6], the current best exact RTA test. Specifically, for each 24-task set $24/M$ tasks were assigned to each of the M order of magnitude ranges (e.g., 100-1,000, 1,000-10,000, 10,000-100,000, etc.). Task periods were then uniformly and randomly generated from the assigned range. The overall utilization was fixed at 0.85 and the UUniFast algorithm [3] was used to determine task utilizations U_i, and, hence, task execution times, $C_i = U_i * T_i$. There are 10,000 task sets in each order of magnitude ranging from 1 to 5.

Fig. 1 and Table I show how the average number of clock cycles required by the HET algorithm with initial values $R_{i-1} + C_i$, by the RTA test with initial values $max_{k=1}^{i}(R_i^{LB}(k))$ [6], and by the LList-based test varied with the number of orders of magnitude spanning task periods. From the figure we can see that the execution time of the RTA test goes nearly steady while the HET and the LList-based tests increase exponentially with increasing number of orders of magnitude spanning task periods. Within three orders of magnitude spanning task periods, however, the performance of the LList-based test outperforms both the HET test and the RTA test with the current best initial values.

TABLE II
THE MAXIMUM NUMBER OF NODES AND CORRESPONDING MEMORY SPACE

	Orders of magnitude spanning tasks periods						
	1(a)	1(b)	2(a)	2(b)	3(a)	3(b)	4
max # nodes	28	58	236	441	2302	3246	20996
KB(int32)	0.3	0.7	2.8	5.2	27.0	38.0	246.0

Table II shows the maximum number of list nodes used in the LList-based test versus number of orders of magnitude range of task periods. From the table we can see that, within three orders of magnitude spanning task periods, the memory space needed by the linked list is completely affordable.

IV. Conclusion

The work presented in this paper is part of our ongoing research on the response time analysis and exact scheduability test for fixed priority preemptive systems. Our preliminary results have shown that the LList-based exact test is a better candidate in exact response-time tests when task periods span no more than three orders of magnitude. More comprehensive experiments will be conducted to investigate the suitable scope of the LList-based exact test by varying the number of tasks, the range of task periods, and the total utilizations.

References

[1] N. C. Audsley, A. Burns, M. Richardson, K. W. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[2] E. Bini and G. C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. on Computers*, 53(11):1462–1473, 2004.

[3] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Syst.*, 30(1-2):129–154, 2005.

[4] R. I. Davis. A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Review*, 11(1):8–19, 2014.

[5] R. I. Davis and A. Burns. Burns standard notation for real-time scheduling. In *Real-Time Systems: The past, the present, and the future. N. Audsley, S.K. Baruah Editors*, pages 38–41, Mar. 2013.

[6] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Trans. on Computers*, 57(9):1261–1276, 2008.

[7] F. Eisenbrand and T. Rothvoss. Static-priority real-time scheduling: Response time computation is NP-hard. In *IEEE RTSS 2008*, pages 397–406, 2008.

[8] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer J.*, 29(5):390–395, 1986.

[9] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE RTSS 1989*, pages 166–171, 1989.

[10] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE RTSS 1990*, pages 201–209, 1990.

[11] C. Liu and L. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[12] W. C. Lu, K. J. Lin, H. W. Wei, and W. K. Shih. Period-dependent initial values for exact schedulability test of rate monotonic systems. In *Proc. IPDPS 2007*, pages 1–8.

[13] M. Park and H. Park. An efficient test method for rate monotonic schedulability. *IEEE Trans. on Computers*, 63(5):1309–1315, 2014.

[14] O. Serlin. Scheduling of time critical processes. In *Proc. AFIPS Spring Computing Conf.*, pages 925–932, 1972.

Online Semi-Partitioned Multiprocessor Scheduling of Soft Real-Time Periodic Tasks for QoS Optimization

Behnaz Sanati, Albert M. K. Cheng
Department of Computer Science, University of Houston, Texas, USA
Emails: {bsanati; acheng}@cs.uh.edu

***Abstract*—*In this paper, we propose a novel semi-partitioning approach with an online choice of two approximation algorithms, Greedy and Load-Balancing, to schedule periodic soft real-time tasks in homogeneous multiprocessor systems. Our objective is to enhance the QoS by minimizing the deadline misses and maximizing the total reward or benefit obtained by completed tasks in minimum response time. Many real-time applications and embedded systems can benefit from this solution including but not limited to video streaming servers, multi-player video games, cloud applications, medical monitoring systems, and IoT.*

Keywords: *Periodic tasks, Quality of service, Partitioning, Multiprocessor scheduling, Approximation algorithms.*

1. INTRODUCTION

Multiprocessor systems are widely used in a fast-growing number of real-time applications and embedded systems. Two examples of such systems are Cloud applications [1] and IoT [2]. In hard real-time systems, meeting all deadlines is critical, while in soft real-time systems, missing few deadlines does not drastically affect the system performance. However, it would compromise the quality of the service.

In such systems, jobs meeting their deadlines will gain a reward (also called benefit). Hence, researchers focus on maximizing rewards to improve the QoS. Besides the total reward, other factors also influence QoS, such as overall response time (makespan plus scheduling time) and deadline-miss ratio. Multiprocessor real-time scheduling algorithms may follow a partitioned or global approach or some hybrid of the two, called semi-partitioning.

Global scheduling can have higher overhead in at least two respects: the contention delay and the synchronization overhead for a single dispatching queue is higher than for per-processor queues; the cost of resuming a task may be higher on a different processor than on the processor where it last executed, due to inter-processor interrupt handling and cache reloading. The latter cost can be quite variable, since it depends on the actual portion of a task's memory that remains in cache when the task resumes execution, and how much of that remnant will be referenced again before it is overwritten [1]. These issues are discussed at some length by Srinivasan *et al.* [3]. Elnably *et al.* [1] study fair resource allocation and propose a reward-based model for QoS in Cloud applications. In contrast, Alhussian, Zakaria and Hussin [4] prefer global scheduling and try to improve real-time multiprocessor scheduling algorithms by relaxing the fairness and reducing preemptions and migrations.

Amirijoo, Hansson and Son [5] discussed specification and management of QoS in real-time databases supporting imprecise computations. Reward-based scheduling of periodic tasks has also been studied by Aydin *et al.* [6], and Hou and Kumar [7]. Aweruck *et al.* [8] proposed a reward-maximizing model for scheduling aperiodic tasks on uniprocessor systems which can also be applied to multiprocessors. We have also previously studied reward-based scheduling of aperiodic real-time tasks on multi-processor systems. We proposed two algorithms, GBBA [9] and LBBA [10], and provided performance analysis and comparative experimental results of those algorithms versus another state-of-the art algorithm [8].

Significant improvements obtained by LBBA method, especially in reducing the overall response time (i.e., scheduling time plus makespan of the task sets), in addition to maximizing the total reward and minimizing tardiness, showed promising enhancement in QoS. That encouraged us to expand our research to solving the problem of scheduling periodic (and sporadic) soft real-time tasks on multi-processor systems, on which relatively very little research has been done. LBBA is using partitioning strategy for aperiodic tasks. Now to extend it for scheduling periodic (and sporadic) tasks, we use semi-partitioning at job boundaries.

Semi-partitioned real-time scheduling algorithms extend partitioned ones by allowing a subset of tasks to migrate. Given the goal of "less overhead," it is desirable for such strategy to be boundary-limited, and allow a migrating task to migrate only between successive invocations (job boundaries). Non-boundary-limited schedulers allow jobs to migrate, which can be expensive in practice, if jobs maintain much cached state.

Previously proposed semi-partitioned algorithms for soft real-time (SRT) tasks such as EDF-fm and EDF-os [11], have two phases: an offline assignment phase, where tasks are assigned to processors and fixed tasks (which do not migrate) are distinguished from migrating ones; and an online execution phase. In their execution phase, rules that extend EDF scheduling are used. The goal in these strategies is to minimize tardiness.

In this paper, we propose a new online reward-based semi-partitioning approach to schedule periodic soft real-time tasks in homogeneous multiprocessor systems. We use an online choice of two approximation algorithms (Greedy approximation and Load-Balancing) for partitioning, which provides an optimized usage of processing time. In this method, no prior information is needed. Hence, there is no offline phase.

Our objective is to enhance the QoS by minimizing tardiness and maximizing the total reward obtained by completed tasks in minimum makespan. Therefore, we allow different jobs of any task get assigned to different processors (migration at job boundaries) based on their reward-based priorities and workload of the processors. This method can also direct SRT systems with mixed set of tasks (aperiodic, sporadic and periodic) by defining their deadlines accordingly.

Many real-time applications can benefit from this solution including but not limited to video streaming servers, multi-player video games, mobile online banking and medical monitoring systems. For example, consider mobile banking applications that are set to send monthly statements, weekly or daily balance notific-ations (periodic) and also notifications when a check is posted or the balance is less than specific amount (aperiodic).

Another example is a medical monitoring application installed on a physician´s laptop or smart phone which periodically receives the patients´ vital signs, such as blood pressure, number of heart beat, breathing per minute, etc. from the body sensor networks attached to the patients. It process and records them periodically and in case they go out of range and the situation is critical, sends alert (aperiodic). In the next sections, we explain our novel semi-partitioning hybrid model, which combines reward and cost models, for optimizing quality of service in soft real-time systems.

2. OUR CONTRIBUTION

2.1. System and Task Model

A multiprocessor system with *m* identical processors is considered for partitioned, preemptive scheduling of periodic soft real-time task sets with implicit deadline. Each processor has its own pool (for ready tasks), stack (for preempted and running tasks) and garbage collection (for completed and tasks which missed deadlines). Each periodic task may be released at any time. Tasks are independent in execution and there are no precedence constraints among them. Pre-emption is allowed. A desired property of the system in this method is the possibility to delay jobs without drastically reducing the overall system performance.

2.2. Our Methodology

Semi-Partitioning Model:

This algorithm applies online semi-partitioning. In our partitioning approach, no job migration is allowed. In other words, each job, i.e. an instance of a task, will be assigned to a processor at release time, based on its priority and worst-case execution time, and also the current workloads of the processors, and it has to stay with that processor during its entire runtime in the system. However, different instances of a periodic task may be assigned to different processors. This method is possible since each processor has its own pool for the ready tasks assigned to it.

Online Choice of Approximation Algorithms:

We consider Greedy and Load-balancing approximation algorithms, one of which will be chosen online based on the conditions of the system at each time instance, for partitioning and scheduling task instances in order to optimize the CPU usage, minimize the makespan and prevent starvation of low priority tasks. We explain it in more details in subsection 2.4.

2.3. Definitions

Periodic Tasks:

A *periodic* task, in real-time systems, is a task that is periodically released at a constant rate. Usually, two parameters are used to describe a periodic task T_i; its execution w_i as well as its period p_i. An instance of a periodic task (*i.e* release) is known as a job and is denoted as $T_{i,j}$, where j=1, 2, 3, … . The deadline of a job is the arrival time of its successor. For example, the deadline of the j^{th} job of T_i, which is $T_{i,\,j}$, would be the arrival time of job $T_{i,(j+1)}$, that is at jp_i.

Notations:

We define the notations used throughout this paper as follows:

$r_{i,j}$ – *release time* of job $T_{i,j}$

w_i – *execution time* of job $T_{i,j}$, simply considered as *workload* of job $T_{i,j}$ in this paper

p_i – *period of task* T_i

$s_{i,j}$ – *start time* of job $T_{i,j}$

$c_{i,j}$ – *completion time* of job $T_{i,,j}$

$Br_{i,j}$ – *break point* or *deadline* of job $T_{i,,j}$, is the minimum of: $Br_{i,j}$ = min ($p_i \,\|\, s_{i,j}$ +2w_i) (1)

$\beta_i(t)$ – *benefit density function* of task T_i at time *t*, for ($t \geq w_i$), which is a non-increasing, non-negative function, with the following restriction to be satisfied for each $\beta_i(t)$:

$$\frac{\beta_{i\,(t)}}{\beta_i(t+w_i\,)} \leq C \qquad (2)$$

Note: for $t < w_i$, there would be no benefit gained by job $T_{i,j}$, since it has certainly not completed its execution at time *t*.

$f_{i,j}$ – *flow time* of job $T_{i,j}$:

$$f_{i,j=}\, c_{i,j} - r_{i,j} \qquad (3)$$

$b_{i,j}$ – *benefit*, gained by a completed job $T_{i,j}$:

$$b_{i,j\,=}\, w_i.\ \beta_i\,(f_{i,j}) \qquad (4)$$

LBBA Algorithm for Periodic Tasks

```
Required: One or more jobs arrive at time $t \geq 0$
{
```

Job Arrival

```
    /* TempList: list of ready jobs waiting for
       distribution among processors */

    Append the arrived job(s) to the TempList
```

Benefit-Based Scheduling

```
    Calculate the priority of each job $T_{i,j}$ in the
      TempList:
        $d_{i,j}(t) = \beta_i(t + w_i - r_{i,j})$
    Sort TempList based on the priority
    If (at least one stack is empty)
    {
            Push the highest priority job(s) $T_{i,j}$
              onto empty stack(s) of idle processor(s) $l$;
            Add its execution time $w_i$ to total workload
              of the stack of the processor $l$ ($\sum Ws_l$),
            Recalculate total workload of processor $l$:
              $W_l = \sum Wp_l + \sum Ws_l$
            Calculate the fixed priority of $j$ using its
              start time $s_{i,j}$:
              $d'_{i,j}(t) = \beta_i(s_{i,j} + w_i - r_{i,j})$
            Start executing $j$,
    }
    Else
    {
        /* no stack is empty */
        /* preempt if possible otherwise distribute
           among the pools */
        Compare the priority of the ready jobs in
            TempList with the priority of the running
            jobs (indicated by index $k$) on top of the
          stacks:
         If ($d_{i,j}(t) \leq 4d'_k$ for ( each job $T_{i,j}$ in TempList
            and each running job $T_k$) )
         {
              /* no preemption allowed */
              /* partition the ready jobs among
                 pools of the processors */
```

