

Second Workshop on Next Generation Real-Time Embedded Systems

NG-RES 2021, January 20, 2021, Budapest, Hungary

Edited by

Marko Bertogna

Federico Terraneo



Editors

Marko Bertogna 

Università di Modena e Reggio Emilia, Italy
marko.bertogna@unimore.it

Federico Terraneo 

Politecnico di Milano, Italy
federico.terraneo@polimi.it

ACM Classification 2012

Computer systems organization → Real-time systems; Computer systems organization → Embedded and cyber-physical systems

ISBN 978-3-95977-178-8

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <https://www.dagstuhl.de/dagpub/978-3-95977-178-8>.

Publication date

January, 2021

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <https://portal.dnb.de>.

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0):
<https://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASlcs.NG-RES.2021.0

ISBN 978-3-95977-178-8

ISSN 1868-8969

<https://www.dagstuhl.de/oasics>

OASlcs – OpenAccess Series in Informatics

OASlcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASlcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

ISSN 1868-8969

<https://www.dagstuhl.de/oasics>

■ Contents

Preface	
<i>Marko Bertogna and Federico Terraneo</i>	0:vii
Program committee	
.....	0:ix

Invited Paper

A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems	
<i>Khalil Esper, Stefan Wildermann, and Jürgen Teich</i>	1:1–1:12

Regular Papers

EDF Scheduling and Minimal-Overlap Shortest-Path Routing for Real-Time TSCH Networks	
<i>Miguel Gutiérrez Gaitán, Luís Almeida, Pedro Miguel Santos, and Patrick Meumeu Yomsí</i>	2:1–2:12
Static Allocation of Basic Blocks Based on Runtime and Memory Requirements in Embedded Real-Time Systems with Hierarchical Memory Layout	
<i>Philipp Jungklass and Mladen Berekovic</i>	3:1–3:14
Event-Based Control Enters the Real-Time World: Perspectives and Pitfalls	
<i>Silvano Seva, William Fornaciari, and Alberto Leva</i>	4:1–4:11
M2OS-Mc: An RTOS for Many-Core Processors	
<i>David García Villaescusa, Mario Aldea Rivas, and Michael González Harbour</i>	5:1–5:13



■ Preface

This volume collects the papers presented at the second edition of the Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2021). The workshop is co-located with the 2021 edition of the HiPEAC conference and was held on January 20th, 2021. Although the workshop was originally planned to take place at Budapest, Hungary, due to the COVID-19 pandemic it switched to a virtual online event.

The traditional concept of embedded systems is constantly evolving to address the requirements of the modern world. Cyber-physical systems, networked control systems and Industry 4.0 are introducing an increasing need for interconnectivity. A steadily increasing algorithmic complexity of embedded software is fueling the adoption of multicore and heterogeneous architectures. As a consequence, meeting real-time requirements is now more challenging than ever. The NG-RES workshop focuses on real-time embedded systems, with particular emphasis on the distributed and parallel aspects. The workshop is a venue for both the networking and multicore real-time communities aiming at cross-fertilization and multi-disciplinary approaches to the design of embedded systems.

The scope of the NG-RES workshop include the following topics:

- Programming models, paradigms and frameworks for real-time computation on parallel and heterogeneous architectures
- Networking protocols and services (e.g., clock synchronization) for distributed real-time embedded systems
- Scheduling and schedulability analysis for distributed and/or parallel real-time systems
- System-level software and technologies (e.g. RTOSs, hypervisors, separation kernels, virtualization) for parallel and heterogeneous architectures
- Application of formal methods to distributed and/or parallel real-time systems
- Compiler-assisted solutions for distributed and/or parallel real-time systems
- Middlewares for distributed and/or parallel real-time systems

In this second edition of the workshop four regular papers were accepted, each of which receiving three peer reviews. In addition, we are glad to have an invited paper by Khalil Esper, Stefan Wildermann and Jürgen Teich titled “A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems”. We would like to thank the authors of the NG-RES 2021 papers, the members of our program committee, our publisher Schloss Dagstuhl as well as the HiPEAC organizers for contributing to the success of this workshop.

Marko Bertogna and Federico Terraneo



■ Program committee

General Chair

- Marko Bertogna, Università di Modena e Reggio Emilia, Italy

Program Chair

- Federico Terraneo, Politecnico di Milano, Italy

Web and Submission Chair

- Federico Reghenzani, Politecnico di Milano, Italy

Program committee

- Alberto Leva, Politecnico di Milano, Italy
- Alessandro Vittorio Papadopoulos, Mälardalen University, Sweden
- Benny K. Akesson, TNO, Netherlands
- Christine Rochange, Institut de Recherche en Informatique de Toulouse, France
- Francisco J. Cazorla, Barcelona Supercomputing Center, Spain
- Jaume Abella Ferrer, Barcelona Supercomputing Center, Spain
- Lucia Lo Bello, University of Catania, Italy
- Luís Almeida, Universidade do Porto, Portugal
- Martina Maggio, Lund University, Sweden
- Marco Solieri, Università di Modena e Reggio Emilia, Italy
- Roberto Cavicchioli, Università di Modena e Reggio Emilia, Italy



A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems

Khalil Esper 

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
khalil.esper@fau.de

Stefan Wildermann 

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
stefan.wildermann@fau.de

Jürgen Teich 

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
juergen.teich@fau.de

Abstract

Many applications vary a lot in execution time depending on their workload. A prominent example is image processing applications, where the execution time is dependent on the content or the size of the processed input images. An interesting case is when these applications have quality-of-service requirements such as soft deadlines, that they should meet as good as possible. A further complicated case is when such applications have one or even multiple further objectives to optimize like, e.g., energy consumption.

Approaches that dynamically adapt the processing resources to application needs under multiple optimization goals and constraints can be characterized into the *application-specific* and *feedback-based* techniques. Whereas application-specific approaches typically statically use an offline stage to determine the best configuration for each known workload, feedback-based approaches, using, e.g., control theory, adapt the system without the need of knowing the effect of workload on these goals.

In this paper, we evaluate a state-of-the-art approach of each of the two categories and compare them for image processing applications in terms of energy consumption and number of deadline misses on a given many-core architecture. In addition, we propose a second feedback-based approach that is based on finite state machines (FSMs). The obtained results suggest that whereas the state-of-the-art application-specific approach is able to meet a specified latency deadline whenever possible while consuming the least amount of energy, it requires a perfect characterization of the workload on a given many-core system. If such knowledge is not available, the feedback-based approaches have their strengths in achieving comparable energy savings, but missing deadlines more often.

2012 ACM Subject Classification Hardware → Power and energy; Hardware → Finite state machines; Computing methodologies → Computational control theory; Computer systems organization → Self-organizing autonomic computing

Keywords and phrases energy optimization, control-theory, timing analysis, soft real-time, dynamic voltage and frequency scaling, finite state machines, multi-core, many-core

Digital Object Identifier 10.4230/OASICS.NG-RES.2021.1

Category Invited Paper

Funding This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research-Foundation) – Project Number 146371743 - TRR 89 Invasive Computing.



© Khalil Esper, Stefan Wildermann, and Jürgen Teich;
licensed under Creative Commons License CC-BY

Second Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2021).

Editors: Marko Bertogna and Federico Terraneo; Article No. 1; pp. 1:1–1:12

OpenAccess Series in Informatics



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

1 Introduction

Many applications have different requirements that should be met during their execution on modern many-core platforms. Embedded systems often have energy and temperature constraints due to their limited power budget. Interesting is the case when an application comes with more than one non-functional requirement, e.g., latency, power or energy consumption, or safety requirements. In this paper, we consider image processing applications as a class of streaming applications that often require the results to be ready within a defined latency, and at the same time should consume a minimal amount of energy on a given platform. For instance, self-driving cars should timely detect crossing people, and also consume as little energy as possible.

Application-specific approaches for latency-aware energy optimization statically (at design time) determine or approximate a set of operating points where each is optimized for a specific workload scenario [18]. Then, for each workload scenario, a set of actions is determined (e.g., voltage/frequency and/or selection of the number of cores to execute given workload) in order to achieve a set of requirements (e.g., a deadline) while optimizing a cost function, e.g., energy. For example, [6] uses a profiling stage to determine a set of Pareto points. While such approaches rely solely on design-time techniques, other approaches like [9] build an offline model from the profiling data and then adapt it online using machine learning algorithms. However, the disadvantage of these approaches is their lack of ability to adapt to unseen workloads or, more severe, new applications that were not analyzed at the offline stage [9]. The authors of [8] use performance and analytical models online to evaluate a machine learning strategy that was derived offline, in order to adapt to unseen workloads. However, it still needs a profiling stage offline to decide an initial strategy and also cannot be generalized as it depends on the analytical models that are used online.

As a remedy, approaches applying control theory to adapt the execution of applications to current needs have been proposed [3, 7, 10, 11, 12, 13, 16]. Control-theoretical approaches apply an *observe-decide-act* loop, which is a three-stages control strategy. First, the system monitors the application to obtain a latency feedback, then it decides on the actions to be applied by computing a generic control signal that reflects the control system's status and the obtained feedback, and finally it applies those decided actions. The main advantage of these approaches is that the system no longer needs to know the expected workloads or analyze the applications that it will control before start of the execution. Another advantage for using control theory is that if the system can be proven to be *stable*, this implies that the system output will be bounded or return to a desired value in a bounded amount of time [1].

As an alternative to control-theoretic approaches, with equally strong mathematical soundness and proof qualities, finite state machines (FSMs) can be used. FSMs are heavily used in digital design, industrial control and robotics. As such, stability properties might equally be proven by state reachability. Moreover, contrary to many control-theoretic approaches proposed in literature such as [3, 10, 11, 13, 16], no assumptions need to be made on the linearity of as well plant as control behaviours.

Contributions. In this paper, we evaluate and compare both an application-specific [17, 18] and a feedback-based [4] state-of-the-art approach using control theory, and propose a very simple alternative feedback-based approach using FSMs. For streaming-based image processing applications, i.e., video sequences with content-based workload, we subsequently compare these approaches in terms of deadline misses and energy savings for a given many-

core platform. We conclude that application-specific approaches have their strength in case of highly predictable workload whereas the feedback-based approaches have their strength in rather unpredictable and even to some degree to so far unseen workloads.

The remaining of this paper is structured as follows. Section 2 introduces the different approaches for latency-aware energy optimization that shall be analyzed and compared. Section 3 introduces the workload and experiments as well as results on their evaluation and comparison. Finally, in Section 4, we conclude this work.

2 Latency-Aware Energy Optimization Approaches

In this section, we introduce one application-specific and two feedback-based approaches for optimizing energy while preserving soft deadlines for many-core platforms. A given (soft) deadline for execution of periodic workload, e.g., images, shall be enforced while minimizing the amount of consumed energy. The latter can be influenced by variation of as well the number of cores n as well as by dynamic power management, i.e., voltage/frequency scaling. Modern processors allow to execute a program in m different voltage/frequency configurations. In addition, we consider a strategy called race-to-idle as an example of a heuristic approach that is neither application-specific nor feedback-based. Race-to-idle makes the system execute in the fastest possible configuration (e.g., highest voltage/frequency settings m_{\max} and number of cores n_{\max}) in order for the application to finish and the system to become idle as fast as possible. However, this method does not provide energy-optimality as has been shown in [5].

2.1 Application-Specific Approaches

Application-specific approaches which are used here synonymously with offline approaches require a knowledge base so they can choose the best decision in the execution phase based on offline optimizations or offline-learned experience. Typically, execution time characteristics are gathered from profiling data. In the following, we introduce one approach as described in [17, 18] more closely.

2.1.1 Profiling Phase

The first phase aims at parameterizing a latency and an energy model for a given application or a task, which is dependent on one or multiple workload indicators (e.g., the number of features in an image) and the configuration (e.g., voltage/frequency and the number of cores) to execute this workload item, respectively task. The approach in [17, 18] considers a class of video (streaming) applications, in particular an object-detection application, in which the workload of some tasks is dependent on the number of input features i . Apart from the number of features, the execution latency L also depends on the voltage/frequency setting m and the number of cores n used for the calculation.

Let $L(1, 1, m_{\max})$ denote the latency for processing one feature on one core in the highest voltage and frequency mode m_{\max} . $L(1, 1, m_{\max})$ may be determined by simulatively determining the execution latency of the execution per image for a representative set of input images. Subsequently, the latency estimate per feature is determined for each image by dividing its latency by the number of features i in that image. Alternatively, the latency could be determined by applying worst-case timing analysis.

2.1.2 Pareto-Front Determination Phase

After determining $L(1, 1, m_{\max})$, a model for estimating the latency $L(i, n, m)$ for processing i features in an image when employing n cores and running in voltage/frequency setting m is derived or learned, resulting in a mathematical characterization of the latency in dependence of workload i and processor setting n and m , e.g.:

$$L(i, n, m) = L(1, 1, m_{\max}) \cdot \frac{i}{n \cdot \text{eff}(n)} \cdot \frac{f(m_{\max})}{f(m)} \quad (1)$$

In Eq. (1), $\text{eff}(n)$ denotes the parallel efficiency in dependence of the number of cores n employed for the computation with $\text{eff}(n) = 1$ in the best case and $f(m)$ being the frequency of setting m . In our experiments described in Section 3, we consider $\text{eff}(n) = 1$. Then, based on a given latency deadline \bar{L} , a design space exploration (DSE) can then be performed [18] to determine those settings n, m that enable to process the workload i within the deadline \bar{L} while consuming the least amount of energy.

2.1.3 Energy-Minimized Timing Enforcement

Finally, the Pareto-optimal settings can be stored in a table or implemented by an automaton that just selects at run-time the energy minimal setting in characterized ranges of inputs i . Before each execution, based on i , the pre-determined energy-minimal setting $\langle n, m \rangle$ is activated as action. Obviously, this hybrid technique must be adjusted to each application and each architecture individually. However, as we will show in our experiments, if the characterization is *safe*, then we may guarantee deadlines to be enforced by 100%. This means that even hard deadlines can be safely enforced.

2.2 Feedback-Based Approaches

These approaches utilize a latency feedback to decide the next configuration (i.e., $\langle n, m \rangle$) to apply. The most prominent approaches are based on control-theory or finite state machines (FSMs).

2.2.1 Using Control Theory

Using control theory for designing adaptive systems [1] has the advantage of formally guaranteeing the properties (e.g., stability) of systems with unknown workloads [19]. In the following, we describe one concrete approach [4] for subsequent evaluation in more detail. Figure 1 illustrates the overall control mechanism.

Control Signal as an Abstraction for Latency

Control-theory based approaches consider latency as a variable that is controlled, where the controller utilizes Eq. (2) to model the relation between the latency $L(t)$ at iteration t (respectively, discrete time index in the following), and the control signal $s(t-1)$ [4, 11].

$$L(t) = \frac{1}{b(t) \cdot s(t-1)} \quad (2)$$

In Eq. (2), the *base signal* $b(t)$ is a time-varying parameter that forms an abstraction of $L(i, n_{\min}, m_{\min})$, which is the latency of executing the application's workload (i.e., number of features i) at iteration t in the lowest configuration (i.e., $\langle n_{\min}, m_{\min} \rangle$). The base signal is computed using Eq. (3).

$$b(t) = \frac{1}{i(t) \cdot L(1, n_{\min}, m_{\min})} \quad (3)$$

Here, $i(t)$ denotes the number of features i observed at iteration t and $L(1, n_{\min}, m_{\min})$ is the latency of executing one unit of workload (i.e., one feature or $i = 1$) at iteration t in the lowest configuration (i.e., $\langle n_{\min}, m_{\min} \rangle$). As $L(i, n_{\min}, m_{\min})$ generally cannot be known until run time, it is estimated using a Kalman filter [4], or in our use case and later experiments, we use a profiling phase to determine $L(1, n_{\min}, m_{\min})$ for its estimation.