Load-Balancing Approximation (for Partitioning)

```
                For (each job $T_{i,j}$ in TempList)
                {
                  Sort the processors in ascending order of
                      their total remaining workload on their
                      pools and stacks:
                      $W_l = \sum Wp_l + \sum Ws_l$
                  Append the job$T_{i,j}$ with largest
                      execution time $w_i$ to the pool of the
                      processor $l$ with minimum remaining
                      work load;   /* load balancing */
                  Remove $T_{i,j}$ from TempList;
                  Add its execution time $w_i$ to total
                    workload of the pool of processor $l$
                    ($\sum Wp_l$);
                  Recalculate total workload of
                    processor $l$:
                    $W_l = \sum Wp_l + \sum Ws_l$
                }
         }
        Else
             /* if ($d_{i,j}(t) > 4d'_k$)  then ($T_{i,j}$ preempts $T_k$)*/
```

Greedy Approximation (multiple-choice Preemption)

```
             /* If $T_{i,j}$ has more than one choice of
                processors, it will be pushed onto
                the stack whose processor has the
                least work load (greedy) */
        {
             Stop the execution of job $k$ (preempt $k$),
             Push the job $T_{i,j}$ onto the stack on top of $T_k$,
             Start executing $T_{i,j}$,
             Calculate the fixed priority of $T_{i,j}$ using its
               Start time $s_{i,j}$: $d'_{i,j}(t) = \beta_i(s_{i,j} + w_i - r_{i,j})$
             Add the execution time of $T_{i,j}$ to the total
               workload of that stack ($\sum Ws_l$),
             Recalculate total workload of the
                 Processor $l$:
                 $W_l = \sum Wp_l + \sum Ws_l$
        }
```

Check for missed Deadlines

```
        /* at each time instance $t$, if any of the running jobs
           on top of the stacks has reached its break point:
               ($t > Br_{i,j}$),  $Br_{i,j} = \min(p_i \| s_{i,j} + 2w_i)$
            remove the job from the stack and send
            it to the processor Garbage Collection
            otherwise, if not preempted, continue its
            execution */
```

Benefit Gained by Completed Jobs

```
         /* for every completed job $T_{i,j}$ calculate $b_{i,j}$ */
        $b_{i,j} = w_i . \beta_i( f_{i,j} )$
        }
```

Total Benefit Calculation

```
        /* calculate the sum of all benefits gained,
           $B$ is initially set to zero*/
        $B = B + b_{i,j}$
}
```

$d_{i,j}$ (t) – *variable priority* of job $T_{i,j}$ at time t, before scheduling ($t < s_{i,j}$): $d_{i,j} (t) = \beta_i (t + w_i - r_{i,j})$ (5)
$d'_{i,j}$ – *fixed priority* of job $T_{i,j},$ when it is scheduled and starts running: $d'_{i,j} = \beta_i (s_{i,j} + w_i - r_{i,j})$ (6)

2.4. Our Algorithm

In this system, the events are new job arrival, job completion, and reaching the break point of a job. The algorithm takes action when a new job arrives, a running job completes, or when a running job reaches its break point. When new jobs arrive they will be prioritized, and partitioned among the processors. The job on top of each stack is the job that is running and all other jobs in the stacks are preempted.

A. *Prioritizing:*

The priority of each ready and unscheduled job (located in each pool) at time t which is denoted by $d_{i,j}(t)$ (for $t \leq s_{i,j}$) is variable with time. However, when a job T_k (k can be any i,j) starts its execution, its priority is calculated as $d'_k = \beta_k (s_k + w_k - r_k)$ (lines 19 and 68 of the pseudo-code). The notation d'_k is used for the fixed priority of the running job T_k on top of the stack. This priority is given to the job T_k when it starts its execution. Its start time, $s_{k,}$ is used in the function instead of variable t, therefore its priority is no longer dependent on time. Since s_k , w_k and r_k are all constants, the priority of a job will not change after its start time (for $t > s_k$).

B. *Scheduling / Execution / Preemption:*

Once a new job $T_{i,j}$ is released, if there is a processor such that its stack is empty (lines 11 through 22), then the newly released job is pushed onto the stack and starts running. If there is no idle processor, but for any running processor $d_{i,j}(t) > 4d'_k$ (lines 58 through 66), the job $T_{i,j}$ preempts the currently running one, and starts its execution. The analysis [9] shows that the factor 4 in the preemption condition ($d_{i,j}(t) > 4d'_k$) plays role in constant ratio competitiveness being equal to $10C^2$.

C. *Online Partitioning (Load-Balancing/Greedy):*

If more than one high priority job is able to preempt some running job(s), to decide which job should be sent to which stack, we send the largest job to the processor with the minimum remaining work load, the second largest job to the processor with the second smallest remaining work load, so on so forth. This way we are able to balance the work load among the processors.

However, in case there is only one high priority job at a time instance which can preempt more than one running job, we assign it to the stack of the processor with minimum remaining execution time (Greedy approximation). If the priority of the released job is not high enough to be scheduled right away, it will be partitioned among the pools of the processors using an online choice of load balancing or Greedy approximation (lines 39 through 75).

D. *Reaching Break Point:*

If a job reaches its break point and its execution is not completed yet, it will not be able to gain any benefit; therefore, it will be popped from the stack and sent to the garbage collection. The break point or deadline of a job is either its period or twice its execution time after it starts running, whichever is less.

E. *Reward Accumulation / Completion / Discarding:*

When a currently running job on a processor completes, it is popped from the stack. Then, the processor runs the next job on its stack (i.e. resumes the last preempted job) if $d_{i,j}(t) \leq 4d'_k$ for all the jobs $T_{i,j}$ in its pool. Otherwise, it gets the job with max $d_{i,j}(t)$ from its pool, pushes it onto the stack and runs it. The completed jobs or those that reach their break points are going to be sent to the garbage collection. If a job completes before reaching its break point, its gained benefit is calculated and added to the total benefit.

3. FUTURE WORK

Ongoing work conducts both theoretical and experimental performance analysis of this algorithm. In order to compare it with state-of-the-art, we consider metrics such as total gained reward, tardiness and overall response time. It also studies the upper bounds on task utilization.

REFERENCES

[1] A. Elnably, K. Du, P. Varman, “Reward scheduling for QoS in cloud applications,” in Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012.

[2] J. Gubbi, R. Buyya, S. Marusic , M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” Future Generation Computer Systems 29 (2013) 1645–1660

[3] A. Srinivasan, P. Holman, J. H. Anderson, and S. Baruah, “The case for fair multiprocessor scheduling,” in Proc. of the 11th International Workshop on Parallel and Distributed Real-time Systems, April 2003.

[4] H. Alhussian, N. Zakaria, F. A. Hussin, “An efficient real-time multiprocessor scheduling algorithm,” in Journal of Convergence Information Technology, January 2014.

[5] M. Amirijoo, J. Hansson, and S. H. Son, “Specification and management of QoS in real-time databases supporting imprecise computations,” in IEEE Transactions on Computers, vol. 55, pp. 304–319, March 2006.

[6] H. Aydin, R. Melhem, D. Mosse and P. M. Alvarez, “Optimal reward-based scheduling for periodic real-time tasks,” in IEEE Transactions on Computers, vol. 50, no. 2, February 2001.

[7] I-H. Hou, P.R. Kumar, ”Scheduling periodic real-time tasks with heterogeneous reward requirements,” in Proc. of the 32nd IEEE Real-Time Systems Symposium, 2011.

[8] B. Awerbuch, Y. Azar, and O. Regev, “Maximizing job benefits online,” in Proc. of the 3rd International Workshop, APPROX, Germany, September 2000.

[9] B. Sanati and A.M.K. Cheng, “Maximizing job benefits on multiprocessor systems using a greedy algorithm,” in WiP session of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), April, 2008.

[10] B. Sanati and A.M.K. Cheng, “Efficient Online Benefit-Aware Multiprocessor Scheduling Using an Online Choice of Approximation Algorithms,” in Proc. of the 11th IEEE International Conference on Embedded Software and Systems (ICESS 2014), Paris, France, August 20-22, 2014.

[11] J.H. Anderson, J.P. Erickson, U.C. Devi, B.N. Casses, “Optimal semi-partitioned scheduling in soft real-time systems,” in Proc. of the 20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), August 20-22, 2014.

Towards Worst-Case Bounds Analysis of the IEEE 802.15.4e

Harrison Kurunathan[§], Ricardo Severino[§], Anis Koubaa[§*], Eduardo Tovar[§]
[§]CISTER/INESC TEC and ISEP-IPP, Porto, Portugal
[*] Prince Sultan University, Saudi Arabia.
Email: (hkur, rarss, aska, emt)@isep.ipp.pt, akoubaa@psu.edu.sa

Abstract—**The IEEE 802.15.4e amendment provides important functionalities to address timeliness and reliability in time-sensitive WSN applications, by extending the IEEE 802.15.4-2011 protocol. Nevertheless, in order to make the appropriate network design choices, it is mandatory to understand the behavior of such networks under worst-case conditions. This paper contributes in that direction by proposing a methodology based on Network Calculus that will, by modeling the fundamental performance limits of such networks, enable in the future a quick and efficient worst-case dimensioning of the networks' schedule and resources.**

I. Introduction

Wireless Sensor Networks have been enabling an ever-increasing span of applications and usages in domains such as industrial automation, environmental monitoring and personal health care. Naturally, each use case imposes a different balance of Quality of Service aspects that must be fulfilled in order to guarantee the correctness of the application. In general Cyber-Physical Systems (CPS) for instance, the provision of deterministic guarantees is of crucial importance. In addition, specially in the industrial domain, robustness and reliability are also of increasing importance, considering the harsh environment in which often these systems must be deployed.

To address several of these properties, the 802.15e Working Group proposed the IEEE 802.15.4e amendment, aiming at enhancing and extending the functionalities of the IEEE 802.15.4-2011 protocol. 802.15.4-2011 protocol. This is achieved for instance by proposing several MAC behaviors, which besides providing deterministic communications are also fitted with a multi-channel frequency hopping mechanisms, such as in the case of the Deterministic and Synchronous Multichannel Extension (DSME) and Time Slotted Channel Hopping (TSCH). However, to correctly address the demands in terms of latency and resources, it is mandatory to carryout a thorough network planning. To achieve this, modeling the fundamental performance limits of such networks is of paramount importance to understand their behavior under the worst-case conditions.

In this paper, we present a model for the DSME and TSCH MAC behaviors of IEEE 802.15.4e, based on Network Calculus formalism. The remaining of the paper is organized as follows: in the following section we overview the related work. In Section III we overview the IEEE 802.15.4e protocol and in particular the TSCH and DSME MAC behaviors. The network model for each of these is proposed next and the paper ends with some final remarks and a discussion of future work.

II. Related Work

There are already a few works that analyze the DSME and TSCH performance. The authors in [1] have compared the DSME MAC behaviour of IEEE 802.15.4e to the traditional IEEE 802.15.4 in terms of throughput and end-to end-latency, using an analytical model. The throughput of the DSME MAC protocol was found to be 12 times higher than that of the IEEE 802.15.4 slotted CSMA-CA in a multi-hop network. The DSME MAC behaviour was also analyzed in [2] under WLAN interference, showing that DSME-GTS was much more resilient to interference in comparison with IEEE 15.4 slotted CSMA-CA due to the included channel hopping mechanism.

Concerning TSCH, in [3], and [4], authors have developed analytical models for channel hopping mechanisms, and proposed efficient ideas to extend these, such as black listing and improved frequency hopping sequence algorithms. A comparative assessment [5] of DSME and TSCH MAC behaviors has also been developed using the OMNet++ simulation environment. QoS parameters such as delay, scalability and reliability were computed in this assessment. Interestingly DSME was found to outperform TSCH in terms of end-to-end latency in some scenarios.

The analytical works of the researchers are more dedicated to determine the throughput and end to end latency. In our researc, we propose to define the delay bounds of the MAC behaviors by using network calculus. As far as we know we are the first to use this methodology to determine the delay bounds of 802.15.4e

III. Overview of the IEEE 802.15.4e protocol

The IEEE standard 802.15.4e [6] proposes an enhanced version of IEEE 802.15.4-2011 [7], introducing a set of MAC behaviors which are tailored to suit the needs of industrial real time communications. Ideas which are prominent in the industrial communications field such as frequency hopping, dedicated and shared timeslots and multichannel communication have been implemented in this amendment. In this section, we provide an insight into two MAC behaviors: DSME and TSCH. These aim fundamentally at guaranteeing determinism and enhancing the network's resilience to interference.

A. DSME MAC Behavior

A DSME enabled PAN coordinator uses a multi superframe structure. A multi superframe is composed of a cycle of multiple superframes similar to the IEEE 802.15.4 superframe format. Every superframe in a DSME multi superframe will have a Contention Access Period (CAP) and a Contention Free Period (CFP). Details like the number of superframes in a multi superframe and the timing synchronization are conveyed to the nodes through an enhanced beacon which is transmitted by the PAN Coordinator at the beginning of the multi superframe. The nodes contend for the channel in the CAP region and the CFP is composed multiple Guaranteed Time Slots (GTS). An available GTS slot can be occupied by any pair of nodes within the transmission range, these occupied slots are called DSME GTSs. Figure 1 shows the multi superframe and superframe structure of the DSME MAC behaviour. In the CFP region of the superframe structure in Figure 1, the columns indicate timeslots and the rows indicate the channels available for hopping.

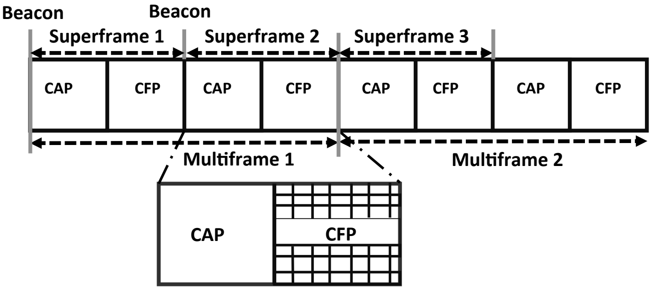


Fig. 1. Multi-frame structure of DSME

B. TSCH MAC Behavior

The concept of superframes has been amended into slotframes in TSCH. Every slotframe is comprised of multiple timeslots. TSCH uses either contention free or contention based communication during the slotframe period, depending on if it is using a guaranteed or a shared timeslot respectively, to transmit a frame and eventually an acknowledgement.

The slotframes are scheduled by the PAN Coordinator and are set to repeat periodically, advertised by enhanced beacons. Multi-channel support is one of the major characteristics of the TSCH MAC behaviour. There are 16 channels available for hopping in TSCH. Every channel is denoted by a channel offset that varies from 0-15. A timeslot absolute number (ASN), which increments globally, is used to compute the channel in any pairwise communication.

Figure 2 shows a three timeslot slotframe in which two devices communicate through 2 channels. In time slot 0 device A transmits its data to B through channel 1 and during time slot 1 B transmits to C through channel 2 and during time slot 2 the device remains in an idle state. The slot frame repeats periodically.

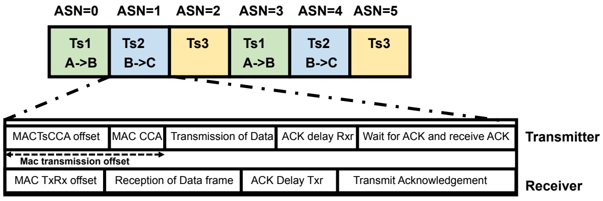


Fig. 2. Three time-slot frame in TSCH

IV. Delay bound using network calculus

Among several analytical methods that have been used to determine the delay bound analysis of distributed networks, Network Calculus is well adapted to controlled traffic sources and provides upper bounds on delays for traffic flows [8]. For a cumulative arrival function $R(t)$ there exists an arrival curve $\alpha(t) = b + r.t$ where b, r, t are the burst rate, data rate and time interval respectively. A minimum service curve $\beta(t)$ is guaranteed to $R(t)$. The maximum delay of the network is given by the horizontal distance between the arrival and the service curves. The delay is computed in accordance to the maximum latency of the service T and the data rate as shown in equation 1:

$$D_{max} = \frac{b}{r} + T, \tag{1}$$

The leaky bucket (b, r) model is used to derive the network models of DSME and TSCH. It is simple and it can represent the higher bound of any kind of traffic. The variance between the (b, r) curve and the realistic model is also adequate for periodic traffic which is commonly the case the of Wireless Sensor Networks. Figure 3 depicts the basic (b, r) model with the arrival and service curves, and the delay bound.

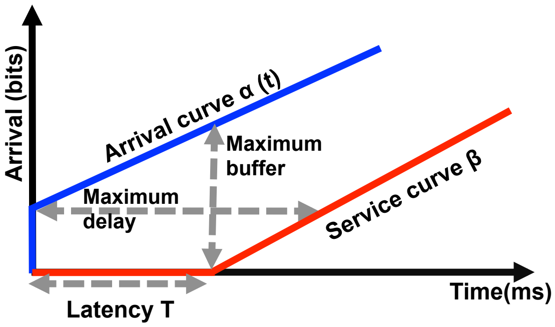


Fig. 3. Arrival curve, service curve, delay bound

A. Service curve analysis of DSME

Let us consider a single PAN coordinator and a set of nodes forming a DSME IEEE 802.15.4e network. The PAN coordinator sends an enhanced beacon for every multi superframe, and a beacon for each superframe. The beacon interval and superframe duration are computed as follows:

$$BI = aBaseSuperframeDuration \quad \times 2^{BO} symbols \tag{2}$$
$$for\ 0 \leq BO \leq 14$$

$$SD = aBaseSuperframeDuration \ \times 2^{SO} symbols \tag{3}$$
$$for\ 0 \le SO \le BO \le 14$$

In these equations $aBaseSuperframeDuration$ represents the minimum length of the superframe (i.e. SO = 0). The IEEE 802.15.4e standard has fixed this value to 960 symbols. Each symbol corresponds to 4 bits, resulting in a duration of 15.36 ms, considering an ideal data rate of 250 Kbps.

It is mandatory that the data transmission, intra-frame spacing and acknowledgements/Group acknowledgements (if requested) are completed within the end of a DSME GTS slot for successful transmission of a message. For the sake of simplicity, we consider one data frame transmission in each a DSME GTS per superframe. As the number of superframes in a multi superframe will remain the same, it is ideal to calculate the delay for a single superframe and multiply by the number of superframes in the multi superframe. Considering the time duration of a superframe is SD, the time duration of a multi superframe will be $Mx(SD)$, where M is the number of superframes. The value of a timeslot in a superframe, Ts is given by equation 5.

$$T_s = \frac{SD}{16} = aBaseSuperframeDuration \ \times 2^{SO-4} \tag{4}$$

Every timeslot T_s in a superframe is made up of T_{data} and T_{idle}. T_{data} is the maximum duration used for data transmission inside the guaranteed timeslots. T_{idle} is the time which is not used by the data, this mainly comprises of the time of inter frame spacing and acknowledgments. The latency is the difference of the bursts arrival and the time the data is served. Burst arrives at the beacon interval. The maximum latency T is given by equation 6, the maximum latency is the time a burst may wait for a service. This is the latency for service provided for the node that allocated one timeslot.

$$T = BI - T_s \tag{5}$$

The overall service provided by the network can be given by the product of the data rate and the time at which system receives the service. For the first superframe, the service curve calculated over time t, this is the minimum number of bits that has to be transmitted during a GTS, this value is given in equation 8.

$$\beta_1 = \begin{cases} C\ ((t-(BI-T_s)))^{+}, \forall\ 0\ \le t\ \le BI -\ t_{idle} \\ 0, \qquad otherwise \end{cases}$$
$$wherex^{+} = max(0,x) \tag{6}$$

This value of the service curve can be derived to N number of superframes, similarly to the equation derived for the service curve for n superframes of IEEE 802.15.4 in [9]. The service of the N_{th} superframe is given by:

$$\beta_N = \begin{cases} (N-1).C.t_{data} + C\,(t-(N.BI-T_N))^{+} \\ \forall 0 \le t \le (N-1)BI - t_{idle} \\ 0, \qquad otherwise \end{cases} \tag{7}$$

The DSME GTS service curves of DSME MAC behaviour is given as a Stair case model in Figure 4.

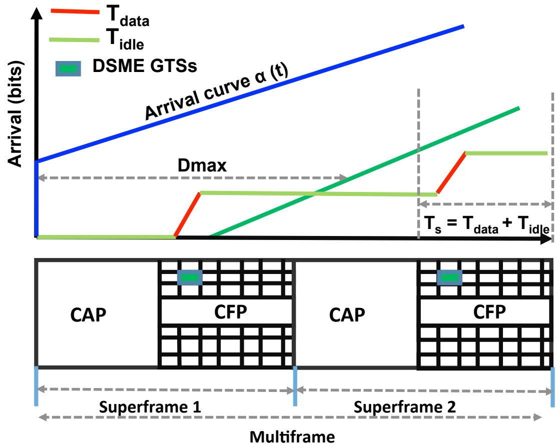


Fig. 4. Service curve of DSME MAC

B. Delay bound analysis of DSME

In a multi superframe the delay bound is calculated for every superframe separately. The sum of these delay bounds will be the overall delay bound of the multi superframe. Considering the burst size b is greater than $C.t_{data}$, the maximum delay bound of the first superframe will be the horizontal angular distance between the arrival curve and the first stair. We consider that a minimum service of (t) will be provided for cumulative data flow $R(t)$:

$$D_{max1} = \frac{b}{C} + (\ BI -\ T_s)\ \ if\ \ b\ \le C.t_{data} \tag{8}$$

In general when $N.(C.T_{data}) < b \le (N+1).C.T_{data})$, the delay of the system with N number of slotframes is given by:

$$D_{maxN} = \frac{b}{C} + (\ (N+1).BI -\ T_s) - N.t_{data} \tag{9}$$
$$ifN(C.T_{data})\ < b \le (N+1).C.T_{data}$$

C. Service curve analysis of TSCH

Although TSCH supports peer-to-peer topologies, in this model we consider only a star topology in which the PAN coordinator sends an enhanced beacon to initiate the slotframe. The aim of this network model is to derive an expression for the delay bound of a data flow $R(t)$ bounded by a (b, r) curve, and that has allocated one timeslot in a slotframe either contention or non-contention-based. The default duration of every timeslot T_s is 10 ms [7], during which it has to accommodate Acknowledgment delays (on the receiver and transmitter end), and the receiver and transmitting frames during a transmission in non-shared timeslot. In shared timeslots, the duration for CCA and CCA CSMA-CA offset also has to be considered.

Every timeslot T_s (time duration of a single timeslot) is of equal duration and is composed of T_{data} and T_{idle}. T_{data} is the maximum duration used for data transmission inside the guaranteed timeslots. T_{idle} is the time which is not used by the data, this mainly comprises of the time of CCA offsets, Acknowledgement delays, MAC transmission and reception offsets and Acknowledgments. Let us consider the fixed duration for which the slotframes repeat in a periodic fashion as T_{cycle}. If a slot is insufficient for a complete message transmission, then the message has to wait for the next timeslot. The latency of the data transmitted in one timeslot of the slotframe is given by:

$$T = T_{cycle} - T_s \tag{10}$$

For the first slotframe, the service curve calculated over a time period t, this is the minimum number of bits that has to be transmitted during a timeslot, this value is given in equation 11.

$$\beta = \begin{cases} C \;\; (t - (T_{cycle} - T_s))^+ \;\; \forall \; 0 \;\; \leq t \;\; \leq T_{cycle} - \;\; t_{idle} \\ 0, \qquad otherwise \end{cases} \tag{11}$$

The service curve will remain constant over time and the entire service of the system can be computed by equation 12:

$$\beta_N = \begin{cases} (N-1).C.t_{data} + C\,(t - (N.t_{cycle} - T_N))^+ \\ \forall 0 \leq t \leq (N-1) \;\; . \,(t_{cycle} - t_{idle}) \\ 0, \qquad otherwise \end{cases} \tag{12}$$

The service curve of TSCH MAC behavior results in a stair case format as depicted in Figure 5.

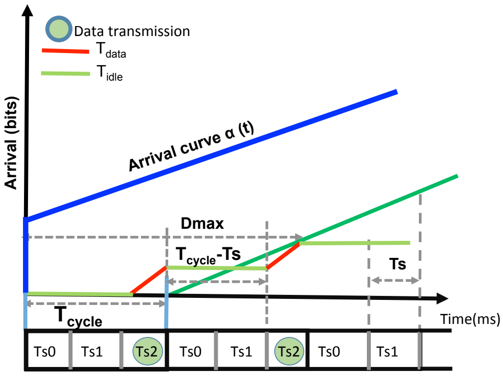


Fig. 5. Service curve of TSCH MAC

D. Delay bound analysis of TSCH

For the first slotframe, assuming $b \leq C.T_{data}$ the maximum delay bound will be the horizontal angular distance between the arrival curve and the first stair. We consider that a minimum service of $\beta(t)$ will be provided for cumulative data flow $R(t)$ using equation 1, resulting as follows:

$$D_{max1} = \frac{b}{C} + (\;\; T_{cycle} - \;\; T_s) \tag{13}$$

The delay of the entire system consisting of N slotframes can be given as:

$$D_{max\ network} = \sum_{0}^{N} . \; D_{maxN} \tag{14}$$

V. Discussion and Future Work

Modeling the performance limits of a network is essential to guarantee the right latency and reliability requirements of a network. In this paper we have derived expressions for computing the worst case delays of DSME and TSCH MAC behaviors using Network Calculus. Though we have provided derivations based on star topology, the proposed results can be extended to all peer-to-peer communication networks. As a continuation of this work the end-to-end delay bounds will be derived for the rest of the MAC behaviors of IEEE 802.15.4e. We also aim at proposing new scheduling algorithms and a simulation model to compare with the analytical results.

Acknowledgment

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0004/2013 - JU grant nr. 621353 (DEWI, www.dewi-project.eu) and ARTEMIS/0001/2012 - JU grant nr. 332987 (ARROWHEAD). This work is also support by the Research and Translation Center (RTC) at Prince Sultan University via Grant Number GP-CCIS-2013-11-10. This work is also supported by the Research and Translation Center (RTC) at Prince Sultan University via Grant Number GP-CCIS-2013-11-10.

References

[1] W.-C. Jeong and J. Lee, "Performance evaluation of ieee 802.15. 4e dsme mac protocol for wireless sensor networks," in *Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012 First IEEE Workshop on*. IEEE, 2012, pp. 7–12.

[2] J. Lee and W.-C. Jeong, "Performance analysis of ieee 802.15. 4e dsme mac protocol under wlan interference," in *ICT Convergence (ICTC), 2012 International Conference on*. IEEE, 2012, pp. 741–746.

[3] P. Du and G. Roussos, "Adaptive time slotted channel hopping for wireless sensor networks," in *Computer Science and Electronic Engineering Conference (CEEC), 2012 4th*. IEEE, 2012, pp. 29–34.

[4] C.-F. Shih, A. E. Xhafa, and J. Zhou, "Practical frequency hopping sequence design for interference avoidance in 802.15. 4e tsch networks," in *Communications (ICC), 2015 IEEE International Conference on*. IEEE, 2015, pp. 6494–6499.

[5] G. Alderisi, G. Patti, O. Mirabella, and L. L. Bello, "Simulative assessments of the ieee 802.15. 4e dsme and tsch in realistic process automation scenarios," in *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. IEEE, 2015, pp. 948–955.

[6] IEEE standard for information technology, WPANs Part 15.4 *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, Sept 2006.

[7] IEEE standard for information technology, WPANs- Part 15.4 amendment 1: Mac sublayer," *IEEE Std 802.15.4e-2012 (Amendment to IEEE Std 802.15.4-2011)*, April 2012.

[8] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[9] A. Koubâa, M. Alves, and E. Tovar, "Energy and delay trade-off of the gts allocation mechanism in ieee 802.15. 4 for wireless sensor networks," *International Journal of Communication Systems*, vol. 20, no. 7, pp. 791–808, 2007.