Configuration Mapping

A *configuration mapping* is used in a later energy optimization step to determine the best configurations $\langle n, m \rangle$ for the computed control signal $s(t)$ [4]. This is done by creating a table with an index $s_{n,m}$ that delivers a corresponding configuration $\langle n, m \rangle$ with $s_{n,m}$ being defined as the speedup when executing one unit of workload (i.e., one feature or $i = 1$) in the configuration $\langle n, m \rangle$ over the case when executing this workload unit in the lowest configuration $\langle n_{\min}, m_{\min} \rangle$:

$$s_{n,m} = \frac{L(1, n_{\min}, m_{\min})}{L(1, n, m)} \quad (4)$$

Here, $L(1, n, m)$ denotes the latency of executing one unit of workload (i.e., one feature or $i = 1$) at iteration t in the configuration $\langle n, m \rangle$.

Computing the Control Signal

At iteration t , the controller uses Eq. (5) to calculate the error between the latency in the last execution $L(t-1)$ and the latency goal \bar{L} [4]. In case of a positive error value, the deadline has therefore been missed (i.e., $\bar{L} < L(t-1)$).

$$e(t) = \frac{1}{\bar{L}} - \frac{1}{L(t-1)} \quad (5)$$

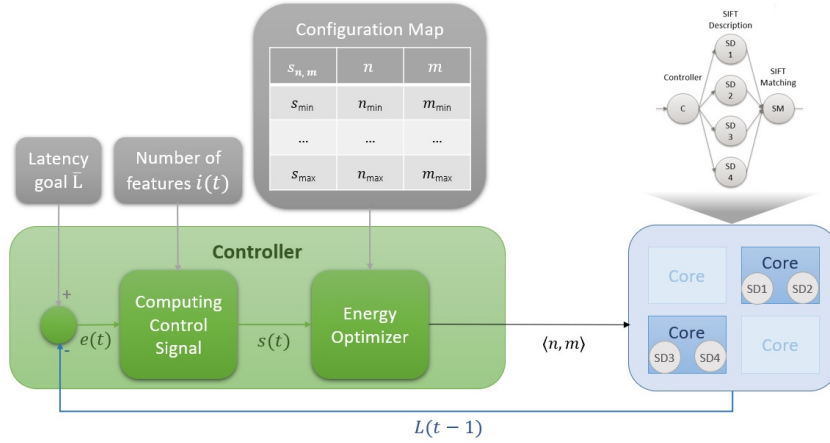
The control signal $s(t)$ is then computed using Eq. (6) [4]:

$$s(t) = s(t-1) + (1-p) \cdot \frac{e(t)}{b(t)} \quad (6)$$

Finally, the control signal $s(t)$ is used by the optimizer to find the best configuration (i.e., $\langle n, m \rangle$) that is needed to be applied in the next iteration t , based on the latency error $e(t)$, the base signal $b(t)$, and the previous control signal $s(t-1)$. The base signal $b(t)$ is computed using Eq. (3) after determining the current workload (i.e., number of features $i(t)$). The *pole* p is a user-specified parameter that lies between 0 and 1. A small value of p increases the importance of the error to the resultant control signal, whereas a large p increases the controller resistance to fast-changing workloads.

Energy Optimization

The optimizer uses the configuration mapping, explained above, for transforming the calculated control signal $s(t)$ into the best configuration (i.e., $\langle n, m \rangle$) to meet a given deadline \bar{L} , while minimizing the amount of consumed energy. Kim et al. [5] claim that there must be an optimal solution that has at most two configurations that have to be scheduled within the time interval between iteration t and $t+1$. These two configurations are computed using the algorithm which is detailed in [4].



■ **Figure 1** The controller uses the error $e(t)$ and the base signal $b(t)$ (based on the number of features $i(t)$) to compute the control signal $s(t)$. The optimizer looks up the configuration mapping for deciding the next actions (i.e., $\langle n, m \rangle$) based on the computed control signal $s(t)$; the value of $s(t)$ is compared with the values $s_{n,m}$ of the configuration mappings to select the configurations under which to operate the system in the time interval between iteration t and $t + 1$ ¹.

2.2.2 Using Finite state machines (FSMs)

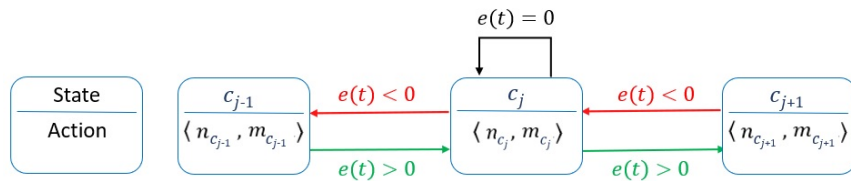
An FSM describes the system's behavior using states, transitions, events, and actions. An FSM is composed of a number of states. It starts in an initial state. Each state outputs actions to perform the desired control activities in case of a Moore machine. Triggered by events (input), it transitions to a next state based on the input and the current state.

FSMs are heavily used for modeling the power states of modern processor architectures including dynamic power management [20]. FSMs have also been proposed for adjusting the processor power depending on a utilization feedback. For instance, the authors of [2] propose an FSM-based controller that uses the processor utilization as a feedback to choose the best configuration (i.e., $\langle n, m \rangle$). Here, the FSM transitions between the states based on utilization thresholds.

A possible realization of an FSM enforcing latency while minimizing energy based on feedback is depicted in Figure 2. The error $e(t - 1)$ is computed similar to Eq. (5) from the latency feedback $L(t - 1)$. This approach neither includes an offline stage nor utilizes a control signal. Instead, it uses a power-ascending list of configurations so that the configuration $\langle n_{c_j}, m_{c_j} \rangle$ associated with state c_j only should have a higher power consumption than that of state c_{j-1} . With a number of N available configurations, the FSM consists of states c_j with $1 \leq j \leq N$. Based on the error $e(t - 1)$, there are three possibilities to define the FSM transitions:

1. The latency error is positive $e(t - 1) > 0$: The deadline has been missed and the FSM responds by switching to the next state with higher power by incrementing the configuration level.
2. There is no latency error $e(t - 1) = 0$: The deadline has been met precisely and the FSM stays in the same state with the same configuration level.
3. The latency error is negative $e(t - 1) < 0$: The application executes faster than needed and the FSM responds by switching to the next state with lower power to decrement the configuration level.

¹ In [4], it is explained in detail how a two-step sequence of two $s_{n,m}$ configurations is activated and optimized so to save energy.



■ **Figure 2** A feedback-based FSM approach, that transitions between adjacent states at each iteration t , based on the feedback $L(t-1)$ and a power-ascending list of configurations.

For the construction of such enforcement FSM, many extensions are possible, e.g., transition conditions containing multiple events (not only one like $e(t)$), possibly also including environmental conditions such as temperature. Moreover, instead of just transiting to one lower, resp. one higher power state, one could transit to non-neighbor states depending on the absolute deviation $e(t)$.

3 Evaluation

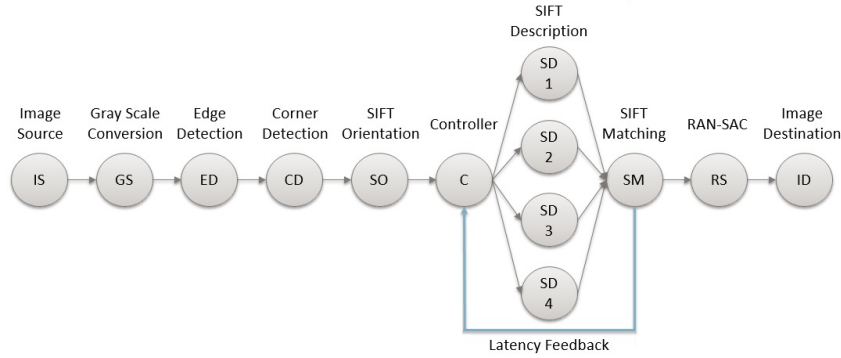
3.1 InvadeSIM simulator

For the following evaluations and comparisons, a simulation framework called InvadeSIM, a many-core simulator for parallel applications [15] is used. InvadeSIM allows to specify symmetric homogeneous, as well as tile-based asymmetric many-core architectures within one framework. It performs a discrete-event simulation of applications mapped to a given architecture model including the processor cores, memory access overheads and communication latencies, e.g., on a network-on-chip. In addition, it provides an emulation for the runtime system and a timing as well as a power model for each core type of a heterogeneous many-core architecture. The discrete-event simulation enables the timed simulation of the different parallel tasks in execution. From a power configuration and the execution times, the energy consumption of application executions can be monitored as well. Using an ActorX10 object-oriented programming library [14], applications are modeled by a graph of actors and then mapped and executed on a modeled multi-core platform.

3.2 Object Detection Application

For our evaluations and comparisons, we introduce an object detection application as shown in Figure 3. It belongs to the class of image processing applications that performs a pipelined processing of input image streams. The job of the object detection algorithm is to detect a given object in each image frame by applying a SIFT feature matching algorithm.

The application consists of an actor chain. Each actor processes one input image at a time. The image source (IS) actor reads in the input images periodically at a constant rate, then follows the gray-scale conversion (GS) actor, and after that the edge detection (ED) and the corner detection (CD) actors to determine respectively edges and corners in an image. After that the SIFT orientation (SO) actor achieves invariance to image rotation. The four SIFT description actors SD_1 to SD_4 extract the features in an image. They can be executed in parallel on $n = 4$ cores, after partitioning the number of features i of a given image evenly into each actor.



■ **Figure 3** Object detection algorithm implemented as a graph of actors for pipelined processing of streams of images.

3.3 Target Architecture and Deadline Model

For the following experiments, let each of the periodic execution of each SD actor be completed within a soft deadline of $\bar{L} = 80$ ms. For the enforcement of this local deadline, the execution power mode m (voltage/frequency) of the SD actors' cores through Dynamic Voltage and Frequency Scaling (DVFS) is used and we assume, a maximum of $n = 4$ cores can be activated in each of $m = 20$ different power modes. However, during the execution of an image, we assume all cores run in the same power mode m , thus resulting in a configuration $\langle n, m \rangle$. According to Figure 3, the SD actor also provides a feedback of the latency which can be used by the controller to properly determine the configuration $\langle n, m \rangle$ to be used to execute the next frame, resp. iteration. Upon each execution, the output of each SD actor is then provided to the SIFT matching (SM) actor to detect common features between the given object to be found and the current input image. Then, the RAN-SAC (RS) actor calculates the transformation between both images based on the matched features. The image is finally sent out by an image destination (ID) actor.

3.4 Application Management Techniques

In the following, we evaluate four techniques for latency-aware energy optimization:

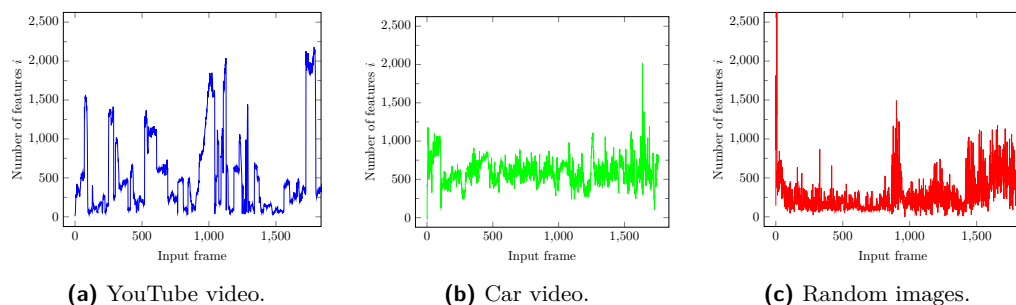
1. Race-to-idle: Executing always in the highest configuration level $\langle n_{\max}, m_{\max} \rangle$.
2. Application-specific: Finding operating points offline by analyzing sequences of input images beforehand, see Section 2.1. At run-time, the Pareto-optimal configuration to enforce the given deadline is retrieved from a table based on the characterized input (feature number i) [17, 18].
3. Control-theoretical: Computing a control signal that translates into a specific configurations $\langle n, m \rangle$ based on latency feedback $L(t-1)$ [4].
4. FSM-based: Using the simple FSM from Figure 2 that is based on latency feedback $L(t-1)$ to transition between neighbor system configurations $\langle n, m \rangle$.

3.5 Workload Types

The following experiments have been performed by applying the following workloads to the object detection application with latency-enforced SD actor:

- A 5-minute YouTube video from national geographic.
- A 40-minutes video of front camera in a car driving through city roads.
- A sequence of 1,750 images with random contents.

Figure 4 shows the number of features i for each image of the sequences.



■ **Figure 4** Distribution of number of features i per frame in the analyzed videos.

3.6 Results

We run the application for each combination of application management technique and input sequence. Each controller behavior has been implemented by the controller actor in Figure 3. The number of available configurations is $N = n \cdot m$ with $n = 4$ and $m = 20$. As latency bound \bar{L} of the enforced SD actor, we chose $\bar{L} = 80 \text{ ms}$. For the control-theoretic approach, the pole was chosen at $p = 0.5$ (changing the pole value did not have a noticeable impact on its results).

In the following, we analyze the performance of the four approaches in terms of number of deadline misses and in terms of energy consumption.

3.6.1 Deadline Misses

Mean Absolute Percentage Error (MAPE) is a standard metric in controllers [4], and we use it to measure number of deadline misses. We can compute MAPE for an application with K iterations (number of frames in our test applications) using Eq. (7):

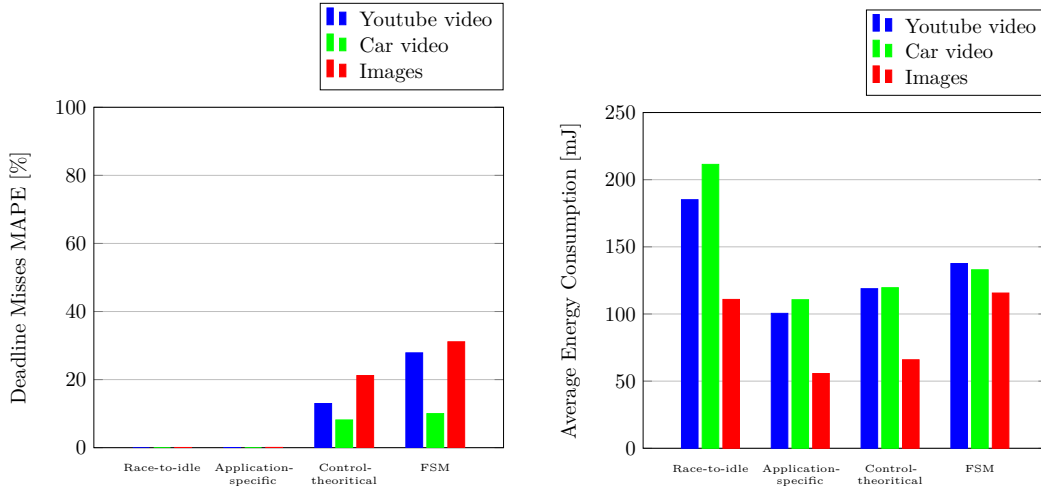
$$MAPE = 100\% \cdot \frac{1}{K} \sum_{i=1}^K \begin{cases} L(t-1) > \bar{L} : \frac{L(t-1) - \bar{L}}{\bar{L}} \\ L(t-1) \leq \bar{L} : 0 \end{cases} \quad (7)$$

Figure 5a shows the number of deadline misses experienced for each of the four evaluated approaches. Race-to-idle and the application-specific approach almost never miss any deadline as they execute in the fastest possible way, respectively the slowest required speed to meet the deadline. Only in a few cases where the deadline cannot be met at all even in the fastest possible configuration, \bar{L} is (necessarily) exceeded. However, evidently, the feedback-based approaches miss more deadlines, especially the FSM-based approach, which obviously by its simple construction provides only a stepwise and thus slow convergence towards a feasible level of configuration $\langle n, m \rangle$, after receiving the latency feedback $L(t-1)$.