Demo Papers

TEMPO: Integrating Scheduling Analysis in the Industrial Design Practices

Rafik HENIA, Laurent RIOUX, Nicolas SORDON
Thales Research & Technology
1 Avenue Augustin Fresnel, 91767, Palaiseau Cedex, France
{Rafik.Henia, Laurent.Rioux, Nicolas.Sordon}@Thalesgroup.com

Usually, the industrial practices rely on the subjective judgment of experienced software architects and developers to predict how design decisions may impact the system timing behavior. This is however risky since eventual timing errors are only detected after implementation and integration, when the software execution can be tested on system level, under realistic conditions. At this stage, timing errors may be very costly and time consuming to correct. Therefore, to overcome this problem we need an efficient, reliable and automated timing estimation method applicable already at early design stages and continuing throughout the whole development cycle. Scheduling analysis appears to be the adequate candidate for this purpose. However, its use in the industry is conditioned by a seamless integration in the software development process. This is not always an easy task due to the semantic mismatches that usually exist between the design and the scheduling analysis models. At Thales Research & Technology, we have developed a timing framework called TEMPO that solves the semantic issues through appropriate model transformation rules, thus allowing the integration of scheduling analysis in the development process of real-time embedded software. In this demonstration paper, we present the basic building blocks and functionalities of the TEMPO framework and describe the main visible stages in the model transformations involved.

Keywords—timing verification; scheduling analysis; model-based design; model transformation

I. Introduction

It has always been a challenge to introduce scheduling analysis into the industrial development process as the inputs required for the analysis, in particular the worst-case execution time and the system behavior description, are moving target all across the different development process phases. Thanks to the introduction of model based methods (in particular viewpoints for non-functional properties) in the industrial development process, this goal seems to be reachable. Starting from very high level system architecture and rough timing allocations, the scheduling analysis has to be refined at each step of the project (architectural design, detailed design, coding, unit test and software validation phases) down to concrete timing measurements on the final system. A major problem however persists: scheduling analysis is often not directly applicable to conceptual design due to the semantic gaps between their respective models. Solving this issue is essential to break the remaining walls separating the scheduling analysis from the development process of real-time embedded systems, and to enable its use in the industry.

At Thales Research & Technology, we have therefore developed a timing framework called TEMPO allowing adapting design models to the semantic of the scheduling analysis timing models through a set of transformation rules. The transformation preserves the timing behavior modeled in the conceptual design. After performing scheduling analysis, the obtained results are, in turn, adapted back to the semantic of the design model.

In this demonstration, we present an integrated tool chain from a design modeling tool to a scheduling analysis tool via the timing framework TEMPO and show how the issue of the semantic gaps between design and scheduling analysis is solved.

II. TEMPO Framework Structure

The TEMPO timing framework that we present in this demonstration represents a contribution to the industrial exploitation of model-driven technologies and response time scheduling analysis in the design of real-time systems in a variety of application domains. The TEMPO framework structure is illustrated in Figure 1. It is composed of two building blocks (the TEMPO Design and the TEMPO Analysis pivot models) as well as a set of transformation rules between them.

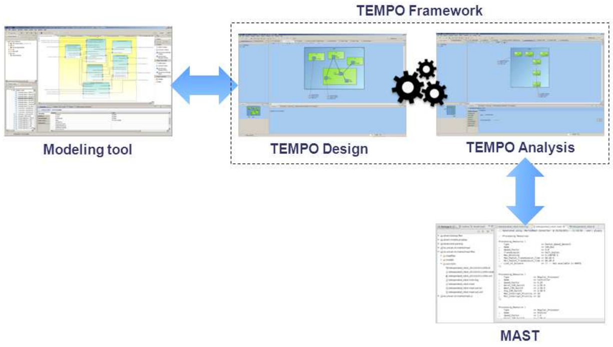


Figure 1: Tool chain including the TEMPO framework

A. *TEMPO Design Pivot Model*

The TEMPO design building block uses a subset of the UML Profile for MARTE standard [1] as a basis to represent a synthetic view of the system design model that captures all elements, data and properties that impact the system timing behavior and that are required to perform the scheduling analysis (e.g. tasks mapping on processors, communication links, execution times, scheduling parameters, etc.). TEMPO Design is not limited to the use of a particular design modeling tool and environment. It can be connected to various environments such as UML, SysML, AADL or any other proprietary environment. This was imposed by the fact that THALES divisions are using various modeling tools, languages and methodologies to design their systems.

B. *TEMPO Transformation Rules*

Scheduling analysis is very often not directly applicable to the conceptual design models in general and to TEMPO Design models in particular due to the semantic mismatch between the latter and the variety of scheduling analysis models known from the classical real time systems research and represented by academic [2] [3] and commercial tools [4]. For instance, in the common scheduling analysis models, a standard assumption is that a task writes its output data at the end of its execution. This is not always the case in design models. Very often in design models, operation calls are either synchronous (blocking) or asynchronous (non-blocking). As a consequence, the task, to which the caller operation is mapped, may write data into the input of a connected task, to which the called operation is mapped, at any instant during its execution and not necessarily at the end. In order to overcome the semantic mismatch between design and scheduling analysis, we have defined a set of rules transforming the TEMPO Design model into a corresponding TEMPO Analysis model, while preserving the initial modeled timing behavior.

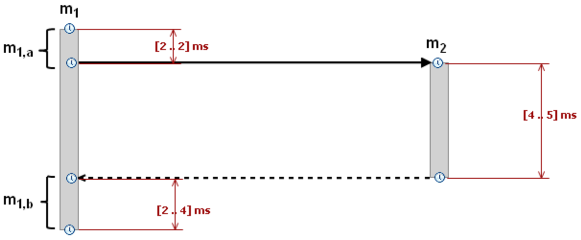


Figure 2: Synchronous call between two operations in the TEMPO Design pivot model

In the following, we present an example of a transformation rule. Figure 2 illustrates an example of a synchronous call between two operations (m_1 and m_2) in TEMPO design. Let us assume that operation m1 (composed of two operation fragments $m_{1,a}$ and $m_{1,b}$) is mapped to a task called T_1, while the operation m_2 is mapped to a task called T_2. Let us assume static priority preemptive scheduling for the tasks. Regardless of the priority assignment for the tasks, the execution order of the operations will always be the following: after its activation, task T_1 will first execute the operation fragment $m_{1,a}$. Then, it calls task T_2. Since the call is blocking, task T_1 is suspended until task T_2 finishes executing the operation m_2 and sends data back. Then, task T_1 executes the operation fragment $m_{1,b}$.

In order to keep the synchronous call behavior of the operations and tasks while being compliant with the scheduling analysis model semantic, we split the operation m_1 in two distinct operations corresponding to the operation fragments $m_{1,a}$ and $m_{1,b}$ as illustrated in Figure 3. We also split task T_1 in two tasks $T_{1,a}$ and $T_{1,b}$ that inherit its priority. Then, we to map the operations $m_{1,a}$ and $m_{1,b}$ respectively to the tasks $T_{1,a}$ and $T_{1,b}$. Obviously, this transformation preserves the same execution order and thus, the synchronous call behavior of the original operations and tasks in the system design model. In addition, it is compliant with the above mentioned timing analysis standard assumption, since task $T_{1,a}$ calls task T_2 at the end of its execution and not before as task T_1 does in TEMPO Design.

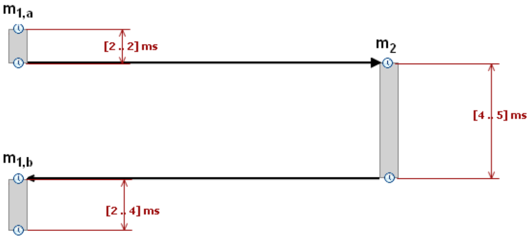


Figure 3: Transformed synchronous call between two operations in the TEMPO Analysis pivot model

C. *TEMPO Analysis Pivot Model*

The TEMPO Analysis pivot model is based on generic modeling concepts known from the classical real time systems research, such as tasks, processors, busses, scheduling parameters (priorities, time slots, deadlines, etc.). TEMPO Analysis models preserve the timing behavior modeled in the corresponding TEMPO Design models, while ensuring the compatibility with the variety of existing scheduling analysis tools. As for TEMPO Design, TEMPO Analysis is not limited to a specific scheduling analysis tool. This ensures a minimum of independence from the analysis tools specificities and allows hiding its complexity to the designer. If required, the used analysis tool can be easily replaced by another. After analysis in the selected scheduling analysis tools, the results are injected in TEMPO Analysis. Then, they are translated to be compliant with the original design model and injected in TEMPO Design

III. Demonstration

Several practical use cases are available as hands-on demonstration of the quality of the TEMPO framework. One in particular might be of interest to the attendees since it appears reported as an industrial challenge for the timing verification of a deployable real system in WATERS 2015 [5].

References

[1] Object Management Group, UML profile for MARTE: Modeling and Analysis of Real‐Time Embedded Systems, version 1.1, OMG document formal/2011‐06‐02, 2011.

[2] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, MAST: Modeling and Analysis Suite for Real‐Time Applications, in Proc. of the Euromicro Conference on Real‐Time Systems, June 2001.

[3] PyCPA: Compositional Performance Analysis in Python ; https://code.google.com/p/pycpa/

[4] SymTA/S: Symbolic Timing Analysis for Systems; https://www.symtavision.com/symtas.html

[5] https://waters2015.inria.fr/files/2014/11/FMTV‐2015‐Challenge.pdf.

Applications of the CPAL language to model, simulate and program Cyber-Physical Systems

Loïc FEJOZ
RealTime-at-Work (RTaW), France
loic.fejoz@realtimeatwork.com

Nicolas NAVET, Sakthivel SUNDHARAM, Sebastian ALTMEYER
University of Luxembourg, Luxembourg
firstname.lastname@uni.lu

Abstract— **CPAL is a new language to model, simulate, verify and program Cyber-Physical Systems (CPS). CPAL serves to describe both the functional behaviour of activities (i.e., the code of the function itself) as well as the functional architecture of the system (i.e., the set of functions, how they are activated, and the data flows among the functions). CPAL is meant to support two use-cases. Firstly, CPAL is a development and design-space exploration environment for CPS with main features being the formal description, the editing, graphical representation and simulation of CPS models. Secondly, CPAL is a real-time execution platform. The vision behind CPAL is that a model is executed and verified in simulation mode on a workstation and the same model can be later run on an embedded board with a timing-equivalent run-time behaviour. The design and development of CPAL have been organized around a set of realistic case-studies that will be demonstrated during the demo session.**

I. MODEL AS THE CODE

CPAL has been initially inspired by the success of three interpretation-based runtime environments, successfully certified at the highest criticality levels and deployed at large scale in railway interlocking systems over the last 20 years at SNCF and RATP in France, and in UK and other countries through the Westlock interlocking system from Westingshouse. Surprisingly to the best of our knowledge, except above applications and some industrial automation (PLCs) *model interpretation* has not been widely explored, albeit it possesses a number of key advantages such as: the model can be directly uploaded on the target, there is no difference between the model and the code, the total software size is greatly reduced both off-line and on the target, hardware independence is ensured, etc.

CPAL supports two types of model interpretation: the direct interpretation of the design models on an interpretation engine running on top of the hardware, called "bare-metal model interpretation" (BMMI), and the interpretation on top of an OS. The latter is less predictable from a timing point of view but more convenient for development and experimentations.

CPAL and its associated tools are jointly developed by our research group at the University of Luxembourg and the company RTaW since 2012. The CPAL documentation, graphical editor and execution engine for Windows, Linux, embedded Linux, and Raspberry Pi are freely available for all uses at http://www.designcps.com. A BMMI port of CPAL is available for Freescale FRDM-K64F boards.

II. CPAL: PROVIDING HIGH-LEVEL ABSTRACTIONS FOR EMBEDDED SYSTEMS

Figure 1, shows that Model-Driven Development is an enabling technology to fill the programming languages gaps. But still existing languages lack the high-level abstractions and automation features that would make them more productive. In addition, the design and development of embedded systems, especially ones with dependability constraints, necessitates the use of many best practices. None of the programming languages we are aware of are well suited to make the development of safe and provably correct embedded systems as quick and easy as possible.

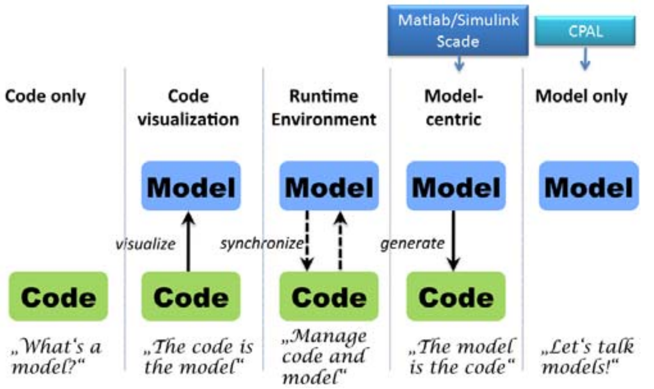


Figure 1: Spectrum of Model-Based Design approaches (core of the figure from [Br04]).

The main requirement when designing CPAL was to natively provide the high-level abstractions which are (i) familiar in the domain of embedded systems, and (ii) needed to express in an unambiguous and concise manner domain specific patterns of functional behaviors as well as non-functional properties. A *process* denotes the core language entity to implement a recurrent activity with its own dynamic. A process is automatically activated at a specified rate, with the optional requirement that a specific logical condition is fulfilled to execute (this is called guarded execution). CPAL processes are classically referred to as tasks, runnables or threads in other contexts.

CPAL provides the programmers with high-level abstractions well suited for the domain of CPS such as

- Real-time scheduling mechanisms: processes are activated with a user-defined period, possibly with an offset relationship with each other, and additional execution conditions such as for instance the occurrence of some external events.

- Finite State Machines (FSM): the logic of a process can be defined as a Finite State Machine (FSM) based on Mode-Automata.
- Communication channels to support data flow exchanges between processes, and reading/writing to hardware ports.
- Introspection mechanisms that enable processes to query at run-time their execution characteristics such as their activation rate and activation jitters.

A key objective behind CPAL is to let designer state the permissible timing behavior of the system in a declarative manner while a system synthesis step involving both analysis and optimization then generates a scheduling solution which at run-time is enforced by the execution environment.

III. Demonstration of CPAL use-cases

CPAL [Na16] supports several use-cases discussed below and that will be demonstrated during the demonstration session.

A. High-level programming language for network simulation environments

CPAL can serve to describe the functional behavior of applications and high-level protocol layers. A CPAL model is for instance used in [Se15] to simulate the SOME/IP Service Discovery protocol in a Daimler Car's prototype network. The model hands over the frames once created to the simulation kernel of RTaW-Pegase, a communication architecture performance analysis tool from RTaW. Interestingly, the same CPAL simulation model can be executed with no changes on an embedded target or a workstation to experiment on a test-bed later in the design process.