3.6.2 Energy Consumption

Figure 5b shows the evaluated average overall energy consumption per image of the SD actor. Race-to-idle consumes the largest amount of energy, because it executes in the fastest possible way. The application-specific approach consumes a significantly less amount of energy, due to the fact that it knows exactly which actions are needed to meet the deadline for all possible

input images. For the feedback-based approaches, the control-theoretical approach consumes slightly more energy than the optimal (i.e., the application-specific approach). On the other hand, the simple FSM-based approach consumes more energy than the application-specific and the control-theoretical approach due to its simplicity in construction. Still, it consumes less energy than the race-to-idle strategy.



(a) Percentage of deadline misses of each approach.

(b) Energy consumption of each approach.

■ **Figure 5** Evaluation results of the race-to-idle, application-specific and feedback-based approaches using the three types of image input sequences and a set of $N = 80$ configurations in terms of deadline misses (a) and consumed energy (b).

3.6.3 The Effect of Available Configurations

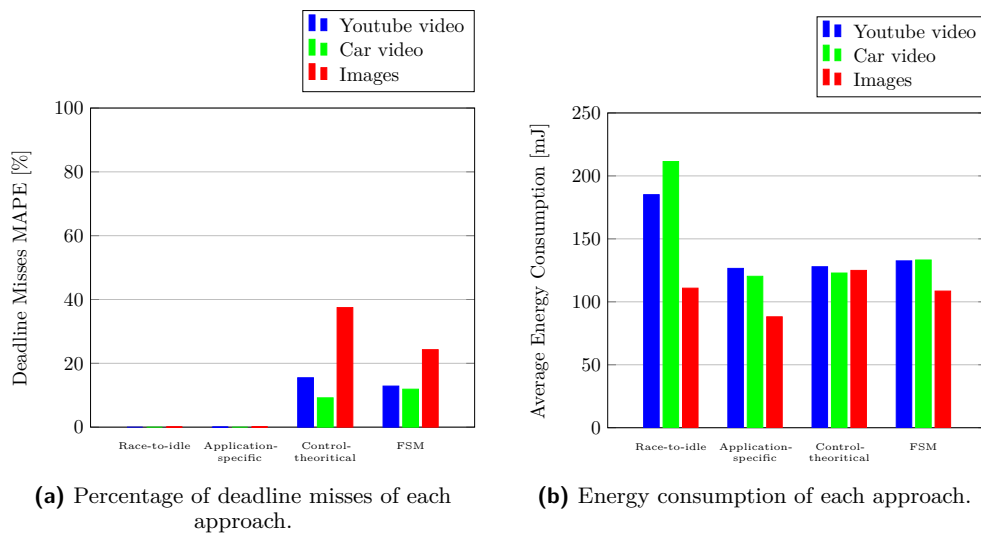
In a further evaluation, we fixed the number of cores to $n = 4$, resulting into only $N = 20$ configurations (i.e., 20 voltage/frequency settings), keeping race-to-idle unaffected.

We notice from the results in Figure 6 an increase in the amount of energy consumption for the application-specific and the control-theoretical approach. For the latency results, the FSM-based approach achieves less deadline misses compared to the case of $N = 80$, because now, the transitions in the FSM are more effective in the sense of faster converging to a feasible configuration level $\langle n, m \rangle$.

4 Conclusion and Future Work

In this paper, we evaluated and compared multiple application-specific and feedback-based categories for latency-aware energy optimization. We evaluated the approaches on a many-core simulator and found out that the application-specific approach respects the latency goal whenever possible while consuming the lowest amount of energy consumption. For the feedback-based category, the feedback-based approaches can achieve energy savings comparable to the application-specific approach, but both missing deadlines more often.

For future work, we aim to focus on FSM-based approaches further, as they are similar to control-theoretical approaches in providing a sound mathematical formalism with opportunities of formally proving requirements on non-functional program properties, e.g., by state reachability and the use of temporal logic to express complex requirements. On



■ **Figure 6** Evaluation results of the race-to-idle, application-specific and feedback-based approaches using three types of image input sequences and a set of $N = 20$ configurations in terms of deadline misses (a) and consumed energy (b).

the other hand, control-theoretic approaches, although theoretically sound and known for their strength in being able to mathematically prove properties such as the stability and robustness of a feedback-based system, are often based on assumptions of linearity of either controller or the many-core system under control, e.g., based on z-transform descriptions and the analysis of poles of related closed-loop transfer functions. However, in multi-core systems, performance such as the speedup hardly scales linearly with the number of cores for most workloads and applications. Therefore, non-linear control techniques would need to be applied.

References

- 1 Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. *Feedback control of computing systems*. John Wiley & Sons, 2004.
- 2 Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the 2007 international symposium on Low power electronics and design (ISLPED'07)*, pages 38–43. IEEE, 2007.
- 3 Henry Hoffmann, Martina Maggio, Marco D Santambrogio, Alberto Leva, and Anant Agarwal. A generalized software framework for accurate and efficient management of performance goals. In *2013 Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 1–10. IEEE, 2013.
- 4 Connor Imes, David HK Kim, Martina Maggio, and Henry Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 75–86. IEEE, 2015.
- 5 David HK Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *2015 IEEE 3rd international conference on cyber-physical systems, networks, and applications*, pages 78–85. IEEE, 2015.
- 6 Zhiquan Lai, King Tin Lam, Cho-Li Wang, and Jinshu Su. Latency-aware DVFS for efficient power state transitions on many-core architectures. *The Journal of Supercomputing*, 71(7):2720–2747, 2015.

- 7 Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Automated control of multiple software goals using multiple actuators. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pages 373–384, 2017.
- 8 S. K. Mandal, U. Y. Ogras, J. Rao Doppa, R. Z. Ayoub, M. Kishinevsky, and P. P. Pande. Online adaptive learning for runtime resource management of heterogeneous socs. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi:10.1109/DAC18072.2020.9218604.
- 9 Sumit K. Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Umit Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 25(3):1–26, 2020.
- 10 Asit K Mishra, Shekhar Srikantiah, Mahmut Kandemir, and Chita R Das. Cpm in cmpps: Coordinated power management in chip-multiprocessors. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2010.
- 11 Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. *ACM SIGPLAN Notices*, 53(2):184–198, 2018.
- 12 Amir-Mohammad Rahmani, Mohammad-Hashem Haghbayan, Anil Kanduri, Awet Yemane Weldezion, Pasi Liljeberg, Juha Plosila, Axel Jantsch, and Hannu Tenhunen. Dynamic power management for many-core platforms in the dark silicon era: A multi-objective control approach. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 219–224. IEEE, 2015.
- 13 Karthik Rao, Jun Wang, Sudhakar Yalamanchili, Yorai Wardi, and Ye Handong. Application-specific performance-aware energy optimization on android mobile devices. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 169–180. IEEE, 2017.
- 14 Sascha Roloff, Frank Hannig, and Jürgen Teich. ActorX10 and run-time application embedding. In *Modeling and Simulation of Invasive Applications and Architectures*, pages 129–164. Springer, 2019.
- 15 Sascha Roloff, Frank Hannig, and Jürgen Teich. *Modeling and Simulation of Invasive Applications and Architectures*. Springer, 2019.
- 16 Stepan Shevtsov and Danny Weyns. Keep it simplex: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 229–241, 2016.
- 17 Jürgen Teich, Behnaz Pourmohseni, Oliver Keszocze, Jan Spieck, and Stefan Wildermann. Run-time enforcement of non-functional application requirements in heterogeneous many-core systems. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 629–636. IEEE, 2020.
- 18 Jürgen Teich, Pouya Mahmoody, Behnaz Pourmohseni, Sascha Roloff, Wolfgang Schröder-Preikschat, and Stefan Wildermann. Run-Time Enforcement of Non-functional Program Properties on MPSoCs. In Jian-Jia Chen, editor, *A Journey of Embedded and Cyber-Physical Systems*. Springer, 2020. doi:10.1007/978-3-030-47487-4.
- 19 Danny Weyns, M Usman Iftikhar, Didac Gil De La Iglesia, and Tanvir Ahmad. A survey of formal methods in self-adaptive systems. In *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*, pages 67–79, 2012.
- 20 Yang Xu, Rafael Rosales, Bo Wang, Martin Streubühr, Ralph Hasholzner, Christian Haubelt, and Jürgen Teich. A very fast and quasi-accurate power-state-based system-level power modeling methodology. In Andreas Herkersdorf, Kay Römer, and Uwe Brinkschulte, editors, *Architecture of Computing Systems - ARCS 2012 - 25th International Conference, Munich, Germany, February 28 - March 2, 2012. Proceedings*, volume 7179 of *Lecture Notes in Computer Science*, pages 37–49. Springer, 2012. doi:10.1007/978-3-642-28293-5_4.

EDF Scheduling and Minimal-Overlap Shortest-Path Routing for Real-Time TSCH Networks

Miguel Gutiérrez Gaitán¹ 

CISTER – Research Centre in Real-Time and Embedded Computing Systems, Porto, Portugal
Faculty of Engineering, University of Porto, Portugal
Institute of Engineering, Polytechnic Institute of Porto, Portugal
Facultad de Ingeniería, Universidad Andrés Bello, Santiago de Chile, Chile
mgg@fe.up.pt

Luís Almeida 

CISTER – Research Centre in Real-Time and Embedded Computing Systems, Porto, Portugal
Faculty of Engineering, University of Porto, Portugal
lda@fe.up.pt

Pedro Miguel Santos 

CISTER – Research Centre in Real-Time and Embedded Computing Systems, Porto, Portugal
Institute of Engineering, Polytechnic Institute of Porto, Portugal
pss@isep.ipp.pt

Patrick Meumeu Yomsi 

CISTER – Research Centre in Real-Time and Embedded Computing Systems, Porto, Portugal
Institute of Engineering, Polytechnic Institute of Porto, Portugal
pmy@isep.ipp.pt

Abstract

With the scope of Industry 4.0 and the Industrial Internet of Things (IIoT), wireless technologies have gained momentum in the industrial realm. Wireless standards such as WirelessHART, ISA100.11a, IEEE 802.15.4e and 6TiSCH are among the most popular, given their suitability to support real-time data traffic in wireless sensor and actuator networks (WSAN). Theoretical and empirical studies have covered prioritized packet scheduling in extenso, but only little has been done concerning methods that enhance and/or guarantee real-time performance based on routing decisions. In this work, we propose a greedy heuristic to reduce overlap in shortest-path routing for WSANs with packet transmissions scheduled under the earliest-deadline-first (EDF) policy. We evaluated our approach under varying network configurations and observed remarkable dominance in terms of the number of overlaps, transmission conflicts, and schedulability, regardless of the network workload and connectivity. We further observe that well-known graph network parameters, e.g., vertex degree, density, betweenness centrality, etc., have a special influence on the path overlaps, and thus provide useful insights to improve the real-time performance of the network.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Networks → Network algorithms; Networks → Data path algorithms

Keywords and phrases Real-time communication, Routing, Scheduling, TDMA, Wireless networks

Digital Object Identifier 10.4230/OASICS.NG-RES.2021.2

Funding This work was partially supported by the project Safe Cities - Inovação para Construir Cidades Seguras, ref. POCI-01-0247-FEDER-041435, co-funded by the European Regional Development Fund (ERDF), through the Operational Programme for Competitiveness and Internationalization (COMPETE 2020); also by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDB/04234/2020).

¹ Corresponding author.



1 Introduction

Wireless sensor and actuator networks (WSAN) play nowadays an important role in industrial facilities. Wireless radio links, in general, bring the flexibility and scalability that wireline infrastructure lacks, but often with relatively lower bandwidth and reliability [4]. Yet, for many applications based on sensor and actuator networks, e.g., for real-time monitoring or even audio streaming [10], low data rate wireless technologies (up to 250 Kbps) are sufficient to satisfy typical bandwidth requirements. Similarly, the reliability of standards like WirelessHART, ISA100.11a, IEEE802.15.4e and 6TiSCH, based on IEEE 802.15.4-PHY, increased to levels that are, in many cases, compatible to wired networks [18].

Time-synchronized channel hopping (TSCH) is among the most popular standards in the scope of WSAN to support real-time data traffic. Salient features, such as time-division multiple-access (TDMA), centralized scheduling and frequency diversity, have gradually underpinned its adoption in a number of application domains, from factory automation and process control [9] to vehicles [16], paving the way for the Industrial Internet of Things (IIoT) and Industry 4.0 [15].

In these domains, real-time communication is essential to ensure satisfactory (and deterministic) performance. The predictable/analyzable (time-slotted) channel access of TSCH is appropriate for that purpose which, coupled with proper (typically centralized) scheduling and routing algorithms, can provide safe operational bounds for worst-case end-to-end delays and schedulability. Several research efforts have pursued real-time communication in TSCH networks, but mostly focusing on packet scheduling. Routing, in the other hand, is often assumed as standard, e.g., using the shortest-path algorithm, leading to sub-optimal real-time performance.

In this work, we deal with the so-called *real-time wireless routing* [19] for TSCH networks, whose primary goal is to enhance and/or guarantee the real-time properties of the network based on routing decisions. In this respect, Wu et al. [19] proposed a *conflict-aware routing* method for WirelessHART networks with packet transmissions scheduled using a fixed-priority policy. We tackle alike foundational questions from this work, but for TSCH WSANs under the earliest-deadline-first (EDF) scheduler, instead. We propose a *minimal-overlap shortest-path* routing based on a greedy heuristic driven by reducing path overlaps among network flows. We show, by leveraging on prior work on schedulability analysis, that our method considerably improves the network schedulability when compared to the conventional (hop-count) shortest-path method.

2 Related Work & Contribution

Theoretical and empirical studies for modelling and assessing the real-time performance of TSCH-like networks have been discussed in recent literature, e.g., [13, 6, 5, 8, 1, 3, 14], usually having as the main focus priority-based packet scheduling algorithms. The span of analytical works includes the design of methods based on response-time analysis [13], supply/demand-based tests [6, 5], network calculus [8], etc., often deriving theoretical/empirical bounds attempting to guarantee worst-case real-time network performance. Both fixed-priority and dynamic-priority schedulers have been covered, most of the times assuming a standard behaviour for the rest of network features, e.g., routing, channel assignment, etc.