This use-case will be briefly demonstrated through a CPAL model that transmits video streams with different coding standards (raw video, H.263) with segmentation an automotive Ethernet network.

B. Modeling and simulation language for Design Space Exploration

CPAL is meant to support the formal description, the editing, graphical representation and simulation of cyber-physical systems. It can be used in its own development environment, like done for the FMTV Challenge [Al15], or within Matlab/Simulink to implement the controller, as done for the landing gear case-study [Bo14]. The simulation models can be executed in real-time (i.e., activation periods are respected) or as fast as possible in simulation mode.

This use-case will be briefly illustrated on the FMTV Challenge 2015, highlighting also the limits of what can be automated.

C. Real-time execution engine

The intention of CPAL is to provide not only a modeling language, but also an interpreter which ensures equivalence between the simulated behavior of the model and the behavior on the execution platform, both in the functional and the temporal domain.

As in Figure 2, this use-case will be demonstrated on the "smart parachute", a remote termination add-on component improving safety of UAS [Ci16] developed in partnership with the company Alérion. The parachute is controlled by a CPAL program, on top of the bare-metal interpreter, executing on a Freescale FRDM-K64F board.

Figure 2: From simulation to field test

D. CPAL for teaching and research

CPAL has been used for teaching Model-Based Design (MBD) for embedded systems since 2012 at the University of Luxembourg at the 3rd year Bachelor level. Practicals include programming a capsule coffee machine, a simplified programmable floor robot and elevator control system, etc. Our experience has been positive in terms of how fast students have been able to work autonomously on the development of the system. Indeed, most students are able to master the language within a few hours. In addition to the simplicity of the language, the free availability of the tools, the on-line examples and the CPAL-Playground facilitate the learning process. CPAL is also used in the experiments of the research conducted at the University of Luxembourg on timing-aware Model-Driven Engineering. We will present small case-studies used in teaching and research outcomes based on CPAL.

References

[Al15a] S. Altmeyer, N. Navet, L. Fejoz, "Using CPAL to model and validate the timing behaviour of embedded systems", WATERS Workshop, July 2015.

[Al15b] S. Altmeyer, N. Navet, "Towards a declarative modeling and execution framework for real-time systems", First IEEE Workshop on Declarative Programming for Real-Time and Cyber-Physical Systems, San-Antonio, USA, December 1, 2015

[Bo14] F. Boniol, V. Wiels, "The landing gear system case study", pp1-18, Proc. ABZ 2014, 2014.

[Br04] A. Brown, "An Introduction to Model Driven Architecture – Part1: MDA and today's systems", IBM technical library, 2004.

[Ci16] L. Ciarletta, L. Fejoz, A. Guenard, N. Navet, "Development of a safe CPS component: the hybrid parachute, a remote termination add-on improving safety of UAS", to appear at ERTSS2016, Toulouse, January 2016.

[Na16] N. Navet, L. Fejoz, L. Havet, S. Altmeyer, "Lean Model-Driven Development through Model-Interpretation: the CPAL design flow", Embedded Real-Time Software and Systems (ERTS 2016), Toulouse, France, January 27-29, 2016.

[Se15] J. Seyler, T. Streichert, M. Glaß, N. Navet, J. Teich, "Formal Analysis of the Startup Delay of SOME/IP Service Discovery", DATE 2015, Grenoble, France, March 9-13, 2015.

Demonstration of the FMTV 2016 Timing Verification Challenge

Arne Hamann, Dirk Ziegenbein, Simon Kramer, Martin Lukasiewycz
Robert Bosch GmbH
Corporate Research, Germany
Email: {arne.hamann|dirk.ziegenbein|simon.kramer2|martin.lukasiewycz}@de.bosch.com

***Abstract*—In [1] we argued that the complex dynamic behavior of automotive software systems, in particular engine management, in combination with emerging multi-core execution platforms, significantly increased the problem space for timing analysis methods. As a result, the risk of divergence between academic research and industrial practice is currently increasing.**

Therefore, we provided a concrete automotive benchmark for the Formal Methods for Timing Verification (FMTV) challenge 2016 [2], a full blown performance model of a modern engine management system based on [1], with a goal to challenge existing timing analysis approaches with respect to their expressiveness and precision.

In the demo session we will present the performance model of the engine management system using the Amalthea tool [3]. Furthermore, we will show the model in action using professional timing tools such as from Symtavision [4], Timing Architects [5], and Inchron [6]. By this means, the demo gives an impression of the current state-of-practice in industrial product development, and serves as baseline for further academic research.

I. The FMTV 2016 Challenge

In short, the FMTV 2016 challenge consists in determining tight end-to-end latency bounds for a set of given cause-effect chains in a full blown engine management software. For this purpose, an Amalthea [3] performance model of the software can be downloaded at [2].

As mentioned above, the dynamic behavior of a engine management software is quite complex and contains mechanisms that explore the limits of existing approaches:

- preemptive and cooperative priority based scheduling
- periodic, sporadic, and engine synchronous tasks
- multi-core platform with distributed cause-effect chains including cross-core communication
- label (i.e. data) placement dependent execution times of runnables

The provided Amalthea model contains a hardware model of a simplified microcontroller architecture with four symmetric cores (see Figure 1). The cores are interconnected by a crossbar (full connectivity, FIFO arbitration at memories). The system-wide frequency is 200 MHz. Furthermore, initial mappings (runnable to task, task to core) are specified.

Each core $CORE_x$ is connected to a local memory $LRAM_x$. Additionally, there exist a global memory $GRAM$ that is shared among all cores. The specified runnable execution times assume that code is executed directly from core-exclusive flashes without contention. In contrast, access to labels including memory arbitration effects are not included in the execution times. Initially, all labels are assumed to be stored in the global memory. The following symmetric memory access times are assumed:

- Reading from and writing to the global memory: 9 cycles
- Reading from and writing to the core local memory: 1 cycle
- Reading from and writing to the local memory of a different core: 9 cycles

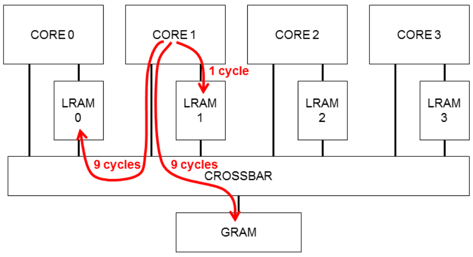


Fig. 1. Microcontroller architecture used in the challenge

The memory access model assumes that runnables read all required labels at the beginning of their execution, afterwards the calculation takes place, before all labels are written back. Local RAMs are single-ported, such that concurrent accesses to the same memory lead to contention, and are arbitrated according to the FIFO policy.

Obviously, solving the intertwined problem of scheduling including the effects of memory accesses to the execution times is very hard. Therefore several separate challenges are formulated:

- calculate tight end-to-end latencies ignoring memory accesses and arbitration
- calculate tight end-to-end latencies including memory access and arbitration accesses
- optimize end-to-end latencies by mapping the labels among the local and global memories

II. Approaches

Figure 2 visualizes different approaches that are addressed by the FMTV 2016 challenge. Two Pareto-fronts show the general trade-off between accuracy and effort of the different approaches.

The first Pareto-front converges towards the the actual worst-case coming from formal and, thus, conservative approximations. Here, compositional timing analysis methods, on the one hand, are very efficient in terms of computational complexity and modeling efforts, but usually lead to overestimated worst-case bounds, especially for distributed systems. Formal method like timed automata, on the other hand, scale badly to large systems due to the underlying exponential nature of model checking.

The second Pareto-front shows simulation-based approach that optimistically underestimate the worst-case. Here, the challenge lies in finding stimuli for the system under simulation, leading to values that are close to the worst-case since enumerating and simulating all possible situation is infeasible from a practical point-of-view.

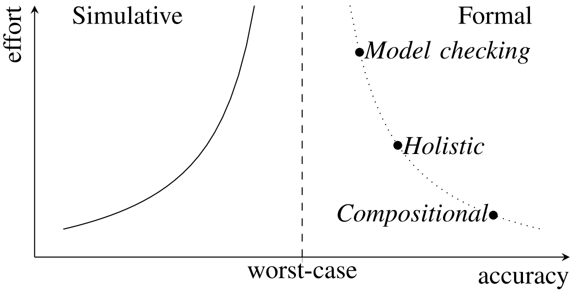


Fig. 2. Illustration of different approaches with respect to accuracy and effort.

In a nutshell, novel methods should improve state-of-the-art and not be dominated by any method on the Pareto-front. That means, either improve in terms of accuracy or reduce effort.

Please note, that the demo focuses on applying industrial-strength simulation tools (see Section II-B) to determine latencies for the end-to-end cause-effect chains formulated in the FMTV 2016 challenge. Nevertheless, we also briefly discuss formal approaches that we would like to be seen applied to the challenge.

A. Formal Approaches

Classical real-time scheduling [7], [8] considers tasks on a single processor and their schedulability, taking into account execution times, release times, and deadlines. These approaches use problem-specific formalizations to model systems and cannot be applied directly to distributed systems with heterogeneous components, schedulers and protocols. An extension of the classical approaches towards distributed systems is known as *holistic scheduling* [9], [10]. Here, the equations of the specific scheduling approaches are combined by introducing dependency formalizations. Due to the quickly growing complexity of this approach, its applicability is limited by the fast increasing number of equations and dependencies that are introduced with each component in the system.

In contrast to holistic scheduling, compositional approaches promise a better extensibility by relying on components that exchange information via event streams [11], [12], [13]. These event streams capture properties like periodic behavior or jitter while end-to-end latencies can be determined by adding delays along the entire data flow. These compositional approaches can deliver relatively tight latencies, but further reduction of the inherent over-approximations of the determined latencies are obliviously limited by the information in the event streams.

Of course a system might be modeled using timed automata such that the resulting end-to-end latencies are exact when applying model checking. However, as the approaches scale exponentially, they are generally integrated as components into compositional approaches [14].

B. Simulative Approaches

In contrast to the formal approaches, simulative approaches do not require an abstraction of the model and can therefore be easily extended with new components, schedulers, and protocols. Without the need to abstract the model, simulative approaches are theoretically capable of determining the exact latency values without any over-approximation. However, determining the right stimuli for the simulation that will result in the actual worst-case latency is extremely difficult, and therefore usually randomized inputs are chosen. It is obvious that with a longer runtime of simulative approaches, the undesirable under-approximation can be reduced.

Due to their good usability and simple integration, simulation tools like TIMING ARCHITECTS SIMULATOR [5], CHRONSIM [15], or SYMTA/S with TRACEANALYZER [4] enjoy great popularity in industry. Nevertheless, for safety critical application it can be dangerous to rely on simulative approaches as they do not guarantee to capture the actual worst-case even if a certain safety margin is added to the observed worst-case after extensive simulations.

C. Hybrid Approaches

Finally, hybrid approaches that combine simulations and formal approaches might be considered. These approaches might use traces of simulations for the input of formal methods to deliver a worst-case that is larger than the simulation result and at the same time lower than a purely analytically determined value. Another approach to mix the two views of simulation and formal approach is the consideration of typical worst-case analysis [16]. Here, a so-called typical worst-case is determined that is only violated by a strictly bounded number of occurrences in a given time window. This approach is very useful in specific scenarios where occasional deadline violations do not affect the correct behavior of the system. As a result, this approach is applied in combination with the domain knowledge of the underlying system, e.g., a control application.

REFERENCES

[1] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmark for free," in *Sixth International Workshop on Analysis Tools and Methodologies for Embedded Real-time Systems (WATERS)*, 2015.

[2] "Formal Methods for Timing Verification (FMTV) Challenge 2016. Bosch Engine Management System." http://ecrts.eit.uni-kl.de/forum/viewtopic.php?f=27&t=62.

[3] "Amalthea 4 Public Project," http://www.amalthea-project.org/.

[4] "Symtavision GmbH," https://www.symtavision.com/.

[5] "Timing Architects," http://www.timing-architects.com/.

[6] "Inchron GmbH," https://www.inchron.de/.

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

[8] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2-3, pp. 101–155, 2004.

[9] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and microprogramming*, vol. 40, no. 2, pp. 117–134, 1994.

[10] J. P. Gutiérrez, J. G. García, and M. G. Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proceedings of the Ninth Euromicro Workshop on Real-Time Systems*. IEEE, 1997, pp. 136–143.

[11] K. Gresser, "An event model for deadline verification of hard real-time systems," in *Real-Time Systems, 1993. Proceedings., Fifth Euromicro Workshop on*. IEEE, 1993, pp. 118–123.

[12] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, vol. 4. IEEE, 2000, pp. 101–104.

[13] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the symta/s approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[14] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," in *Proceedings of the seventh ACM international conference on Embedded software*. ACM, 2009, pp. 107–116.

[15] "ChronSIM," http://www.inchron.com/tool-suite/chronsim.html.

[16] S. Quinton, T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst, "Typical worst case response-time analysis and its use in automotive network design," in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC '14. New York, NY, USA: ACM, 2014, pp. 44:1–44:6. [Online]. Available: http://doi.acm.org/10.1145/2593069.2602977

Response-Time Analysis for Task Chains in Communicating Threads with pyCPA

Johannes Schlatow, Jonas Peeck and Rolf Ernst
Institute of Computer and Network Engineering, TU Braunschweig
{schlatow,jonasp,ernst}@ida.ing.tu-bs.de

***Abstract*—When modelling software components for timing analysis, we typically encounter functional chains of tasks that lead to precedence relations. As these task chains represent a functionally-dependent sequence of operations, in real-time systems, there is usually a requirement for their end-to-end latency. When mapped to software components, functional chains often result in communicating threads. Since threads are scheduled rather than tasks, specific task chain properties arise that can be exploited for response-time analysis by extending the busy-window analysis for such task chains in static-priority preemptive systems. We implemented this analysis by means of an analysis extension for pyCPA, a research-grade implementation of compositional performance analysis (CPA).**

The major scope of this demo is to show how CPA can be reasonably performed for realistic component-based systems. It also demonstrates how research on and with CPA is conducted using the pyCPA analysis framework. In the course of this demo, we show two approaches for the extraction of an appropriate timing model: 1) the derivation from a contract-based specification of the software components and 2) a tracing-based approach suitable for black-box components. We also demonstrate how this timing model is fed into the analysis extension in order to obtain response-time results for the task chains of interest. Finally, we present how the developed analysis extension speeds up the CPA and therefore enables an automated design-space exploration and optimisation of the threads' priority assignments in order to satisfy the pre-defined latency requirements.