While for routing there are many works available in the literature [12] addressing TSCH networks, only a few of them fit into the class of *real-time wireless routing* [19], i.e., tailored routing methods aiming to enhance and/or guarantee the real-time performance of wireless networks. Wu et al. [19] made a step ahead in this direction by proposing a *conflict-aware*

real-time routing for WirelessHART networks under a fixed-priority policy, but they did not address dynamic-priority schedulers. Their work leverages on a prior delay analysis for TSCH-like networks, which derives in part from the real-time CPU scheduling theory [7].

We highlight the importance of this prior analysis in our work, allowing to split up the end-to-end delay analysis into two components: (i) the effect of channel contention, and (ii) the effect of wireless transmission conflicts. The former is conveniently mapped to the multiprocessor contention concept, by assuming the number of cores as equal to the number of (radio) channels in TSCH networks. The second is specific to wireless transmission scheduling, and model the restriction of half-duplex transceivers to transmit/receive alternately, a challenging condition under mesh network topologies, which has a significant impact on end-to-end delays and schedulability.

As in [19], we focus on the latter factor to improve routing decisions, i.e., the effect of the transmission conflicts on real-time performance, but we target transmissions scheduled under EDF. Moreover, distinctly to [19], we do not generate routes one-by-one until they become schedulable; instead, we provide a set of paths with minimal (node-) overlaps between flows, and then, we test the overall schedulability. We draw attention to the effectiveness of our approach against to the more common (hop-count) shortest path method, as well as in terms of the bandwidth utilization benefits brought by EDF in comparison to fixed-priority schedulers. We note that to the best of our knowledge, this is the first joint EDF scheduling and *real-time wireless routing* framework for TSCH networks.

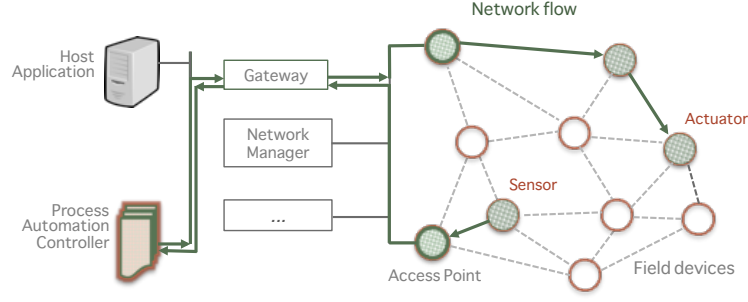
3 System Model: TSCH-based network and EDF scheduling

We consider a WSN as the one represented in Figure 1. The network consists of a finite number of $N \in \mathbb{N}$ nodes, including one gateway, multiple access points (APs), and several field devices (i.e., sensors and actuators). The field devices are wirelessly connected to the APs forming a mesh network topology, and each of them is equipped with half-duplex omnidirectional radio transceivers. The APs are directly linked to the gateway, which, in turn, enables bidirectional communication between the field devices and other entities outside the network, e.g., the network manager, process controller, host application, etc. The network manager is a software module (typically running on the gateway) which collects network topological information and is responsible for both scheduling and routing functions.

We assume the network is TSCH-based, i.e., relies on an IEEE 802.15.4 compatible physical layer, and uses a centralized multi-channel TDMA protocol with global synchronization. The multi-channel feature enables concurrent per-slot (but not per-channel) transmissions based on a channel hopping technique over a number of m active (i.e., not-blacklisted) radio channels, with $1 \leq m \leq 16 \in \mathbb{N}$. The length of the time slots is fixed (~ 10 ms), and corresponds to a (dedicated) time interval to allocate a single packet transmission, a maximum number of $w - 1$ retransmissions (with $w \in \mathbb{N}$), and their corresponding acknowledgements.

Sensor nodes periodically transmit data through the network to external entities, e.g., to a remote controller (located at the gateway's node position), which, in turn, deliver control commands to the actuators (see Fig. 1). We consider transmissions occur in per-slot/per-hop basis, following predefined multi-hop routes between the sensors and the gateway (uplink), and between the gateway and the actuators (downlink), both with stringent timing delivery constraints. As for the sake of simplicity, we consider the routes are set under *source routing*, i.e., using pre-defined single routes for both uplink and downlink.

Without loss of generality, we also assume the topology maintenance is a native in-built centralized service or that can be further implemented, e.g., as in [17].



■ **Figure 1** Pictorial representation of an industrial WSN.

3.1 Network model

Given the above features, the network is modelled as a graph $G = (V, E)$, where V represents the set of vertices (or nodes), and E the set of edges (or links) between those nodes. We assume the graph is undirected and connected, but not complete, i.e., there is a path between any pair of nodes in the network, but not a link between every node pair. The total number of nodes N corresponds to the total number of vertices $|V|$, i.e., $N = |V|$, where one node acts as a gateway, and the rest $N - 1$ nodes correspond to the multiple APs and the several field devices. We further assume the gateway is the node with the highest *betweenness centrality*, i.e., the node if being removed, has the greatest impact on the overall network connectivity.

We consider a subset $n \in \mathbb{N}$ of the field devices are used to generate data (e.g., sensor measurements), and the rest $N - n - 1$ nodes act as relay. Note that when not transmitting their own data, the n transmitter field devices can act as relay too.

3.2 Flow model

We denote $F \stackrel{\text{def}}{=} \{f_1, f_2, \dots, f_n\}$ the set of n real-time network flows to be transmitted from source to destination by following an EDF policy. Each f_i represents a periodic time-constrained end-to-end communication flow, characterized by a 4-tuple (C_i, D_i, T_i, ϕ_i) . C_i denotes the effective transmission time between source and destination, T_i the period (or the sampling rate of sensors), D_i the relative deadline, and ϕ_i the routing path. These parameters are given with the interpretation that each flow f_i releases a potentially infinite number of transmissions. The γ^{th} instance of those transmissions, with $\gamma \in \mathbb{N}$, is denoted as $f_{i,\gamma}$, and is released at the time $r_{i,\gamma}$, such that $r_{i,\gamma+1} - r_{i,\gamma} \stackrel{\text{def}}{=} T_i$. Then, in accordance with the EDF policy, $f_{i,\gamma}$ is constrained to reach its destination before its absolute deadline, i.e., $d_{i,\gamma} \stackrel{\text{def}}{=} r_{i,\gamma} + D_i$. We assume this is a constrained deadline model, i.e., $D_i \leq T_i$, thus allowing only a single transmission of flow f_i at any time slot.

Note that here C_i is interpreted as the time required by flow f_i to be completely transmitted from source to destination, but when it does not suffer whatsoever interference from other flows. We thus assume C_i can be computed as $C_i = \zeta_i \times w$ (slots), where ζ_i is total number of links in the route path ϕ_i , and w is the number of transmission slots assigned to a flow in each link, including retransmissions. We adopt w fixed (as in WirelessHART) for all the links, thus C_i being exclusively dependent on topology and routing dynamics.

3.2.1 Supply/demand-based schedulability analysis for EDF

For the sake of completeness, we briefly revisit our prior work on schedulability analysis for WSANs [6]. Particularly, we leverage on the so-called *forced-forward demand-bound function* (FF-DBF) [2] when applied to the WSANs domain, which is a state-of-the-art supply/demand-bound schedulability assessment for TSCH-like WSANs under EDF [5].

We reproduce here the formal expression for the schedulability test, defined as the relationship between the *supply-bound function* (SBF) [20], i.e., the minimal transmission capacity offered by a network with m channels, and the FF-DBF, i.e., the upper-bound on the maximum possible demand for a set of real-time network flows $F = \{f_1, f_2, \dots, f_n\}$ (with $n \in \mathbb{N}$) when evaluated over a given time interval of length ℓ

$$\sum_{i=1}^n \text{FF-DBF}(f_i, \ell) \leq \text{sbf}(\ell), \forall \ell \geq 0 \quad (1)$$

where the SBF is formally defined as follows

$$\text{sbf}(0) = 0 \wedge \forall \ell, k \geq 0 : \text{sbf}(\ell + k) - \text{sbf}(\ell) \leq m \times k \quad (2)$$

and the FF-DBF (for TSCH-based WSANs) is defined as

$$\frac{1}{m} \sum_{i=1}^n \text{FF-DBF}^{ch}(f_i, \ell) + \sum_{i,j=1}^n \left(\Delta_{i,j} \max \left\{ \left\lceil \frac{\ell}{T_i} \right\rceil, \left\lceil \frac{\ell}{T_j} \right\rceil \right\} \right) \quad (3)$$

where $\Delta_{i,j}$ denotes the transmission conflicts delay (due to path overlaps) between any pair of flows f_i and $f_j \in F$, and T_i and T_j the respective transmission periods of these flows.

► **Note.** On (1), unlike to the common understanding on multiprocessors [2], the FF-DBF notion refers to the upper bound on network demand due to the contribution of two components: (i) channel contention, equivalent to the (core) contention concept on multi-core platforms, and (ii) transmission conflicts, an abstraction specific to wireless transmission scheduling. On (3), these two components are formally dissociate as a summation. On the left side, the expression denoted as FF-DBF^{ch} corresponds to the channel contention contribution, which is equivalent to the expression on multiprocessors, but here it models the restriction imposed to network flows to be simultaneously scheduled on different channels (see [6] for the complete expression of FF-DBF^{ch}). On the right side, the expression designates the contribution of transmission conflicts, which represents the delay experienced due to multiple flows encountering on a common node. We also note that this is a key aspect for the motivation and further understanding of the solution later proposed in the present work.

4 Problem Formulation

Given the network and flow models presented in Section 3, we consider the problem of finding the optimal set of flow paths $\Phi_{opt} \stackrel{\text{def}}{=} \{\phi_1^{opt}, \phi_2^{opt}, \dots, \phi_n^{opt}\}$ that minimizes the *overall number of path overlaps* between any pair of flows in the network.

► **Definition 1.** We denote as Ω the overall number of path overlaps *between any pair of flows in the set* $F \stackrel{\text{def}}{=} \{f_1, f_2, \dots, f_n\}$. Particularly, Ω is the sum of all the individual node overlaps $\delta_{i,j}$ between the routes of any pair of flows f_i and f_j in the set F , where $i, j \in [1, n] \wedge i \neq j$.

► **Definition 2.** We denote as $F_0 \stackrel{\text{def}}{=} \{f_1^0, f_2^0, \dots, f_n^0\}$ the original set of flows in the network which set of flow paths $\Phi_0 \stackrel{\text{def}}{=} \{\phi_1^0, \phi_2^0, \dots, \phi_n^0\}$ is obtained using a conventional (hop-count) shortest-path algorithm, ergo Ω_0 is the respective overall number of path overlaps for Φ_0 .

► **Definition 3.** We denote as $F_k \stackrel{\text{def}}{=} \{f_1^k, f_2^k, \dots, f_n^k\}$ (with $k \in \mathbb{N}$) the k^{th} variation of the set of flows F_0 when a sub-optimal set of routes $\Phi_k \stackrel{\text{def}}{=} \{\phi_1^k, \phi_2^k, \dots, \phi_n^k\}$ is considered, thus Ω_k is the k^{th} overall number of path overlaps produced.

Given a network graph G , an initial solution Φ_0 with respective Ω_0 , and a number of $k_{max} \in \mathbb{N}$ sub-optimal sets of routes Φ_k , we formulate the problem of minimizing Ω as follows:

$$\begin{aligned} \underset{k}{\text{minimize}} \quad & \Omega_k = \sum_{\forall i, j \in [1, n] \wedge i \neq j} \delta_{i, j}(\Phi_k) \\ \text{subject to} \quad & k \in [1, k_{max}], \\ & \Phi_k \in [\Phi_1, \Phi_{k_{max}}] \end{aligned} \quad (4)$$

where $\delta_{i, j}(\Phi_k)$ is the number of *node overlaps* between the routes of the flows f_i^k and $f_j^k \in F_k$ (i.e. δ_{ij}^k), $\forall i, j \in [1, n] \wedge i \neq j$. The result is $\Omega = \Omega_k^{\text{min}}$, i.e., the minimal *overall number of overlaps* within all the k_{max} possible sets of flow paths $[\Phi_1, \Phi_{k_{max}}]$. We denote as Φ_{opt} the set of optimal routes, to any of the Φ_k sets of flow paths that produce Ω_k^{min} .

► **Note.** Each individual route ϕ_i^k in Φ_k is defined as a sub-optimal version of the shortest-path ϕ_i^0 , i.e., the route length ζ_i^k of ϕ_i^k is always greater than or equal to ζ_i^0 . In the same way, each individual transmission time C_i^k of f_i^k is always a sub-optimal version of C_i^0 , thus C_i^k being always greater than or equal to C_i^0 . Yet, we make clear that a larger C_i^k does not necessary implies a larger Ω_k , and vice-versa. We also note that although, in general, a larger C_i^k may lead to larger end-to-end delays, we conjecture that this impact is less detrimental than the impact of flow path overlaps, thus we target to minimize this latter factor.

5 Proposed Solution: Minimal-Overlaps (MO) Greedy Heuristic

We propose an approximate method to solve the problem formalized in (4). The approach is based on a greedy heuristic that recommends a single set Φ_k at each k^{th} iteration, and then computes the corresponding Ω_k . The smallest of the Ω_k after k_{max} iterations is designated as Ω_k^{min} , which is reported at the last iteration. We detail the proposed method as follows:

Step 1 (Initial solution)

- At $k = 1$, $\Phi_k = \Phi_1$ is computed as function of the path overlaps resulting from Φ_0 , i.e., from the initial set of (hop count) shortest-paths, and from the graph $G = (V, E)$.
- Φ_1 is computed as the set of (weighted) shortest-paths of $G^1 = (V, E^1)$, a modified version of the unitary-weighted G whose set of edges is weighted as function of the node-overlapping degree derived from the set of flow paths Φ_0 .
- The cost function $W_{i, j}(u, v)$ that weight any edge (u, v) in G whose nodes u and $v \in V$ are simultaneously in any pair of routes ϕ_i^0 and $\phi_j^0 \in \Phi_0$ is defined as follows:

$$W_{i, j}(u, v) = 1 + \sum_{e=1}^{\delta_{i, j}^0} \psi \quad (5)$$

where $\psi \in \mathbb{R}$ is an arbitrary factor², and $\delta_{i, j}^0$ is the number of *node overlaps* resulting from the routes ϕ_i^0 and $\phi_j^0 \in \Phi_0$, $\forall i, j \in [1, n] \wedge i \neq j$.

² The arbitrary factor Ψ is a user-defined parameter, here assumed as constant, but that can be optimized to provide better (e.g., faster) solutions for Ω_k^{min} . Yet, this aspect is not covered in this work.

- Therefore, Φ_1 is the resulting set of (weighted) shortest-paths over G^1 , and Ω_1 the corresponding *overall number of overlaps* for Φ_1 .
- If $\Omega_1 < \Omega_0$, then $\Omega_k^{min} = \Omega_1$, else $\Omega_k^{min} = \Omega_0$.

Step 2 (Greedy search)

- For any $k \in]1, k_{max}]$, the search for a Ω_k^{min} is generalized.
- $G^k = (V, E^k)$ is defined as the modified version of $G^{(k-1)} = (V, E^{(k-1)})$, whose set of edges is weighted using the following cost function due the node-overlapping of flow paths:

$$W_{i,j}^k(u, v) = 1 + \sum_{e=1}^{\delta_{i,j}^{(k-1)}} \psi \quad (6)$$

where $\delta_{i,j}^{(k-1)}$ results from the routes $\phi_i^{(k-1)}$ and $\phi_j^{(k-1)} \in \Phi_{(k-1)}, \forall i, j \in [1, n] \wedge i \neq j$.

- Φ_k is thus computed as function of the path overlaps resulting from the set $\Phi_{(k-1)}$.
- If $\Omega_k < \Omega_k^{min}$, then $\Omega_k^{min} = \Omega_k$, else $\Omega_k^{min} = \Omega_k^{min}$.

Step 3 (Best solution)

- At $k = k_{max}$, the algorithm finishes and provides Ω_k^{min} , i.e., the minimal *overall number of overlaps* within all the k_{max} Φ_k sets recommended. Note that the quality of Ω_k^{min} will depend on the quality of the generated Φ_k sets, as well as on the total number of iterations.
- The optimal set of routes Φ_{opt} is thus the Φ_k which provides Ω_k^{min} .

6 Performance Evaluation

We report here the relevant information for the data sets generation and performance evaluation of both, the proposed *minimal-overlap* (weighted) shortest-path method denoted as **MO**, and the baseline (hop-count) *shortest-path* method, denoted as **SP**. We present the assessment of the real-time performance through the schedulability ratio metric by considering varying network topologies and workload conditions. We further assess the influence of varying topologies and workload on the number of overlaps, routes length, channel contention and transmission conflicts. We show that MO significantly outperforms SP in terms of schedulability ratio, number of overlaps and transmission conflicts, while having marginal impact on the average length of the routes and channel contention.

6.1 Simulation setup

The configuration details related to the generation of random network topologies and real-time network flows are described next:

Network topologies

We prepared a set of 100 network topologies built upon the random generation of network graphs. Each graph was created based on a sparse uniformly distributed random matrix of size $N \times N$ (with $N \in \mathbb{N}$) and density Λ (with Λ in $[0, 1] \in \mathbb{R}$). Each sparse matrix acted as an adjacency matrix for the graph generation. The size of the sparse matrix ($N \times N$) as well as the number of vertices in the graph (N) were fixed for all the simulation instances. We set $N = 66$ as in [19], for benchmarking purposes. The vertex with the highest betweenness

centrality³ was chosen as gateway, while the rest $N - 1$ vertices represented field devices and access points. A subset of $n \subset N$ of field devices is chosen as sensors, thus assumed to periodically transmit data to the gateway. For comparison, the range of n varied within $[2, 22]$ as in [19]. We considered varying values of $\Lambda = \frac{\lambda}{N}$, where λ indicates the median vertex degree of the graph. We controlled Λ by varying $\lambda \in \mathbb{N}$ in the range $[4, 12]$. We justify this choice since in practical WSA deployments, each node is typically required to be connected at least to other 3 nodes (i.e., $\lambda \geq 3$). We note that, in general, this setup provided connected graphs, but in the few cases that nodes were disconnected, we forced a random connection (edge) with any of the other connected vertices.

Given the above configuration, we generated a set of shortest-path (hop-count) routes between the set of n sensors and the gateway, for each graph, thus providing 100 instances of sets of n routes for the baseline method. In the case of the MO method, the set of routes was generated from the baseline, but taken into consideration the degree of node-overlapping between routes as described in Section 5. The proposed set of (minimal-overlap) routes is thus the result of the evaluation of k_{max} edge-weighted versions of each baseline graph instance. On the weight functions, we used the arbitrary factor Ψ as equal to the graph density, i.e. $\Psi = \Lambda$, for all the cases. We considered this value based on empirical observations. For the sake of scalability, we considered for all the experiments $k_{max} = 100$. We observed that a greater number of iterations (k_{max}) can lead to a lower node-overlapping degree, but at the cost of higher execution times; thus we do not further explore this factor. Note that in the case of overall number of overlaps Ω_k that reaches zero, the algorithm stops.

Network flows

We consider a random set of $n \in [2, 22]$ real-time network flows for each of the 100 topologies generated. The complete set of flows corresponds to the set of periodic data transmissions generated by the $n \subset N$ sensor nodes in the graph. Each of these flows f_i is characterized by a 4-tuple (C_i, D_i, T_i, ϕ_i) following the model described in 3.2. Each C_i represents the effective transmission time (in slots) for the route ϕ_i , and can be obtained directly from the product of the number of hops (links or edges) traveled by the path ϕ_i from source to destination, and the number of transmissions assigned to each slot (we assume $w = 2$, as in WirelessHART). Thus, each of the 100 random graph instances is also generating, randomly, the C_i and ϕ_i occurrences. Hence, being applicable for both the MO and the SP methods. The corresponding T_i periods were assumed as random harmonically generated in the form of 2^η time slots, with $\eta \in \mathbb{N}$ in the range $[4, 7]$ (as in [19]). This assumption leads to a direct computation of the hyperperiod $H \in \mathbb{N}$ (a.k.a, superframe length) as the maximum period within the range of harmonic periods, or as generally defined, the least common multiple of the set of periods, i.e., $H = lcm(T)$, where $T = \{T_1, T_2, \dots, T_n\}$. So, in this case, $H = 2^7 = 128$ slots (or equivalently 1280ms). We used $H = 128$ slots as the length of the time interval ℓ for the purposes of schedulability assessment, as well as for the performance evaluation of all the other metrics. The schedulability was evaluated using the test presented in 3.2.1, when considering a worst-case $\Delta_{i,j}$ as in [11]. Finally, we assume $D_i = T_i$ for all the cases, thus reducing the original problem to an implicit-deadline model.

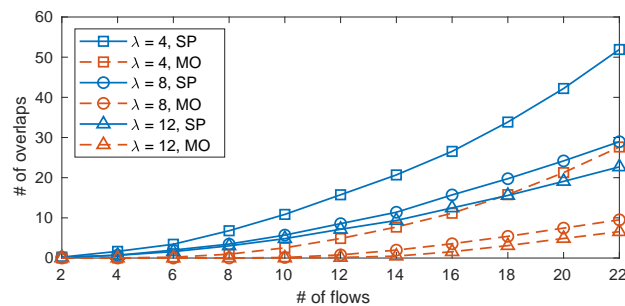
³ The betweenness centrality metric was chosen to maintain a consistent relevance of the gateway within the random topologies, thus avoiding an arbitrary gateway position (e.g. at the border). As different centrality metrics can be defined based on application needs, this consideration requires further research.

► **Note.** For the sake of simplicity, we have only considered network flows that travel from sensors to the gateway, thus the case of uplink deadline-constrained single (non-redundant) set of paths (as in convergecast). Yet, we note the work can be extended to consider the downlink component, e.g., by considering additional routes traveling from the gateway to actuators in a deadline-constrained fashion, both for symmetric or asymmetric (uplink-downlink) routing, thus including the case of graph routing (as in [11]) with multiple redundant paths (and/or for multi-cast). We aim to further analyze these aspects in future research works.

6.2 Simulation Results

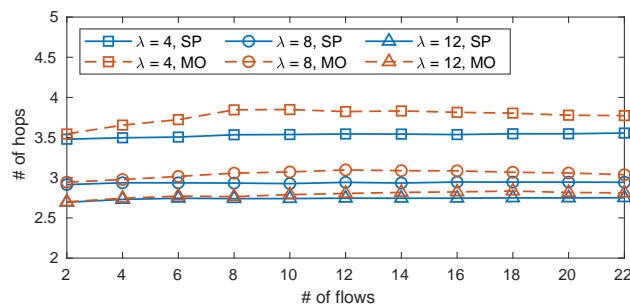
The average performance of 100 test cases for both the SP and MO methods under varying network topologies and workload conditions is reported next:

(i) Number of overlaps



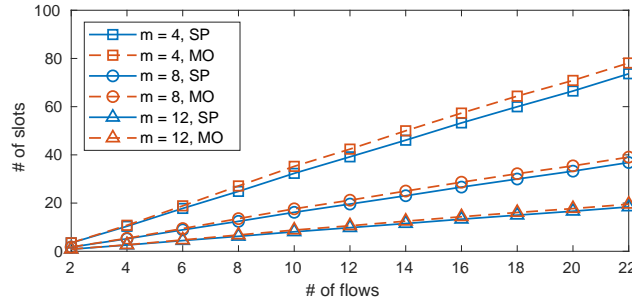
■ **Figure 2** The average number of overlaps under varying number of flows $n \in [2, 22]$, and varying median vertex degree $\lambda = \{4, 8, 12\}$. $N = 66$ nodes, $m = 8$ channels.

(ii) Routes length



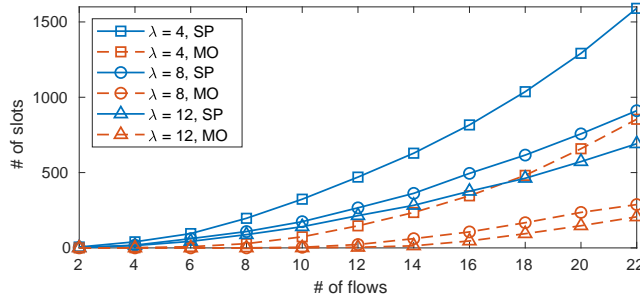
■ **Figure 3** The average length of the routes under varying number of flows $n \in [2, 22]$, and varying median vertex degree $\lambda = \{4, 8, 12\}$. $N = 66$ nodes, $m = 8$ channels.

(iii) Channel contention



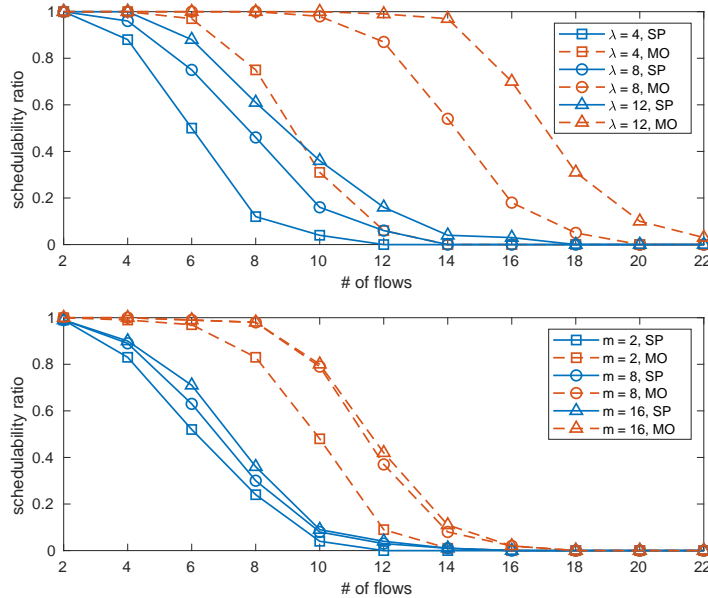
■ **Figure 4** The average contention demand under varying number of flows $n \in [2, 22]$, and varying number of channels $m = \{4, 8, 12\}$. $N = 66$ nodes, median vertex degree $\lambda = 4$.

(iv) Transmission conflicts



■ **Figure 5** The average conflict demand under varying number of flows $n \in [2, 22]$, and varying median vertex degree $\lambda = \{4, 8, 12\}$. $N = 66$ nodes, $m = 8$ channels.

(v) Schedulability ratio



■ **Figure 6** The schedulability ratio under varying number of flows $n \in [2, 22]$ and fixed number of nodes $N = 66$. On top, the case of varying median vertex degree $\lambda = \{4, 8, 12\}$ and $m = 8$ channels. On bottom, the case of varying number of channels $m = \{2, 8, 16\}$ and median vertex degree $\lambda = 4$.

6.3 Discussion

The performance comparison of MO and SP in terms of the overall (average) number of overlaps confirms the main intuition behind the proposed method. The greedy heuristic search of MO albeit sub-optimal is able to effectively reduce the node-overlapping degree in up to half (and more) of the baseline (Fig. 2). This, depending on the number of flows and network density (or vertex degree). The exponential growth of the overlaps as function of the number of flows justifies the need for its mitigation. This trend is also observable in the growth imposed on transmission conflicts, which directly depends on the number of overlaps (Fig. 5). The compromise on the route lengths is marginal, whose impact is even negligible on more connected networks (Fig. 3). The influence on the channel contention is also minor, being discernible (in practice) only on networks with a lower number of active channels (Fig. 4). The benefits on the overall network schedulability are clear, regardless of the degree of connectivity (Fig. 6, top) or the available radio channels (Fig. 6, bottom), but suggesting a superfluous effect on this latter parameter at a given point (see $m = 8$ and $m = 16$). All in all, the prospect for the proposed method is promising, both as a general technique to reduce the impact on conflict delays on wireless mesh network, or as a specific joint EDF scheduling and routing framework to provide real-time guarantees on TSCH-based networks.

7 Summary & Conclusion

We have developed an effective *real-time wireless routing* for TSCH-based WSANs with packet transmissions scheduled under an EDF policy. The approach based on a greedy heuristic for path-overlap minimization shown to be successful in reducing transmission conflicts and improving schedulability, while having marginal impact on contention and routes length. Simulation results under varying topologies and workload conditions revealed a remarkable dominance of our approach over the more common (hop-count) shortest path method. To conclude, we leverage on our prior work on schedulability analysis to frame both together as a novel *joint real-time scheduling and routing* framework for TSCH WSANs.

References

- 1 Giuliana Alderisi, Svetlana Girs, Lucia Lo Bello, Elisabeth Uhlemann, and Mats Björkman. Probabilistic scheduling and adaptive relaying for WirelessHART networks. In *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–4. IEEE, 2015.
- 2 Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Sebastian Stiller. Improved multiprocessor global schedulability analysis. *Real-Time Systems*, 46(1):3–24, 2010.
- 3 Keoma Brun-Laguna, Pascale Minet, and Yasuyuki Tanaka. Optimized scheduling for time-critical industrial IoT. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.
- 4 Domenico De Guglielmo, Simone Brienza, and Giuseppe Anastasi. IEEE 802.15. 4e: A survey. *Computer Communications*, 88:1–24, 2016.
- 5 Miguel Gutiérrez Gaitán, Patrick M. Yomsi, Pedro M. Santos, and Luís Almeida. Work-in-progress: Assessing supply/demand-bound based schedulability tests for wireless sensor-actuator networks. In *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, pages 1–4. IEEE, 2020.
- 6 Miguel Gutiérrez Gaitán and Patrick Meumeu Yomsi. FF-DBF-WIN: On the forced-forward demand-bound function analysis for wireless industrial networks. In *2018 30th Euromicro Conference on Real-Time Systems (ECRTS), Proceedings of the Work-in-Progress Session*, pages 13–15, 2018.