I. Introduction

Larger embedded systems are often implemented as a collection of functions, each described as a task graph. To derive end-to-end response times in such task graphs, we are interested in the response times between the respective tasks. In this paper, we are interested in task chains which are derived from communicating software threads leading to specific task chain properties that can be exploited for response-time analysis covering both synchronous and asynchronous communication. Such communicating threads have become the standard implementation vehicle, e.g. in modern automotive software components [1] or in microkernel-based systems [2], [3].

When we try to perform a timing analysis for these systems, we first encounter a mismatch between the programming model and the timing analysis model: On the one hand, the programmer implements a thread that communicates at arbitrary points in its execution using the available primitives such as **synchronous IPC** or **asynchronous notifications** which are prominent in microkernel-based systems. On the other hand, the timing architect models the system by tasks that are only allowed to communicate at the end of their execution, implicitly assuming asynchronous communication semantics.

Figure 1 shows a thread (Thread 1) that (synchronously) calls another thread (Thread 2) at some point in its execution. Thread 1 can only continue after the latter completed and returned. The right side of the figure illustrates how this scenario is reflected in the timing model by a chain of tasks that represent the segments of the threads ($1a$, 2 and $1b$) along with their precedence relations.

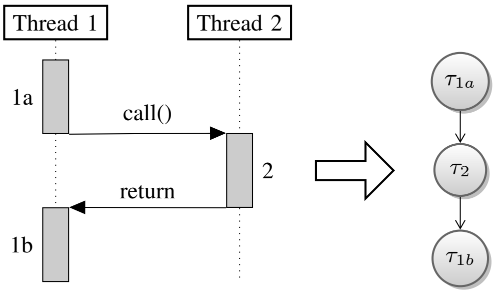


Figure 1. Communicating threads (implementation) naturally split up into a chain of tasks (timing model).

Although this is a straightforward transformation, it already obfuscates important information that should be respected by the timing analysis: The task timing model does not reflect the blocking behaviour of the synchronous call, i.e. it does not reflect that τ_{1a} cannot execute again before τ_2 returned. It neither represents the execution dependencies between the thread segments, i.e. that τ_{1a} cannot execute before τ_{1b} finished.

In [4], we presented an extension of the busy-window response-time analysis which exploits the particular semantics in task chains resulting from communicating threads in static-priority preemptive (SPP) systems. This approach is able to cope with varying priorities along the chain and even reduces the computational effort and overestimation in comparison to conventional CPA.

We implemented this by means of an extension of the pyCPA analysis framework [5], [6], which is specifically tailored for easy and modular extensibility. This implementation augments the task graph model provided by the pyCPA analysis kernel with additional information about the task chains and their activation semantics: In this extended model, we differentiate between the activation semantics, i.e. we distinguish synchronous (i.e. blocking) from asynchronous (i.e. non-blocking) edges in the task graph. As our response-time analysis approach [4] considers entire task chains as opposed to single tasks, we introduce a preprocessing step in which the superordinate task chains are defined. Note that this preprocessing can either be done manually by the designer or timing architect, or automatically by tool support.

II. Demo

In the scope of this demo, we will present the design flow that can be applied for the timing verification during the integration of component-based systems. We assume that the implementation of the software components is already completed and that some of the software components may be provided by a third party, i.e. their source code is either not available or not fully understood. Furthermore, we already obtained a valid composition of the components either by a manual designer-driven process or an autonomous approach [7]. The remaining task is thus to extract a timing model for the system in order to evaluate and explore possible scheduling parameters. As we use SPP scheduling, the only scheduling parameter is the priority of a thread. Our demo splits up into a model-extraction and analysis part that we illustrate on an exemplary but realistic application. The software components and run-time environment used for our demonstrator are based on the Genode OS Framework [8].

A. Model extraction

We demonstrate two methods for extracting the timing model for a particular application in a system. On the one hand, we apply a contract-based approach that requires a formal description of a component, its threads and their interaction with other components via the specified interfaces. On the other hand, we show a tracing-based approach in which the components' behaviour is extracted by acquiring tracing information of one or multiple test runs.

1) Contract-based: The contract-based model extraction is based on an abstract formal specification of the components implementation. In order to extract a timing model, a component's contract must specify an abstract task graph, which dissects the threads into sequential parts (i.e. tasks) and communication directives (i.e. synchronous or asynchronous communication to another thread or another component). Each task is annotated with an upper and lower bound on its execution time. As the communication partner is not necessarily known during component development, the communication directives must be (automatically) resolved after a valid composition is chosen in order to link the abstract task graphs of all components. In this part, we demonstrate how our tools automatically derive this composite timing model from the contracts of our example application.

2) Tracing-based: For the tracing-based model extraction, we execute the software components (i.e. their composition) on the target platform and record their execution times and communication pattern. For this purpose, we leverage the existing tracing infrastructure of the Genode OS Framework in order to record the corresponding timestamps. We further developed a set of tools that process these traces as well as extract and visualise Gantt charts, similar to the sequence diagram in Figure 1, that assist the manual derivation of a timing model. Note that these tools can also be used for a validation of an existing timing model. It should furthermore be mentioned, that this approach is not solely suitable for the timing verification of critical application as, in general, the traces do not capture the entire range of timing behaviour.

B. Analysis, evaluation and exploration

Once we acquired a timing model of our application, we can run the response-time analysis for all paths of interest using pyCPA and the analysis extension presented in [4]. By this, we can explore the performance of different scheduling parameters and investigate their feasibility w.r.t. the applications' latency constraints specified by contracts. We demonstrate this by conducting an automated design-space exploration and by visually inspecting the analysis results in relation to the solution space defined by the contracts. This eventually allows as to optimise the scheduling parameters and validate the expected results by executing our example application on our demonstrator in order to record, extract and visualise the corresponding traces.

Acknowledgements

This work was supported by the DFG Research Unit Controlling Concurrent Change (CCC), funding number FOR 1800. We thank the members of CCC for their support.

References

[1] AUTOSAR website. [Online]. Available: http://www.autosar.org/

[2] PikeOS Hypervisor. [Online]. Available: https://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/

[3] seL4 Microkernel. [Online]. Available: https://sel4.systems/

[4] J. Schlatow and R. Ernst, "Response-time analysis for task chains in communicating threads," in *22nd Real-Time Embedded Technology and Applications Symposium (RTAS 2016)*, Vienna, Austria, April 2016.

[5] pyCPA website. [Online]. Available: https://bitbucket.org/pycpa/pycpa

[6] J. Diemer, P. Axer, and R. Ernst, "Compositional Performance Analysis in Python with pyCPA," in *3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Jul. 2012.

[7] J. Schlatow, M. Moestl, and R. Ernst, "An extensible autonomous reconfiguration framework for complex component-based embedded systems," in *12th International Conference on Autonomic Computing (ICAC 2015)*, Grenoble, France, July 2015, pp. 239–242.

[8] Genode OS Framework. [Online]. Available: http://genode.org/

Run-Time Monitoring Environments for Real-Time and Safety Critical Systems

Geoffrey Nelissen, Humberto Carvalho, David Pereira, Eduardo Tovar
CISTER/INESC TEC, ISEP
Polytechnic Institute of Porto
Porto, Portugal
Email: {grrpn, hjesc, dmrpe, emt}@isep.ipp.pt

***Abstract*—In this work, we present four different implementations of a run-time monitoring framework suited to real-time and safety critical systems. Two implementations are written in Ada and follow the Ravenscar profile, which make them particularly suited to the development of high integrity systems. The first version is available as a standalone library for Ada programs while the second has been integrated in the GNAT run-time environment and instruments the ORK+ micro-kernel. Information on the task scheduling events, directly originating from the kernel, can thus be used by the monitors to check if the system follows all its requirements. The third implementation is a standalone library written in C++ that can be used in any POSIX compliant run-time environment. It is therefore compatible with the vast majority of operating systems used in embedded systems. The last implementation is a loadable kernel module for Linux. It has for main advantage to be able to enforce complete space partitioning between the monitors and the monitored applications. It is therefore impossible for memory faults to propagate and corrupt the state of the monitors.**

I. Introduction

As a part of the development process of embedded systems, there is a need to verify that the functional and timing requirements defined in the system specifications will always be respected after the system deployment. This is even more important for safety critical systems, which must go through a thorough certification process. However, with the increasing complexity of embedded systems, it becomes always more complicated and sometimes impossible to statically verify offline that all the requirements will be respected at run-time. Specifically, with the advent of multicore processors, several new challenges arose: (i) the manufacturers sacrificed the determinism of their computing platforms to improve the average case performances, (ii) the number of applications running concurrently on the same processor and hence competing for the shared resources, is increased, (iii) the applications are becoming more complex and make use of intra-task parallelism to take advantage of the processing power offered by the several cores. Additionally, the integration of applications developed by different companies or development teams, the utilisation of legacy code and/or the lack of access to the source code of some of the executed functionalities, render the verification process even more complex.

Under such conditions, it becomes unrealistic to formally verify that all the system requirements will be respected under any possible execution scenario. The worst-case analyses that are usually performed before the system deployment are also based on a set of assumptions (e.g., minimum activation period, worst-case execution time, maximum release jitter) that may not always be respected at run-time. For all those reasons, run-time monitoring and run-time verification become an interesting alternative to the traditional offline verification. Run-time verification is based on the instrumentation of the target applications. Monitors are then added to the system to verify at run-time that the system requirements are respected during the execution. If a misbehaviour is detected, an alarm can be raised so as to trigger appropriate counter-measures (e.g., execution mode change, reset or deactivation of some of the functionalities).

Run-time monitoring and verification can be used during the system development phase to test and debug the applications. However, the monitors can also be left in the system after its deployment, in which case they play the role of a safety net, preventing the system to enter in an unexpected or dangerous state.

Safety related standards recommend the use of run-time monitoring and verification solutions in safety critical systems [1]–[3]. However, their use is not limited to safety critical applications. They can be very useful for the development of mission critical and business critical applications, or simply to improve the reliability of any embedded system.

II. Reference Architecture

In [4], a reference architecture for a safe and reliable run-time monitoring framework was proposed. This architecture is depicted on Figure 1. It is based on four main components: (i) event buffers in which events (i.e., a timestamp associated to a data) can be pushed by the instrumented application, (ii) event writers used by the monitored application to push events in the buffers, (iii) event readers that may be used by the monitors to access the events that are saved in the buffers, and (iv) monitors, implemented as periodic tasks, that read events and check that the application respect its specifications.

As shown on Figure 1, there can be only one writer per buffer. This avoids parallel accesses to the same buffer, which may have lead to unwanted blocking times. Thanks to this restriction, the writing operation in a buffer is wait free. There can however be more than one reader connected to the same

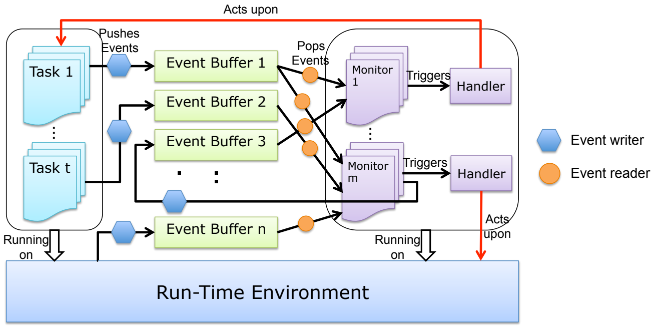


Fig. 1. Run-time monitoring reference architecture [4].

buffer, which allows several monitors to use the same events in parallel.

III. Implementations for Different Execution Environments

Four different implementations of the reference architecture proposed in [4] have been developed and should be presented during the demo session.

A. *Ravenscar Compliant Ada Library*

The first implementation is written in Ada, a programming language particularly suited to the development of critical applications. The library respects all the restrictions associated with the Ada Ravenscar profile [5]. The Ravenscar profile was defined to ensure timing predictability and hence ease the timing analysis of critical applications, by enforcing strict coding rules.

The developed library can be used in any application written in Ada (Ravenscar compliant or not). It provides all the needed facilities to instantiate monitors, buffers, buffer readers and writers discussed above.

B. *Integration in the ORK+ Micro-Kernel*

ORK+ is a Ravenscar compliant micro-kernel [6] implemented in Ada and integrated in the GNAT GPL 2011 compilation system developed by AdaCore. The kernel is packaged together with the compiler and the other libraries proposed by the GNAT runtime environment. ORK+ is currently one of the reference run-time environments in the ESA EagleEye reference mission [7] used for testing new technologies for future space applications.

The Ada library mentioned in the previous section was added to the GNAT runtime environment and has been used to instrument the ORK+ micro-kernel. This means that monitors can now have access to task scheduling related events extracted directly at the kernel level. Those events are saved in a set of predefined buffers that can be accessed by user-defined monitors.

C. *POSIX Compliant C++ Library*

The third implementation is written C++ and assumes a POSIX execution environment. It can thus be used in a vast majority of real-time operating systems available for embedded applications (e.g., Linux, RTEMS, ...). Similarly to the Ada library, the C++ version offers all the facilities required for the implementation of an efficient run-time verification framework compliant with the reference architecture described in Section II. Each monitor is encapsulated in a POSIX thread which is periodically executed.

D. *Integration in Linux as a Kernel Module*

In addition to the POSIX implementation, a loadable kernel module has been implemented for Linux. The major advantage of this Linux implementation is that it achieves total space partitioning. Indeed, the monitors can be instantiated in different processes than the monitored application. As each process runs within its own virtual sandbox, it is impossible for a monitor or a monitored application generating memory errors to corrupt monitors instantiated in other processes. Additionally, the buffers are living at the kernel level while the monitors and monitored applications are instantiated at the user level. Given that the event buffers are allocated in kernel memory, they cannot be corrupted, and will persist even if the monitored application crashes, thereby allowing the monitors to continue extracting critical information even when the system malfunctions. Finally, kernel land allows for finer-grained control over the hardware. The preemptions can thus be disabled during critical sections, guaranteeing wait-free read and write operations in the buffers.