- 7 Miguel Gutiérrez Gaitán and Patrick Meumeu Yomsi. Multiprocessor scheduling meets the industrial wireless: A brief review. *U. Porto Journal of Engineering*, 5(1):59–76, 2019.
- 8 Harrison Kurunathan, Ricardo Severino, Anis Koubâa, and Eduardo Tovar. Worst-case bound analysis for the time-critical MAC behaviors of IEEE 802.15. 4e. In *2017 IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, pages 1–9. IEEE, 2017.
- 9 Chenyang Lu, Abusayeed Saifullah, Bo Li, Mo Sha, Humberto Gonzalez, Dolvara Gunatilaka, Chengjie Wu, Lanshun Nie, and Yixin Chen. Real-time wireless sensor-actuator networks for industrial cyber-physical systems. *Proceedings of the IEEE*, 104(5):1013–1024, 2015.
- 10 Rahul Mangharam, Anthony Rowe, Raj Rajkumar, and Ryohei Suzuki. Voice over sensor networks. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS)*, pages 291–302. IEEE, 2006.
- 11 Venkata Prashant Modekurthy, Dali Ismail, Mahbubur Rahman, and Abusayeed Saifullah. A utilization-based approach for schedulability analysis in wireless control systems. In *2018 IEEE International Conference on Industrial Internet (ICII)*, pages 49–58. IEEE, 2018.
- 12 Marcelo Nobre, Ivanovitch Silva, and Luiz Affonso Guedes. Routing and scheduling algorithms for WirelessHART networks: A survey. *Sensors*, 15(5):9703–9740, 2015.
- 13 Abusayeed Saifullah, You Xu, Chenyang Lu, and Yixin Chen. Real-time scheduling for WirelessHART networks. In *2010 31st IEEE Real-Time Systems Symposium (RTSS)*, pages 150–159. IEEE, 2010.
- 14 Stefano Scanzio, Mohammad Ghazi Vakili, Gianluca Cena, Claudio Giovanni Demartini, Bartolomeo Montrucchio, Adriano Valenzano, and Claudio Zunino. Wireless sensor networks and TSCH: A compromise between reliability, power consumption, and latency. *IEEE Access*, 8:167042–167058, 2020.
- 15 Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734, 2018.
- 16 Rasool Tavakoli, Majid Nabi, Twan Basten, and Kees Goossens. Topology management and TSCH scheduling for low-latency convergecast in in-vehicle WSNs. *IEEE Transactions on Industrial Informatics*, 15(2):1082–1093, 2018.
- 17 Federico Terraneo, Paolo Polidori, Alberto Leva, and William Fornaciari. TDMH-MAC: Real-time and multi-hop in the same wireless MAC. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 277–287. IEEE, 2018.
- 18 Mališa Vučinić, Tengfei Chang, Božidar Škrbić, Enis Kočan, Milica Pejanović-Djurišić, and Thomas Watteyne. Key performance indicators of the reference 6TiSCH implementation in Internet-of-Things scenarios. *IEEE Access*, 8:79147–79157, 2020.
- 19 Chengjie Wu, Dolvara Gunatilaka, Mo Sha, and Chenyang Lu. Real-time wireless routing for industrial internet of things. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 261–266. IEEE, 2018.
- 20 Changqing Xia, Xi Jin, and Peng Zeng. Resource analysis for wireless industrial networks. In *Proceedings of the 12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, pages 424–428. IEEE, 2016.

Static Allocation of Basic Blocks Based on Runtime and Memory Requirements in Embedded Real-Time Systems with Hierarchical Memory Layout

Philipp Jungklass

IAV GmbH, Gifhorn, Germany
philipp.jungklass@iav.de

Mladen Berekovic

Universität zu Lübeck, Institute of Computer Engineering, Germany
berekovic@iti.uni-luebeck.de

Abstract

Modern microcontrollers for safety-critical real-time systems use a hierarchical memory system to increase execution speed and memory capacity. For this purpose, flash memories, which offer high capacity at low transfer rates, are combined with scratchpad memories, which provide high access speed at low memory capacities. The main goal is to use both types of memory in such a way that their advantages are optimally exploited. The target is to allocate runtime-intensive code fragments with low memory requirements to the fast scratchpad memories. Previous approaches to separate program code on system memories consider the executed functions as the smallest logical unit. This is contradicted by the fact that not all parts of a function have the same computing time in relation to their memory usage. This article introduces a procedure that automatically analyses the compiled source code and identifies runtime intensive fragments. For this purpose, the translated code is executed in an offline simulator and the maximum repetition for each instruction is detected. This information is used to create logical code fragments called basic blocks. This is repeated for all functions in the overall system. During the analysis of the functions, the dependencies between them are also extracted and a corresponding call-graph with the call frequencies is generated. By combining the information from the call graph and the evaluation of the basic blocks, a prognosis of the computing load of the respective code blocks is created, which serves as base for the distribution into the fast scratchpad memories. To verify the described procedure, EEMBC's CoreMark is executed on an Infineon AURIX TC29x microcontroller, in which different scratchpad sizes are simulated. It is demonstrated that the allocation of basic blocks scales significantly better with smaller memory sizes than the previous function-based approach.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture

Keywords and phrases Memory Architecture, Memory Management, Real-time Systems

Digital Object Identifier 10.4230/OASICS.NG-RES.2021.3

Acknowledgements Special thanks to Arne Bredemeier from Infineon.

1 Introduction

In modern systems with hard real-time requirements, microcontrollers are increasingly used, which implement a hierarchical memory layout with different memory technologies. The reason for this is that memories with a large capacity, such as flash, have a comparatively slow access time and a significantly reduced bandwidth in contrast to fast scratchpad memories based on Static Random-Access Memory (SRAM). In comparison, the disadvantage of SRAM is that it requires a lot of space on the wafer during manufacturing, which significantly



© Philipp Jungklass and Mladen Berekovic;
licensed under Creative Commons License CC-BY

Second Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2021).

Editors: Marko Bertogna and Federico Terraneo; Article No. 3; pp. 3:1–3:14

OpenAccess Series in Informatics



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

increases production costs. For this reason, the manufacturers of such microcontrollers try to combine these two types of memory efficiently with each other, thus exploiting the advantages of both technologies and reducing the disadvantages [13] [14].

■ **Table 1** Relation of the memory sizes of microcontrollers for real-time systems [7] [15] [18] .

Microcontroller	Flash-Memory / KB	SRAM-Memory / KB
Infineon AURIX TC29x (3 processor cores)	Code: 8192 Data: 1024	Code: 96 Data: 600
Texas Instruments TMS320F2838x (5 processor cores)	Code: 1536 Data: -	Code: - Data: -
ST SPC58 family (3 processor cores)	Code: 6144 Data: 256	Code: 48 Data: 160

As table 1 shows, the storage capacity of SRAM-based memories is significantly smaller compared to flash memories. It should also be noted that all specifications refer to the respective overall system. Therefore it is important to note that, for example, the Infineon AURIX TC29x provides a relatively large SRAM memory of 96 KB for code execution, but this is divided among the three processor cores, so that only 32 KB is directly available for each core [7]. The great potential of these small but powerful memories is that the fast access times significantly increase the execution speed. In case of the already mentioned Infineon AURIX TC29x, code from the local SRAM memory of the corresponding processor core is executed 3.5 times faster than using the flash memory. Furthermore, by allocating local copies of selected code parts to the respective SRAM memory of the cores, the number of concurrent accesses can be reduced significantly, which is particularly relevant for real-time multicore systems [16] [12] [8]. Due to the limited memory capacity it is essential to evaluate which code fragments should be allocated in these memories. As these code fragments are needed more frequently, the increased execution speed has a greater influence on the performance of the overall system. For this reason, this article presents a procedure that analyzes the code at instruction level and allocates particularly runtime-intensive basic blocks to the fast scratchpad memories. For this purpose, the call frequency is set in relation to the memory consumption, with the goal of an optimized usage of the fast memory. The structure of this article is divided into six chapters. After the introduction, related work as well as the previous optimization approaches are presented. This is followed by a description of the developed concept, which is implemented practically in the fourth section. To prove the functionality, the presented method is applied to the CoreMark of EEMBC in the fifth chapter. Finally, the results are discussed and an outlook on future extensions is given.

2 Related Work

Previous work on optimizing the use of existing scratchpad memory can be divided fundamentally into two categories. In static distribution, the allocation of code and data takes place during development and is fixed at system runtime. The advantage of this approach is that a static allocation can more easily be mapped in a timing model. This is necessary for the calculation of the Worst-Case Execution Time (WCET), which is mandatory for the certification of a safety-critical real-time system. The disadvantage, in contrast, is the sometimes suboptimal utilization of the small scratchpad memory, which cannot realize its speed advantage optimally, depending on the execution path. Dynamic approaches try to

compensate this by adapting the contents of the scratchpad memory at runtime according to the program flow. It should be noted, that dynamic approaches cause an overhead due to copy processes and the associated administration, that should not be underestimated.

2.1 Static Allocation

In [10] a procedure for the static distribution of code and data into the available memory of a real-time multicore system is shown. For this purpose priorities are calculated for all functions and variables in the system, which result from the call frequency as well as the number of concurrent accesses. Depending on the priority, the code and data are distributed to the memories in the system and local copies are created if necessary. In contrast to this work, in the article, functions are defined as the smallest allocatable unit, whereby less frequently executed code paths are also allocated to the scratchpad memory. A further approach of static separation is discussed in [2]. The article describes a compiler strategy for distributing the stack and global variables into the Random-Access Memory (RAM) of the microcontroller. As boundary conditions for the procedure the renouncement of an Memory Management Unit (MMU) as well as the use of a Non-Uniform Memory Access (NUMA)-based memory management is demanded. Analogous to the method presented in this article, the execution speed shall be improved by an optimized memory distribution depending on the call frequency. In contrast, the analysis is limited to global variables and the separation of the stack. In the article [9] and [11] approaches for the optimization of explicit intercore communication in multicore systems are presented. The concepts are based on the assumption that shared variables should be allocated to the local scratchpad memory of those processor cores that access it more often. The two methods have a similar approach, but differ in their respective implementation. In contrast to the procedure described here, the concepts in the two articles are limited to the allocation of variables for intercore communication.

2.2 Dynamic Allocation

One possibility for the dynamic use of scratchpad memory is described in the approach [13]. For the presented procedure the functions are divided into basic blocks in the first step. In the second step, the basic blocks are determined, which are called in the Worst-Case Execution Path (WCEP). Then these basic blocks are allocated to the fast scratchpad memories, which increases the execution speed. In contrast to the approach in this article, the described concept uses dynamic memory management, which reloads the basic blocks as required. As a result, there is no evaluation or prioritisation of functions among themselves, which would be necessary for static memory management. Instead, an offline evaluation of the basic blocks is carried out to determine if they are qualified for reloading at runtime, because the copy time causes a significant overhead, which has to be compensated by the increased execution speed. In [19] a method is described which optimizes concurrent accesses to shared memory in real-time capable multicore microcontrollers. For this purpose the scheduling of the operating system is extended, whereby a task consists of three phases. In the first phase, all required data is loaded from the shared memory into the local caches of the processor cores before it is executed in the second phase. In the last phase, the generated results are copied back into shared memory. Although the procedure in this article focuses on caches, this method is also possible when using scratchpad memories. In contrast to the concept in this article, dynamic memory management is implemented here to reduce competing accesses. However, in order to reduce the number of concurrent accesses, all possible paths of a task must be completely copied to the local memory. Only in this case copying phases during execution

3:4 Static Allocation of Basic Blocks

can be effectively prevented. However, this also copies paths of a task whose computing time has no significant influence on the runtime. The method presented in the article [4] describes a procedure that dynamically allocates basic blocks into scratchpad memory based on their call frequency. To reduce the overhead of dynamic memory management, basic blocks with a fixed size are used, which can compensate the problem of fragmentation. The problem is, fixed-size memory blocks are not always completely filled, which in effect reduces the efficiency of memory usage.

3 Concept

The concept described here is to identify code fragments within a function that have a high runtime and low memory requirements. For this purpose, the translated code is analyzed in an offline simulator and the minimum and maximum execution frequency is determined for each instruction. In addition, the memory requirements of each instruction and its execution time in clock cycles are extracted from the processor architecture description. This information can be used to create a table for each function, which contains all the data required to evaluate each instruction. The goal is to allocate the particularly computationally time-intensive areas of a function to the fast scratchpad memories of the microcontroller, which allows the low capacity of these high-performance memories to be used more efficiently. The procedure is illustrated using listing 1 as an example.

```
1  /* function to copy the message from
2  receive buffer in destination buffer */
3  uint32 CopyMessage(uint8* dest, uint32 length)
4  {
5      /* return value */
6      uint32 errorCode = TRUE; /* TRUE = 1 */
7
8      /* check the maximum buffer length */
9      if(length <= 32)
10     {
11         for(uint32 i = 0; i < length; i++)
12         {
13             /* copy the received byte from receive
14             buffer (rb) into destination buffer */
15             *dest++ = rb[i];
16         }
17     }
18     else
19     {
20         /* invalid length */
21         errorCode = FALSE; /* FALSE = 0 */
22     }
23     return errorCode;
24 }
25
```

■ **Listing 1** Example CopyMessage: Source code (ANSI C)

The minimal example in the listing contains a function which copies the data from a global array `rb[]` byte by byte into a target memory, whose address is passed to the function as bytewriter `dest`. To avoid memory overflow, the requested variable `length` is checked for the maximum value of 32 before copying. Depending on the result of the check, the data is copied to the target memory or a corresponding error message is returned to the calling function. Using the C code shown here, it can already be estimated that the runtime is primarily dependent on the length of the data to be copied.

3.1 Machine Code Analysis

In the first step of the analysis, the source code for the target platform is compiled. The reason for this procedure is that modern compilers support a variety of optimizations that take into account architectural characteristics like jump predictions, superscalar architectures or errors in processor design. Therefore, depending on the configured optimization level, the translated machine code can differ massively from the original source code. To ensure that the available scratchpad memory can still be used efficiently, the translated machine code is therefore used as the basis for optimization.

In the example shown here, this is done for an Infineon AURIX TC29x, using the Tasking Compiler in version 6.2r2 with the optimization level 00 as compiler. The result can be taken from the commented listing 2.

```

1  mov d2,#1           ;Move
2  mov d15,#32        ;Move
3  jge.u d15,d4, .L5  ;Jump if Greater Than or Equal
4  j .L17             ;Jump Unconditional
5  mov d15,#0        ;Move
6  j .L14             ;Jump Unconditional
7  movh.a a15,#@his(rb) ;Move High to Address
8  lea a15,[a15]@los(rb) ;Load Effective Address
9  addsc.a a15,a15,d15,#0 ;Add Scaled Index to Address
10 ld.bu d0,[a15]    ;Load Byte Unsigned
11 st.b [a4],d0      ;Store Byte
12 add.a a4,#1       ;Add Address
13 add d15,#1        ;Add
14 jge.u d15,d4, .L16 ;Jump if Greater Than or Equal
15 j .L17            ;Jump Unconditional
16 j .L18            ;Jump Unconditional
17 mov d2,#0        ;Move
18 j .L19            ;Jump Unconditional
19 ret               ;Return from Call
20

```

■ **Listing 2** Example CopyMessage: Source code (machine code)

With the help of the Infineon TriCore Simulator (TSIM) the translated code is executed and combined with the information from the processor architecture description. The result is shown in the table 2.

As it can be seen in the table 2, the offline simulator analyzes all paths of the function and determines the minimum and maximum call frequency for each instruction. This information is combined with the memory requirements and the runtime. This combination of parameters is used to identify fragments that require a lot of computing power. Additionally, the WCEP is determined for each function. This procedure is particularly interesting for safety-critical real-time systems, because only the WCET is relevant for the evaluation of such systems.

3.2 Basic Block Prioritisation

After the identification of the relevant blocks, a prioritisation is carried out. In the first step, all basic blocks along the WCEP are evaluated regarding of their runtime. Due to the focus on real-time systems, these fragments receive the highest priority within the function. Subsequently, all other paths are examined for their optimization potential. After completion of this analysis, all basic blocks of this function have a priority based on their call frequency, with the WCEP having the highest rating. The formula (1) describes the calculation.

$$p_f(bb) = \sum_{i=0}^{i_{\max}} (e_{\max}(i) \cdot r_{\max}(i)) + p_{WCEP} \quad (1)$$

3:6 Static Allocation of Basic Blocks

$p_f(bb)$	Priority of the basic block bb within the function f
$e_{\max}(i)$	Maximum execution frequency of instruction i
$r_{\max}(i)$	Maximum runtime of instruction i
p_{WCEP}	Additional priority for the WCEP

As the procedure presented here is a static memory allocation, a dynamic adjustment of the scratchpad contents at runtime is not possible. In fact of this, it is not sufficient to prioritise only the basic blocks within a function. It is also necessary to consider the call frequencies of the individual functions in the overall system in the evaluation. For this reason, all function jumps and their frequency are logged during the analysis of the functions and used to construct a call graph. On the basis of this information, the call frequency can be determined for each function, which is then included in the prioritisation of the basic blocks. The following formula (2) illustrates the procedure.

$$p_s(bb) = e_{\max}(f) \cdot p_f(bb) \quad (2)$$

$p_s(bb)$	Priority of the basic block bb within the system s
$e_{\max}(f)$	Maximum execution frequency of function f

■ **Table 2** Example CopyMessage: memory usage, runtime and call frequency.

Instruction	Memory usage /Byte	Runtime /Ticks	Execution frequency Min/Max /Ticks
mov d2,#1	4	1	1/1
mov d15,#32	4	1	1/1
jge.u d15,d4, .L5	4	1	1/1
j .L17	4	1	0/1
mov d15,#0	4	1	0/1
j .L14	4	1	0/1
movh.a a15,#@his(rb)	4	1	0/32
lea a15,[a15]@los(rb)	4	1	0/32
addsc.a a15,a15,d15,#0	4	1	0/32
ld.bu d0,[a15]	4	1	0/32
st.b [a4],d0	4	1	0/32
add.a a4,#1	2	1	0/32
add d15,#1	2	1	0/32
jge.u d15,d4,.L16	4	1	0/33
j .L7	4	1	0/32
j .L18	4	1	0/1
mov d2,#0	4	1	0/1
j .L19	4	1	1/1
ret	4	4	1/1
Total	72	22	10/300

3.3 Basic Block Separation

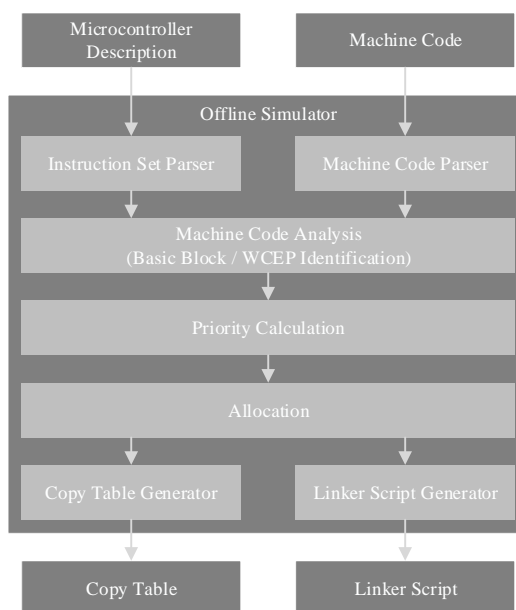
The basic blocks are separated by jumps, which are already contained in the machine code. Only the address have to be manipulated. By this procedure both the runtime and the memory consumption can be kept constant. Ideally, most compilers generate entry and exit jumps during the translation of loops, which are suitable for this modification. In the example from listing 2 used here, the jump instructions in line 6 and line 16 are used for this purpose. Analogous to the procedure with loops, this is also done with branches, which are also implemented using conditional jump instructions. Depending on the architecture used, however, it must be taken into account that often only one branch path is reached by a jump. This becomes clear in listing 2, where in line 3 the conditional jump is executed. If the check is unsuccessful, the following line is executed, but without a jump. In fact of this, care must be taken during compilation that the runtime-intensive path is always reached via a jump, so that this can be allocated to the fast scratchpad memory if necessary. Applying the presented concept to the example shown here results in all instructions from line 7 to line 16 inclusive being allocated to the fast scratchpad memory. By this procedure the required memory can be reduced from 72 bytes, when allocating the complete function, to 36 bytes. Despite the 50% reduction in memory requirements, 290 of the 300 required instructions are executed from scratchpad memory in the case of WCEP.

4 Implementation

The offline simulator for the analysis of the translated machine code basically consists of the components shown in figure 1. As input variables, the simulator receives a description of the microcontroller, which contains the addresses of the memories, the addressing types and the instruction set. On the other hand the translated machine code is entered into the simulator. Both input files are then processed by a parser, which converts the information into an uniform format. This input parser is intended to enable easy expandability for further microcontroller architectures or compilers. In the next step, the code is analyzed using the concept described in chapter 3 and then the priority for each basic block is calculated. Using the knowledge gained, an optimized memory allocation can be calculated in the next-to-last step with the help of the allocation block. At last, a modified copy table as well as the corresponding linker script will be created using two generators.

4.1 Basic Block Separation

After the complete analysis of all functions in the overall system and their prioritisation, the jump instructions in the compiled machine code are manipulated to separate the basic blocks. In general, modern microcontrollers use an instruction set that provides a large number of such instructions. These jump instructions can be divided into two categories. The first category consists of absolute jumps, which refer directly to an address in memory. In contrast, the second variant always uses relative jumps in relation to the current address. For example, a relative jump can refer to an address that is 20 bytes further in memory. The advantage of these relative jumps is their reduced memory requirement, since no complete address has to be stored. The disadvantage is their reduced range, since it is only possible to refer to a much smaller memory area. In order to implement the method presented in this article, it must be possible to implement a jump to the scratchpad within a function allocated in the flash memory. Depending on the microcontroller used, absolute jumps are absolutely necessary for this, since the two types of memory often use different address ranges. For the Infineon AURIX TC29x used in this article, the absolute addresses for the different memories are listed in the table 3.



■ **Figure 1** Offline Simulator - Detailed Design.

■ **Table 3** Infineon AURIX TC29x - Code Memory Addresses [7].

Memory	Address Range	Size/KB
Code Scratchpad (CPU0)	0x7010.0000-0x7010.7FFF	32
Code Cache (CPU0)	0x7010.8000-0x7010.FFFF	32
Flash	0x8000.0000-0x807F.FFFF	8192

The relative jumps of the TriCore architecture of the AURIX use one byte for addressing, which allows a maximum jump width of 512 bytes with an addressing granularity of two bytes. As a result, it is only possible to switch between the flash memory and the scratchpad with an absolute jump. Due to the fact that a relative jump requires less memory space compared to an absolute jump, this creates a problem because the two types of addressing cannot simply be replaced. There are three different approaches to solve this problem, which are explained in the following sections [6].

4.1.1 Absolute Jumps

Modern compilers offer the option to avoid relative jumps in memory by various configuration settings, whereby only absolute addresses are used. The advantage of this variant is that the used addresses in the memory can be easily replaced. The disadvantage is the increased memory consumption of this method, which results from the exclusive use of absolute addresses. One way to reduce this problem is the specific use of compiler commands in the source code, whereby the use of absolute addresses is only applied at defined positions.

4.1.2 Jump Table

If the compiler that is used, does not provide an option to disable relative jumps, an additional jump table in memory can be used. For this purpose, a table containing the absolute addresses is stored in memory near the function to be optimized. The relative jumps within the function

are manipulated in such a way that they refer to the correct lines within the table where the absolute address is located. The advantage of this implementation is that no support from the compiler is required. In contrast, the jump table occupies additional memory and each jump into the fast scratchpad requires an additional relative jump into the table before the actual target address is reached. This increases the runtime, which must be taken into account in the evaluation of the separation.

4.1.3 Memory Reservation

Another possibility is to reserve additional memory space directly next to a relative jump instruction. This can be achieved by integrating so-called zero-operations already in the C code. After compiling the source code, the memory space of the relative jump and the zero-operation is used to integrate an absolute jump. Similar to the jump table, this procedure does not require any support from the compiler. However, this variant requires a complex analysis to integrate the required zero-operations at the correct position in the source code.

4.2 Basic Block Allocation

To copy the selected basic blocks into the scratchpad memory of the microcontroller, the copy table must be extended accordingly. Using the copy table, the microcontroller copies the required functions and variables into the system's RAM during the startup code. Since this table is automatically created by the compiler during the compilation process, this causes a problem. In general, most compilers only allow complete functions and variables to be allocated to the scratchpad memory. For this reason the copy table is extended by a script to include the corresponding entries. In addition, the required memory is reserved in the linker script by means of a dummy section, with an adjusted size according to the required memory.

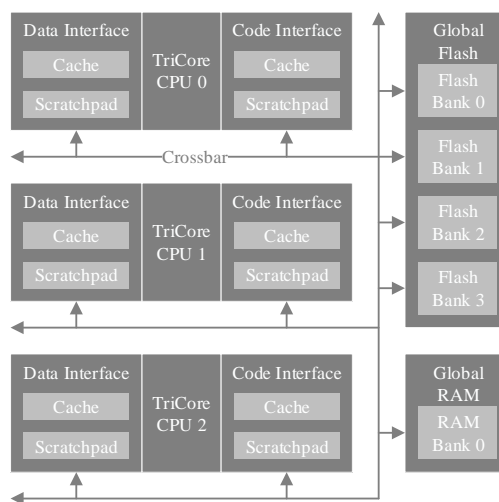
5 Experimental Results

The CoreMark of EEMBC in version 1.0 is used to prove the functionality. The reason for choosing the CoreMark as benchmark is that it is available in open source and has already been used in other publications. This circumstance makes it easier to relate the results of this article to the overall context of previous publications. Furthermore, the CoreMark was developed with a special focus on the evaluation of embedded microcontrollers and also offers multicore support.

The microcontroller used is an Infineon AURIX of the first generation, type TC29x with three processor cores, which is operated with a clock frequency of 200 MHz. The Infineon AURIX family uses the proprietary TriCore architecture, which is a modified Harvard architecture. Therefore each processor core has two interfaces, one for data and one for code. Each of these interfaces has two memories, one is a cache and one is a scratchpad RAM. The communication between the cores and the connection to the global memories is done via a crossbar. The basic structure can be seen in the figure 2. The memory sizes are described in the table 4.

For compiling the source code of the CoreMark, the TASKING compiler in version 6.2r1 for the TriCore architecture is used. The translation process of the this compiler is done in three steps. In the first step the C code is compiled into the Src format, which is then translated into the machine code by the assembler in the second step. At last the code is combined by the linker and allocated to the corresponding memories. The configurations for all steps used in this article can be taken from the table 5 [17].

3:10 Static Allocation of Basic Blocks



■ **Figure 2** Infineon AURIX TC29x - Basic Memory Layout [7].

For the manual validation of the generated results of the offline simulator all optimization levels of the compiler as well as the linker are deactivated. This configuration generates entry and exit jumps for loops, which are required for the separation of the basic blocks in this test series. Another special feature of the assembler configuration is the avoidance of relative jumps. With the settings made in this experiment, all jumps are implemented absolutely, which, as already described in chapter 4, is one way to separate the program code [17]. The previous approach to distributing source code to the available memory in the system is based on the call frequency and the memory consumption of the respective functions. For this purpose, the maximum number of times the individual functions are called during a defined period of time is determined. The call frequency is used to calculate their priority and the functions with the highest priority are allocated to the fast SRAM memory until it is filled completely. For this comparison, this method is taken as a reference. Therefore, in the first step, the maximum call frequency and its memory consumption is determined for each function of the CoreMark. The table 6 shows the number of calls at 20,000 iterations for each function, whereby the ordering already corresponds to the priorities calculated. For better overview, only the functions that are called during the benchmark measurement are listed.

In the table 7, the decomposition into basic blocks is carried out for three functions as an example. For the purpose of better overview, only those basic blocks are shown that are relevant for the proposed optimization. The remaining instructions within this

■ **Table 4** Infineon AURIX TC29x - Memory Dimensioning [7].

Category	Memory	Size/KB
Local	Data Scratchpad	240
	Data Cache	8
	Code Scratchpad	32
	Code Cache	32
Global	Flash	8192
	SRAM	32

■ **Table 5** TASKING Compiler 6.2r1 - Configuration.

Utility	Configuration
C-Compiler	-O0
Assembler	-OgS
Linker	-O0

■ **Table 6** EEMBC CoreMark 1.0 - Call Frequency and Memory Usage (Functions).

Function Name	Call Frequency	Size /Byte
ee_isdigit	78400000	44
core_state_transition	20480000	644
crcu8	11680008	92
crcu16	5840004	32
crc16	5240004	16
calc_func	4442360	240
cmp_idx	4161115	76
core_list_find	4120000	106
core_list_reverse	4080000	38
cmp_complex	2221180	44
crcu32	1280000	36
matrix_sum	320000	128
matrix_add_const	160000	74
core_bench_state	80000	422
matrix_test	80000	272
matrix_mul_matrix_bitextract	80000	162
matrix_mul_matrix	80000	146
matrix_mul_vect	80000	112
matrix_mul_const	80000	74
core_bench_matrix	80000	52
core_list_mergesort	60001	290
core_bench_list	40000	440
core_list_remove	40000	46
core_list_undo_remove	40000	40
Total		3626

function are only executed once per function call. In fact of this they cannot be used for further decomposition. For each of the basic blocks, the repetition during the entire benchmark execution is specified, which is calculated from the call frequency of the function and repetitions within the function. In addition, the memory requirements for each basic block in the scratchpad are also specified.

All basic blocks in the table 7 are loops which, due to their repetitions, cause a high computing load with low memory consumption. The basic blocks of the `matrix_sum` function are special because they are nested loops. However, since the analysis of the machine code is performed at the instruction level, the offline simulator detects this structure due to the different execution frequencies.

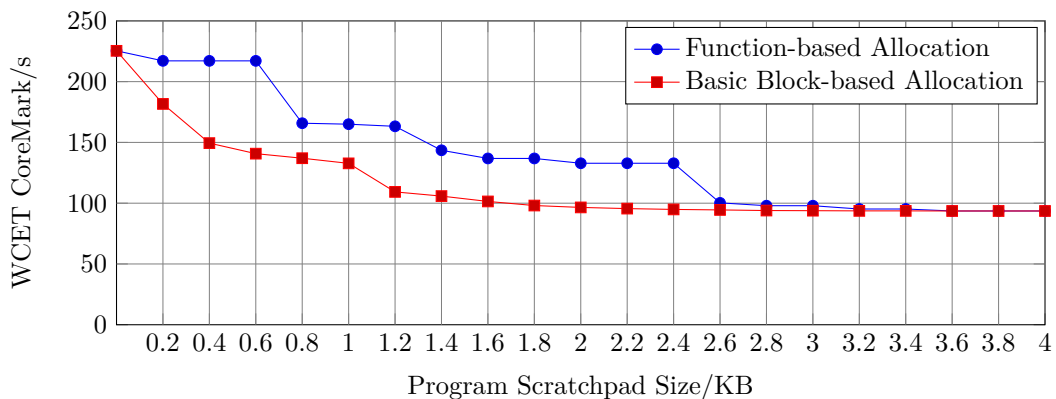
In the measurement, which is shown in figure 3, the presented method is set in relation to the previous approach. To evaluate the two allocation strategies, the CoreMark is executed with different scratchpad sizes. In the function-based approach, the functions are allocated to the fast memory in the order of their priority, which is shown in table 6. For a comparison of the two approaches, the distribution of the basic blocks is done according to a similar scheme. The advantage of the basic block allocation is the finer granularity. Due to the fact that functions often contain execution paths that are rarely processed, function-based allocation

3:12 Static Allocation of Basic Blocks

■ **Table 7** EEMBC CoreMark 1.0 - Call Frequency and Memory Usage (Basic Blocks).