IV. Demonstration

During the demo session, we will (i) show how applications running in the different execution environments described in the previous section can easily be instrumented, (ii) show how monitors can be implemented either for logging, for runtime verification purposes or for providing adaptive capabilities to the instrumented application, (iii) provide indications on the impact of the framework on the application performances.

Acknowledgments. This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER); also by FCT/MEC and the EU ARTEMIS JU within project(s) ARTEMIS/0003/2012 - JU grant nr. 333053 (CONCERTO) and ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

References

[1] DO-178C, *Software Considerations in Airborne Systems and Equipment Certification*. RTCA, Inc., 2011.
[2] ISO26262, *Road vehicles Functional safety*. ISO, 2011.
[3] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "How realistic is the mixed-criticality real-time system model?" in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*. ACM, 2015, pp. 139–148.
[4] G. Nelissen, D. Pereira, and L. M. Pinho, "A novel run-time monitoring architecture for safe and efficient inline monitoring," in *Reliable Software Technologies–Ada-Europe 2015*. Springer, 2015, pp. 66–82.
[5] A. Burns, B. Dobbing, and T. Vardanega, "Guide for the use of the ada ravenscar profile in high integrity systems," *ACM SIGAda Ada Letters*, vol. 24, no. 2, pp. 1–74, 2004.
[6] Universidad Politécnica de Madrid, "ORK+," 2014. [Online]. Available: http://www.dit.upm.es/ str/ork/index.html
[7] V. Bos, P. Mendham, P. K. Kauppinen, N. Holst, A. Crespo Lorente, M. Masmano, J. A. d. l. Puente Alfaro, and J. R. Zamorano Flores, "Time and space partitioning the eagleeye reference mission," 2013.

Timing Aware Hardware Virtualization on the L4Re Microkernel System

Adam Lackorzynski[†,‡], Alexander Warg[†]

Kernkonzept GmbH[†]
Dresden, Germany

Technische Universität Dresden[‡]
Operating-Systems Group
Dresden, Germany

Email: adam.lackorzynski@kernkonzept.com,
alexander.warg@kernkonzept.com

adam.lackorzynski@tu-dresden.de

***Abstract*—Hardware virtualization support has found its way into real-time and embedded systems. It is paramount for an efficient concurrent execution of multiple systems on a single platform, including commodity operating-systems and their applications. Isolation is a key feature for these systems, both in the spatial and temporal domain, as it allows for secure combinations of real-time and non real-time applications. For such requirements, microkernels are a perfect fit as they provide the foundation for building secure as well as real-time aware systems. Lately, microkernels learned to support hardware-provided virtualization features, morphing them into microhypervisors. In our demo, we show our open-source and commercially supported L4Re system running Linux and FreeRTOS side by side on a multi-core ARM platform. While for Linux we use the hardware features for virtualization, i.e., ARM's virtualized extension, we revert to paravirtualization for running the FreeRTOS guest. Paravirtualization adapts the guest kernel to run as a *native* application on the microkernel. For simple guests that do not use advanced hardware features such as virtual memory and multiple privilege levels, virtualization is simplified and the state of a virtual machine is significantly reduced, improving interrupt delivery and context switching latency. Both guests as well as the native application drive LEDs to exemplify steering actual devices as well as to show their liveliness. Taking down the Linux guest will not disturb the others.**

I. Introduction

Virtualization technology enables many interesting application scenarios, which require combining commodity off-the-shelf applications and real-time tasks in a secure, dependable and, most importantly, timing preserving manner. For example, cyber-physical systems such as autonomous cars, UAVs for wood-fire detection and SCADA systems, which besides many applications control our power grid, combine latency sensitive tasks such as model predictive control tasks for engines and road situations, flight stabilization and grid stability with maintenance tasks or other, less timing critical tasks that benefit greatly from the extended execution environments of commodity operating-systems (OSs).

Consider, for example, an autonomous driving scenario. As long as a safe exit to the emergency lane can be maintained at all points in time and in all situations and as long as this exit route can be executed entirely in the real-time subsystem, resource intensive tasks such as vision, scenery analysis and maneuver planning can remain in rich commodity environments, which speed up development time and reduce costs but sacrifice stringent timing guarantees. The aforementioned assumptions allow the real-time system to transition into a fail-safe mode for those situations where the commodity operating system ceases to respond in a timely manner.

However, for real-time tasks to operate reliably next to commodity OSs and their applications, faults in the latter must be confined and timing guarantees of the former preserved.

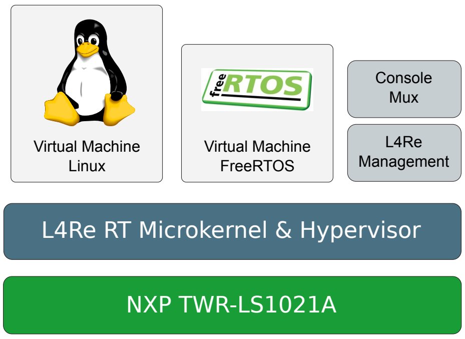


Fig. 1. Demo Setup, virtualized Linux and FreeRTOS running on an ARM platform.[1]

In this demo, we show how our open-source and commercially supported L4Re microkernel system [1], [2] exploits ARMv7's hardware virtualization capabilities to consolidate Linux and FreeRTOS on a multi-processor platform. The commodity OS Linux and the real-time kernel FreeRTOS are run independently of each other in two virtual machines (VMs), which prevents any malfunctioning of one to affect the other. While for Linux we use ARM's hardware virtualization capabilities, the FreeRTOS guest is paravirtualized.

II. The L4Re Microkernel System

L4Re is a capability-based third-generation microkernel-based system [1]. It evolved starting from the DROPS real-time system [3], later including secure system construction [4] and a major interface redesign towards a capability-driven security model [2] while keeping its real-time roots. With L4Re one can now securely isolate real-time and non real-time applications in a single system.

[1] Tux logo copyright by Larry Ewing, Simon Budig, Anja Gerwinski; FreeRTOS logo from http://www.freertos.org/

The system consists of the L4Re kernel and the L4Re user-level infrastructure that provides the necessary framework to build a wide range of applications, including services. Through its capability design and thus the inherent local naming scheme, interposing interfaces has become an essential part of the system, which allows easily exchanging and enhancing of the functionality of the system. For example, by interposing part of the scheduling interface, new core placement policies can be added and efficiently executed.

III. Hardware vs. Software Virtualization

Classical real-time operating systems, such as FreeRTOS, run on systems that do not provide hardware features for isolation, such as virtual memory and multiple privilege levels. Thus, the virtualization requirements for such guests are much simpler, allowing them to be virtualized as a native user-level process of the microkernel. In comparison to a hardware-virtualized virtual machine, the state of such a task, which is to be maintained and stored by the kernel during a context switch, is much smaller. Therefore, unless additional hardware is added to simultaneously maintain multiple virtual machines, paravirtualization reduces interrupt latency by requiring the kernel to capture only the user-level state of paravirtualized guests.

To further reduce this latency while allowing for fine grained control and scheduling of VM interrupts, we implemented the scheduling context scheme proposed in [5]. In this scheme, multiple scheduling contexts (an abstraction of time) can be attached to threads and VMs be activated upon arrival of an interrupt. Upon such an arrival, the host kernel scheduler compares the interrupt's and current scheduling context's priority to determine whether it can inject this interrupt and schedule the VM immediately or whether a higher prioritized task is present.

For commodity operating systems (such as Linux), virtualization is more involved. These systems use multiple privilege levels and virtual memory, which evades a naive virtualization using a user-level task. Although, source availability provided, it is possible to para-virtualize these systems [6], [7], hardware features for virtualization offer a major benefit as they provide additional privilege levels and virtual memory capabilities [8], [9]. However, these additional hardware features come at the cost of a larger state to be context switched, which translates to higher latencies and response times.

IV. Beyond Virtualizing Guest OSs

While virtualization is a crucial feature to isolate subsystems, virtualization by itself is not of much use. Instead, tasks, which implement a functionality while running inside the VMs, must also be able to interact with the outside world through sensors, actuators and other devices. Of course, guarantees about the timeliness of such device accesses must be preserved by the virtualization layer, in particular if part of the system becomes compromised.

To exemplify this requirement in our demo, the virtualized guest OSs (Linux and FreeRTOS) each steer an LED as an example of a more complex device and to report their health status. In addition, a native task of the host system performs the same operation to resemble scenarios where real-time functionality is implemented as native L4Re applications. Figure 1 shows this setup.

In the demo, the Linux guest can be commanded interactively and none of the activities within the Linux, including crashing the whole VM, shall affect the execution of the FreeRTOS guest or of the native application. The LEDs of the application and FreeRTOS task display the situation accordingly by showing no difference in their blinking behavior while the LED driven by Linux will eventually stop blinking. It remains in the state, which corresponds to the last setting made by Linux.

This elementary demo setup shall illustrate what is possible on modern microkernel-based systems today and inspire more sophisticated usage scenarios. We invite everyone interested to try out the open-source L4Re system, available at [1].

References

[1] "L4Re microkernel system," https://l4re.org/.

[2] A. Lackorzynski and A. Warg, "Taming Subsystems: Capabilities as Universal Resource Access Control in L4," in *IIES '09: Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems*. Nuremberg, Germany: ACM, 2009, pp. 25–30.

[3] H. Härtig, R. Baumgartl, M. Borriss, C.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter, "DROPS: OS support for distributed multimedia applications," in *Proceedings of the Eighth ACM SIGOPS European Workshop*, Sintra, Portugal, Sep. 1998.

[4] H. Härtig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter, "The Nizza Secure-System Architecture," in *In IEEE CollaborateCom 2005*. IEEE Press, 2005.

[5] A. Lackorzynski, M. Völp, and A. Warg, "Flat but trustworthy: Security aspects in flattened hierarchical scheduling," *SIGBED Rev.*, vol. 11, no. 2, pp. 8–12, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2668138.2668139

[6] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter, "The performance of μ-kernel-based systems," in *Proceedings of the 16th ACM Symposium on Operating System Principles (SOSP)*, Saint-Malo, France, Oct. 1997, pp. 66–77.

[7] "L^{4}Linux," https://l4linux.org/.

[8] ARM Limited, *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*, ARM DDI 0406C.c ed., 2014.

[9] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B, 3C and 3D*, 325462-057US, December 2015 ed., 12 2015.

Predictable SoC architecture based on COTS multi-core

Nitin Shivaraman
Nanyang Technological University
Email: nshivaraman@ntu.edu.sg

Sriram Vasudevan
Nanyang Technological University
Email: sriram006@e.ntu.edu.sg

Arvind Easwaran
Nanyang Technological University
Email: arvinde@ntu.edu.sg

***Abstract*—With the increasing complexity of real-time embedded applications and the availability of Commercial-Off-The-Shelf (COTS) multi-cores, time-predictable execution on these platforms has become a necessity. However, there are several challenges to achieving this predictability, primarily arising due to hardware resources shared between the cores (memory controllers, caches and shared interconnect).**

In this demo, we present a novel System-on-Chip (SoC) architecture based on COTS multi-cores that address some of these challenges. Specifically, we develop an architecture that enables COTS multi-cores to predictably access external memory. This SoC is designed using hybrid hardware platforms comprising a COTS multi-core and closely coupled Field Programmable Gate Array (FPGA), e.g., Xilinx Zynq ZC706. In our design, the COTS multi-core (ARM Cortex-A9 dual-core) is integrated using a high-speed interconnect with an arbiter module and the Memory Interface Generator (MIG) Xilinx memory controller on the FPGA. Through experiments we show that the proposed architecture has a precisely predictable worst-case memory access latency when compared to a COTS-only design.

I. Introduction

Embedded systems support time-critical and safety-critical functionalities such as those applications in automotive, avionics, medical systems, etc. In such Real-Time Systems (RTS), being able to predict the Worst-Case Execution Time (WCET) of the application on the underlying hardware platform is an essential requirement and we refer to this property as *time-predictability*. In order to support the increasing hardware performance demands of such RTS, multi-core processors seem like a natural choice. One of the primary advantages of using multi-cores is its Size, Weight and Power (SWaP) characteristics making the die fit into a single package. Execution-time of an application running on a core depends on how the application on the other cores uses the shared resources of the multi-core. Additionally, COTS multi-cores are generally preferable in safety-critical applications due to a long service history in a variety of applications. The proposed architecture illustrates the potential of using COTS multi-cores together with closely coupled Programmable Logic (PL) for real-time embedded applications with strict timing requirements. We use Xilinx Zynq ZC706 platform for this demo to illustrate the predictability of the architecture.

II. Related Work

Several studies have been done to obtain predictability in COTS multi-cores. Some of these focused on analytical techniques to estimate the WCET (e.g., [1] and [2]). These techniques have limited applicability due to either unrealistic assumptions or lack of information about the architecture.

There have also been several studies that propose modifications to the hardware such as arbitration schemes for interconnect and controllers (e.g., [3], [4], [5], [6] and [7]). However, they do not address the problem of how COTS multi-cores can be modified to provide such capabilities. Our customizable SoC architecture is built to address this problem. Also, the techniques proposed in the above studies are orthogonal to our work and can be implemented in the PL of our architecture.

Another category of research focuses on building the entire multi-core from scratch on FGPA (e.g., [8]). Considering the SWaP characteristics and service history of COTS multi-cores against FPGA-based designs, our architecture offers the advantage of using as many COTS components as possible with minimal support from FPGAs.

The work by Sha et.al. proposes single core equivalence of multi-core processors by combining different software-level techniques to address predictability challenges of various shared hardware components [9]. Such software-level mechanisms have limitations in terms of the arbitration policies that can be implemented, and may also have substantial implementation overheads.

III. Proposed Architecture

The proposed architecture is shown in Figure 1. In this design, memory requests from the Processing System (PS) are re-routed to the FPGA/PL and eventually sent to the DDR which is connected to the PL. The key idea of this design is that the components that have been identified to cause unpredictability in COTS multi-cores (memory controllers, cache hierarchy and shared interconnect) are disabled and their functionalities are handled separately on the FPGA. This gives us the flexibility to provide custom solutions to handle memory requests in a predictable manner.

In this initial phase of the architecture, we disable caches at all levels as we focus on the feasibility of predictably routing memory requests from PS to PL. As seen from Figure 1, the memory access requests from the cores bypass the Snoop Control Unit (SCU), the L1 and L2 caches and arrives at the master interconnect. As the cores are address mapped to the General Purpose port 1 (GP port physically connects

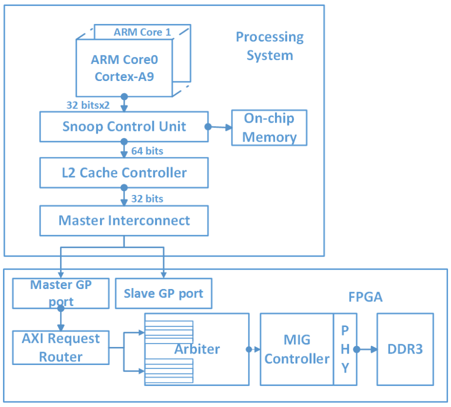


Fig. 1. Predictable SoC architecture

the ARM cores with the FPGA), the request that arrives at the interconnect pass through this port and are received by the First In First Out (FIFO) buffers which are present in the arbiter module of the design. There are two separate buffers which collect the memory requests based on the core that generates the request. The arbiter module implements a round-robin arbitration scheme based on which it selects the memory request from the buffers and sends them to the Memory Interface Generator (MIG) memory controller [10]. This arbiter module manages the scheduling of requests external to the MIG controller thereby providing memory access in a predictable fashion. MIG interfaces to a PHY (physical layer) which in turn connects to the external DDR memory.

IV. Demo

In this demo, we show the predictability of the proposed architecture by performing two different experiments and displaying the output as graphs. The experiment setup would consist of the Xilinx Zynq ZC706 board connected to a host for programming and UART display. The first experiment is to measure the latency for each memory transaction (reads and writes) and the second is to measure the throughput of the architecture. The experiment details are presented below:

1) For single core latency experiments, we perform 100 memory transactions of which every 10 transactions are to a single row. After 10 transactions we switch to a new row. All transactions are performed to the same bank. This would demonstrate the predictable nature of access. We will also be able to observe row switching and refresh in the DDR memory precisely.
2) For all dual core experiments, we shall use core 0 as the observing (measuring) core and core 1 will be the one which causes interference.
3) For dual core latency experiments, we perform 100 memory transactions from core 0 to a single row and a single bank. On the other hand, core 1 will continuously perform memory transactions to the same row and bank in a loop. This is done to make sure that core 0 memory transactions suffer sufficient interference from memory transactions of core 1.
4) For latency experiments, to measure the time for each memory transaction, we initially measure the overhead of the timer (global clock of Zynq 706) by starting and halting the timer without performing any memory transaction. This overhead is subsequently negated from the time measured for all memory transactions.
5) The results of the experiments are plotted as a histogram of latency versus the frequency of occurrence of the latencies. This would show the evidence of the different latencies measured, described in steps 1 and 3.

V. Work in Progress

The main contribution of this work is to demonstrate an architecture which uses COTS multi-core extensively and supported by logic on FPGA to predictably access memory. This initial design has the MIG memory controller which is externally supported by an arbiter module which predictably schedules memory transactions from both the cores. However, the arbiter has an overhead which reduces the throughput of the current architecture. The private and shared caches are also disabled in this architecture.

The next step to this work in progress is to replace the MIG with our custom memory controller solution. The custom memory controller solution will integrate the arbiter unit along with the memory controller which will not only help to improve the throughput but also help to implement fine grain arbitration techniques. The inclusion of a predictable caching architecture is also a part of the future work to make the architecture robust and practical to be used in RTS.

Acknowledgment

This work was funded in part by MoE Tier-1 grant number RG21/13.

References

[1] M. Lv, W. Yi, N. Guan, and G. Yu, “Combining abstract interpretation with model checking for timing analysis of multicore software,” in *RTSS, 2010 IEEE 31st*, Nov 2010, pp. 339–349.

[2] D. Hardy and I. Puaut, “WCET analysis of multi-level set-associative instruction caches,” *CoRR*, vol. abs/0807.0993, 2008.

[3] M. Paolieri, E. Quiones, F. Cazorla, and M. Valero, “An analyzable memory controller for hard real-time cmps,” *Embedded Systems Letters, IEEE*, vol. 1, no. 4, pp. 86–90, Dec 2009.

[4] J. Reineke, I. Liu, H. D. Patel, S. Kim, and E. A. Lee, “Pret dram controller: Bank privatization for predictability and temporal isolation,” ser. CODES+ISSS ’11, pp. 99–108.

[5] Y. Krishnapillai, Z. P. Wu, and R. Pellizzoni, “A rank-switching, open-row dram controller for time-predictable systems,” in *Real-Time Systems (ECRTS), 2014 26th Euromicro Conference on*, July 2014, pp. 27–38.

[6] E. Lakis, “FPGA implementation of a time predictable memory controller for a chip-multiprocessor system,” Master’s thesis, Technical University of Denmark, DTU, 2013.

[7] B. Akesson, K. Goossens, and M. Ringhofer, “Predator: A predictable sdram memory controller,” ser. CODES+ISSS ’07, 2007, pp. 251–256.

[8] M. Paolieri, J. Mische, S. Metzlaff, M. Gerdes, E. Quiñones, S. Uhrig, T. Ungerer, and F. J. Cazorla, “A hard real-time capable multi-core smt processor,” *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 3, Apr. 2013.

[9] L. Sha, M. Caccamo, R. Mancuso, J.-E. kim, M.-K. Yoon, R. Pellizzoni, H. Yun, R. Kegley, D. Perlman, G. Arundale, and R. Bradbord, “Single core equivalent virtual machines for hard real-time computing on multicore processors,” Tech. Rep., 2014.

[10] Zynq-7000 SoC and 7 Series Devices Memory Interface Solutions. http://www.xilinx.com/products/intellectual-property/mig.html

A real-time low datarate protocol for cooperative mobile robot teams

Gaetano Patti*, Giovanni Muscato*, Nunzio Abbate#, Lucia Lo Bello*
* Department of Electrical, Electronics and Computer Engineering, University of Catania, Catania, Italy
{gaetano.patti, lobello}@unict.it, giovanni.muscato@dieei.unict.it

STMicroelectronics, Catania, Italy
nunzio.abbate@st.com

I. Introduction

Mobile cooperating robot teams are increasingly used in several applications, for instance, to access and explore dangerous areas for humans (e.g., nuclear plants, minefields or volcanoes). As in these applications the mobile robots have to communicate in order to cooperate and fulfill a common task, communication plays a key role and has to meet the requirements imposed by the applications [1], i.e., bounded end-to-end delays, mobility support, high scalability, and low costs. Recent works investigate the combination between cooperating mobile robot applications and Wireless Sensor Networks (WSNs), e.g., applications in which mobile robots are considered the mobile sensors of a WSN [2]. This demo presents RoboMAC, a new MAC protocol for communicating between mobile cooperating robots. RoboMAC enables the integration of robots with WSNs, supports real-time communications and mobility, and provides high scalability. RoboMAC was implemented on the STMicroelectronics SPIRIT1 Sub-GHz devices, which operate on less crowded frequencies than the other Industrial, Scientific and Medical (ISM) ones and provide a higher radio coverage.

The main contributions of RoboMAC are summarized below.

- It enables the integration of robots with WSNs, being specifically devised for low datarate communications.
- It provides support to mobility, combining clustering with a distributed topology management mechanism based on Received Signal Strength Indicator (RSSI) assessments.
- It provides scalable real-time communications thanks to the combination of a TDMA-based mechanism with multichannel transmissions and clustering.

II. Overview on The RoboMAC Protocol

As mentioned in the Introduction, the RoboMAC protocol aims to provide bounded delays, mobility support, scalability, and low cost. The way RoboMAC achieves each of these properties is described in the following.

A. Bounded delays

To provide bounded delays, a TDMA transmission scheme is implemented, in which the time is divided into superframes, which are cyclically repeated. Each superframe is, in turn, divided into slots. A node is assigned one or more slots in which it is allowed to transmit a single frame. In RoboMAC each node schedules its messages according to their priority. Here static priorities are assumed, which derive from and depend on the application. In Fig. 1. The RoboMAC node architecture is shown. The MAC layer provides two sublayers, i.e., the Medium Access and Synchronization sublayer and the Clustering and Routing one. The upper sublayer communicates with the lower one through two prioritized queues (i.e., the IntraCluster and the InterCluster PrioQueue, respectively) for the frame transmissions, and one FIFO queue for the incoming frames.

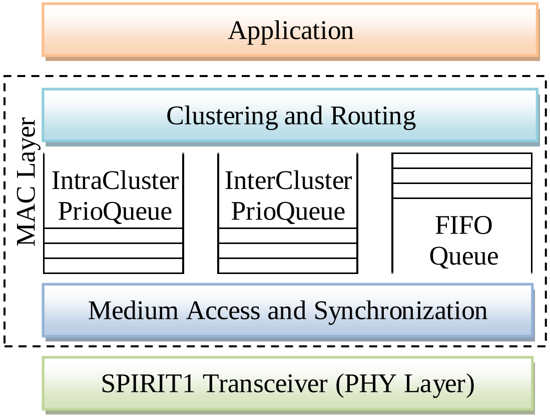


Figure 1 Node architecture

Node synchronization is achieved by taking into account the transmission time of other nodes. As the start time of a frame transmission is known to all the network nodes, when a node receives a frame it calculates the difference between the expected reception time of the first byte of the frame and the actual time at which such a byte is received. Such a value is the time difference between the two nodes. If the frame is received outside a guard interval, it is not taken into account for the synchronization. The guard interval is the same for all the network nodes and it depends on the maximum synchronization skew supported by the application.

B. Scalability

To efficiently support large networks, in RoboMAC the nodes are organized in clusters, depending on their position in the network (i.e., the nodes that are close to each other belong to the same cluster). During the network initialization, the position of network nodes is estimated by exchanging a matrix containing the Received Signal Strength Indicators (RSSI), which holds the link relations between the nodes. In this way the nodes are aware of the network topology. Two transmission channels are used, one for intracluster communications, the other for intercluster ones. Intercluster communications are allowed between the nodes of two clusters when the relevant clusters do not have on-going intracluster communications.

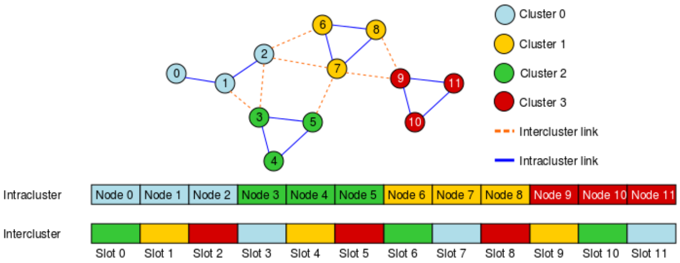


Figure 2 RoboMAC network example.

Looking at the example in Fig. 2, during the slots assigned to the Cluster 2 nodes (i.e., Nodes 3, 4, and 5), the nodes of Clusters 0, 1 and 3 can communicate on a different channel provided that the message destination does not belong to Cluster 2. Intercluster slots are assigned to the nodes in turn.

C. Mobility support

RoboMAC provides dynamic clusters, so their composition varies over time. In fact, due to the mobility, the distances change, and so the RSSI. For this reason RoboMAC provides a distributed topology management, which is based on the RSSI that is regularly acquired during the communications. Mobility issues, like the unpredictability of the network topology, are solved transmitting either intercluster or intracluster topology information within the header of each frame.

D. Low cost

The RoboMAC protocol has been implemented on the STMicroelectronics STEVAL-IKR002V5 [3] board (Fig. 3), a commercial-off-the-shelf device available at low cost.

III. The Demo

The demo presents the RoboMAC protocol implementation on the STMicroelectronics STEVAL-IKR002V5 board, which is composed of a motherboard equipped with a STM32L1 family microcontroller (MCU) and the SPIRIT1 transceiver, which operates at 915MHz and provides a datarate of 250kbps. The communication between the MCU and the transceiver goes through the SPI port, which operates at 1Mbps.

The slot duration is configured so as to accommodate the delays caused from the communication between the transceiver and the MCU, the channel switching time, and the transmission delay.

The demo will show the protocol in a scenario made up of 5 nodes. One board is connected to a PC that acts as a graphical interface to show, online, the network status, the transmitted frames, and the relevant timings.

Figure 3. STMicroelectronics STEVAL-IKR002V5 boards.

During the interactive session the RoboMAC protocol configuration will be shown. Several examples of communications will demonstrate how the protocol works and how it can offer bounded latencies (in the order of hundreds of milliseconds) on COTS low datarate devices. Moreover, several videos of cooperative mobile robot applications will be shown. In the first application two robots (Fig. 4) cooperate to search a radio target that periodically transmits beacons, while in the second application the two robots cooperate in order to maintain the connectivity during the exploration of an area.

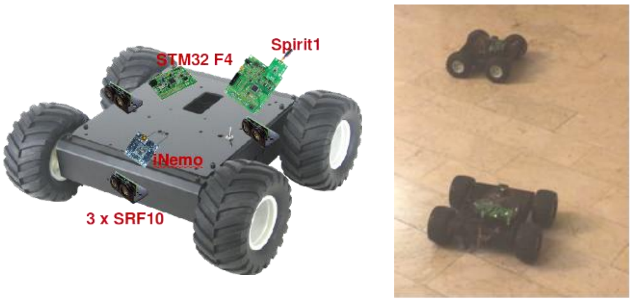


Figure 4. The cooperating robots during an experiment.

Experimental results, in terms of maximum message latencies and packet loss ratio of the two cooperative mobile robot applications will be also presented.

References

[1] G. Patti, G. Muscato, N. Abbate, and L. Lo Bello, “Towards Low-datarate Communications for Cooperative Mobile Robots”, IEEE World Conference on Factory Communication Systems (WFCS), Palma de Mallorca, Spain, 27-29 May 2015.

[2] A. Koubaa and J.Ramiro Martínez-de Dios, Eds., Cooperative Robots and Sensor Networks 2015, ser. 604. Springer, 2015.

[3] STMicroelectronics STEVAL-IKR002V5, “SPIRIT1 - low data rate transceiver - 915 MHz - full kit”, Apr. 2014, online available: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/data_brief/DM00093462.pdf.