Function Name	Basic Block Call Frequency Total	Size /Byte
crcu8	93440064	72
matrix_sum	2880000	14
core_bench_list	25920000	90
	4080000	192
	1120000	32
	1160000	32

is less efficient when using the small scratchpad memory. The difference is particularly noticeable at the beginning of the measurement series, where the execution time decreases significantly faster with the basic block-based method. The disadvantageous course of the function-based procedure is caused by the fact that the function `core_state_transition` is called frequently, but consumes a lot of memory. As a result, additional functions can only be allocated to the scratchpad once its size exceeds 800 bytes. Analogous to the staged sequence of the function-based allocation, a similar paragraph for the basic block-based procedure can be seen at 1 KB. This is a relatively large basic block that is unsuitable for further decomposition. By using better distribution algorithms, however, the process could be even better balanced. This should be evaluated in further investigations. Furthermore, it can be seen in the course that with increasing size of the scratch pad, the two methods converge, since all frequently used instructions are available in the fast memory. As the CoreMark with the current settings requires 3626 bytes of memory, the measurement results are identical from this scratchpad size on.



■ **Figure 3** Execution Time of the CoreMark 1.0 with different Scratchpad Allocation Methods.

6 Discussion

The main goal of this work is to use the local scratchpad memory more efficiently for the program code. Especially in systems with hard real-time requirements these memories provide a fast and deterministic way to increase the execution speed. However, the table 1 shows that the size of the flash memory is often larger than the program scratchpad by a factor of 85, as shown in the Infineon AURIX TC29x. As this relation already illustrates, the optimal use of

the scratch pads is therefore essential for reaching full performance. The method presented in this article shows a way how the low capacity of the scratchpad memory can be utilized better. The results on the AURIX platform in chapter 5 clearly show that, in contrast to the function-based approach, the use of basic blocks scales better with reduced memory sizes. However, this is at the cost of a slightly increased memory requirement in flash memory for all variants of separation. This is due to the fact that for absolute jumps, compared to the relative jumps, the complete address must be stored in memory. This restriction only exists for Instruction Set Architecture (ISA) that support relative jumps. A further problem is currently the analysis of the machine code. Particularly in safety-critical systems, interrupts and external signals are analyzed and it is necessary to react accordingly. Due to these unpredictable input variables, a realistic estimation of the maximum call frequency of functions is almost impossible. Furthermore, there are states in every system which are mutually exclusive, which effects the call frequency of functions. These complex correlations are extremely difficult to extract and require further investigations for a better evaluation of the WCET [5] [1]. In previous studies, the code is translated with the optimization level O0, whereby only absolute addresses for the jump instructions are generated. By this procedure optimizations are deactivated too, which have a significant influence on the execution speed. Therefore the goal is to consider further optimization levels and compilers in future extensions to achieve more realistic results. For the measurements performed so far, only the CoreMark from EEMBC was used, which only represents a small number of use cases. For this reason, a wide range of benchmarks will be ported in future research so that many different memory access types and patterns in the allocation can be analyzed and taken into account. In this context, the previous approaches to an optimized storage strategy will be ported and compared with the concept presented in this article. The intention is to give an overview which distribution method achieves the best results with which memory access pattern. A potential extension is the integration of caches in the distribution of the basic blocks. In contrast to the dynamic use of scratch pads, the address calculation for caches is done by the microcontroller. Thus, the overhead that would normally occur when calculating addresses in software can be avoided. However, the use of caches in real-time systems is associated with significantly higher effort in timing analysis, which in turn makes allocation more difficult [3]. Furthermore, support for other microcontroller architectures is planned for future work. The focus will be on multicore architectures, as these will benefit even more from local scratch pads due to concurrent access to shared memory. By optimizing the use of these core-exclusive memories, timing anomalies resulting from concurrent accesses could be prevented even more effectively. For this purpose, the analysis and prioritisation of the basic blocks would have to be extended accordingly [14].

References

- 1 N. Akram, Y. Zhang, S. Ali, and H. M. Amjad. Efficient task allocation for real-time partitioned scheduling on multi-core systems. In *2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pages 492–499, 2019. doi:10.1109/IBCAST.2019.8667139.
- 2 Oren Avissar, Rajeev Barua, and Dave Stewart. An optimal memory allocation scheme for scratch-pad-based embedded systems. *ACM Transactions on Embedded Computing Systems*, 1(1):6–26, November 2002.
- 3 Marco Caccamo, Marco Cesati, Rodolfo Pellizzoni, Emiliano Betti, Roman Dudko, and Renato Mancuso. Real-time cache management framework for multi-core architectures. In *Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE Computer Society, 2013.

- 4 Zhong-Ho Chen and Alvin Su. A hardware/software framework for instruction and data scratchpad memory allocation. *ACM Transactions on Architecture and Code Optimization*, 7, April 2010. doi:10.1145/1736065.1736067.
- 5 C. Dietrich, P. Wagemann, P. Ulbrich, and D. Lohmann. Syswct: Whole-system response-time analysis for fixed-priority real-time systems (outstanding paper). In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 37–48, 2017.
- 6 Infineon Technologies AG, 81726 Munich, Germany. *TriCore TC1.6P & TC1.6E Core Architecture*, February 2012.
- 7 Infineon Technologies AG, 81726 Munich, Germany. *AURIX TC29x B-Step User's Manual V1.3*, December 2014.
- 8 Philipp Jungklass and Mladen Berekovic. Effects of concurrent access to embedded multicore microcontrollers with hard real-time demands. In *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, pages 1–9, 2018.
- 9 Philipp Jungklass and Mladen Berekovic. Intercore-kommunikation für multicore-mikrocontroller. In *Tagungsband Embedded Software Engineering Kongress 2018*, 2018.
- 10 Philipp Jungklass and Mladen Berekovic. Memopt: Automated memory distribution for multicore microcontrollers with hard real-time requirements. In *2019 IEEE Nordic Circuits and Systems Conference (NORCAS)*, 2019.
- 11 C. Kachris, G. Nikiforos, V. Papaefstathiou, X. Yang, S. Kavadias, and M. Katevenis. Low-latency explicit communication and synchronization in scalable multi-core clusters. In *2010 IEEE International Conference On Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*, pages 1–4, 2010.
- 12 Yooseong Kim, David Broman, Jian Cai, and Aviral Shrivastava. Wcet-aware dynamic code management on scratchpads for software-managed multicores. *Real-Time Technology and Applications - Proceedings*, 2014:179–188, October 2014. doi:10.1109/RTAS.2014.6926001.
- 13 Isabelle Puaut and Christophe Pais. Scratchpad memories vs locked caches in hard real-time systems: a qualitative and quantitative comparison. *Institut de Recherche en Informatique et Systèmes Aléatoires - Publication Interne No 1818*, January 2006.
- 14 Selma Saidi, Rolf Ernst, Sascha Uhrig, Henrik Theiling, and Benoît Dupont de Dinechin. The shift to multicores in real-time and safety-critical systems. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*, pages 220–229. IEEE Press, 2015.
- 15 STMicroelectronics, Schiphol, Amsterdam, Niederlande. *SPC58xEx/SPC58xGx 32-bit Power Architecture microcontroller for automotive ASILD applications - Reference Manual*, August 2018.
- 16 V. Suhendra, T. Mitra, A. Roychoudhury, and Ting Chen. Wcet centric data allocation to scratchpad memory. In *26th IEEE International Real-Time Systems Symposium*, pages 10 pp.–232, 2005. doi:10.1109/RTSS.2005.45.
- 17 TASKING BV, Spoetnik 50, 3824 MG Amersfoort, Netherlands. *TASKING VX-toolset for TriCore User Guide*, ma160-800 (v6.2) edition, December 2016.
- 18 Texas Instruments Incorporated, Post Office Box 655303, Dallas, Texas 75265. *TMS320F2838x Microcontrollers Technical Reference Manual*, May 2019.
- 19 G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo. Global real-time memory-centric scheduling for multicore systems. *IEEE Transactions on Computers*, 65(9):2739–2751, 2016. doi:10.1109/TC.2015.2500572.


Event-Based Control Enters the Real-Time World: Perspectives and Pitfalls

Silvano Seva 

DEIB, Polytechnic University of Milan, Italy
silvano.seva@polimi.it

William Fornaciari 

DEIB, Polytechnic University of Milan, Italy
william.fornaciari@polimi.it

Alberto Leva 

DEIB, Polytechnic University of Milan, Italy
alberto.leva@polimi.it

Abstract

In the last years, event-based control techniques have been gaining a steadily increasing importance owing to the advantages they bring, such as reduced network traffic, low actuator wear, reduced energy consumption of the involved devices. Applying the event-based paradigm in the context of real-time control opens up new opportunities, but introduces new challenges as well. In this paper we provide an overview of both opportunities and challenges, outlining the major problems to be tackled and as a consequence future research directions.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture; Computer systems organization → Embedded systems

Keywords and phrases Real-time control, Wireless control, Event-based control, Cyber-physical systems, Industrial control networks, Industry 4.0

Digital Object Identifier 10.4230/OASICS.NG-RES.2021.4

1 Introduction and motivation

In the last years, the industrial environment has been characterised by the emergence of the “wireless factory” concept, fostered by paradigms like the Industry 4.0 (I40) and the Industrial Internet of Things (IIoT) ones. A prominent argument to promote the wireless factory idea is a strong reduction of cables, and therefore the mitigation of the related problems concerning for example – and most notably – installation and maintenance.

A key feature of the so emerging *scenario* is the use of wireless communication techniques also for control applications [4, 8, 12], where real-time requirements inevitably come into play. As such, the advantages of wireless communications come at a cost in terms of

1. tighter energy efficiency requirements, as in many cases cabling reduction and system layout reconfigurability call for battery-operated devices,
2. and increased criticality of band occupation, as one transmission medium can host a large number of applications, some of which requiring real-time guarantees in terms of latency, data rate, and so on.

Event-Based Control (hereinafter EBC for short) helps mitigating these issues by transmitting measurement and control data “only when needed”. The consequent energy saving is quite evident, as a notoriously battery-killing action for wireless devices is the exchange of data, due to the high power demand of the radio transceiver. Not equally obvious are the advantages as for band occupation, as these are in fact relevant only when slack reclamation techniques are employed, allowing other applications to reuse time slots temporarily made empty as some transmission “was not needed”.



© Silvano Seva, William Fornaciari, and Alberto Leva;
licensed under Creative Commons License CC-BY

Second Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2021).

Editors: Marko Bertogna and Federico Terraneo; Article No. 4; pp. 4:1–4:11

OpenAccess Series in Informatics



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

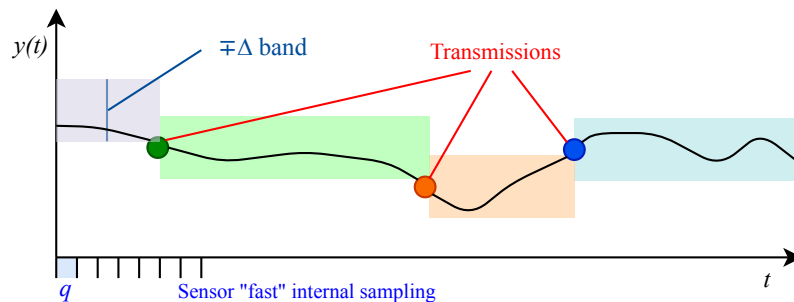
It is quite intuitive to notice, as our main point, that EBC has a potentially strong impact on real-time properties, no matter how formulated. As we shall discuss later on, giving up the periodic transmissions of fixed-rate digital control makes end-to-end latency (from the occurrence of a physical event in the controlled object till the physical reaction on that object) involve new phenomena, deeply intertwined with the synthesis of a control law.

Said otherwise, from a real-time perspective, EBC couples control algorithm, processor and network scheduling in a much tighter and more complicated manner than fixed-rate control does. As will be shown, for example, the idea itself of latency (a very typical subject of real-time requirements) needs extending to distinguish a “cyber” latency – the one addressed in the mainstream real-time literature – and a “cyber-physical” latency. The latter heavily depends not only on the workload required by the control law, but also on the way that law is conceived and tuned – an important subject in industrial control, see e.g. [5, 22] – and even on how the corresponding algorithms invoked by the event-generation mechanism. Needless to say, therefore, “real-time EBC” poses more than one challenging problem.

In this paper, continuing the research presented in [23], we analyse the real-time EBC *scenario* from the control theory and engineering viewpoint, but with an eye on the underlying architecture and technology as this is made necessary in the light of the considerations just reported, evidencing some of the new challenges arising when EBC systems coexist with other real-time applications and proposing possible solutions.

2 Event-based control in a nutshell

The core idea of EBC is to act on the controlled system not periodically, as in standard digital control, but “only when necessary”. Many meanings can be attributed to this idea of “necessity”, and since we are not providing here a complete treatise but just the bare necessary for this paper, we only describe the so-called “Send on Delta” one (SoD for short) as it is the most widely used in the applications. The very intuitive operation of a SoD sampler sensor, that triggers a control action by transmitting a new sample of the controlled variable when this “has varied enough”, is illustrated in Figure 1 and its caption.

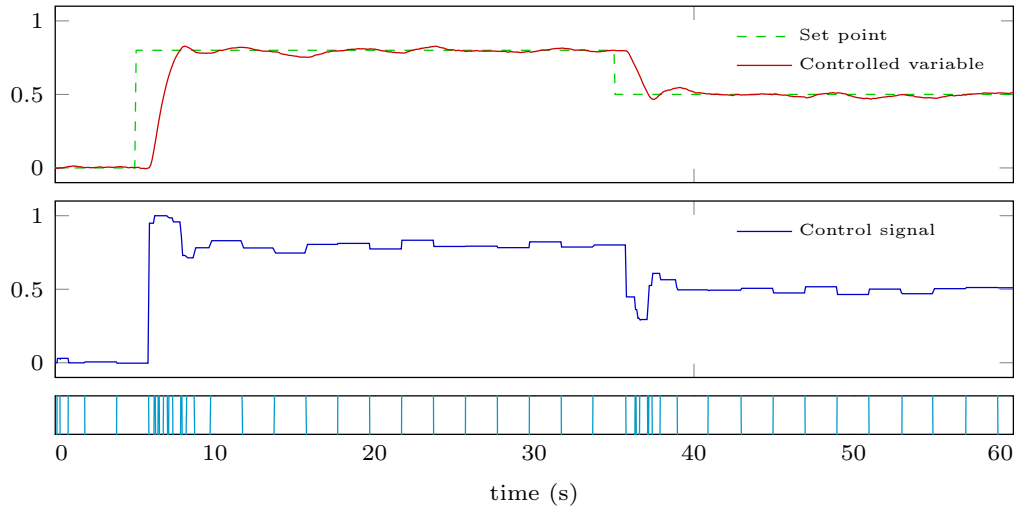


■ **Figure 1** Send on Delta (SoD) sampler operation in the periodic case – the sensor transmits a sample of the controlled variable $y(t)$ at the first integer multiple of the time quantum q where it differs in magnitude more than Δ from the last transmitted one.

In practice, one typically completes the mechanism in Figure 1 with a timeout, i.e., makes the sensor transmit a new sample unconditionally after a number N_{to} of *quanta* since the last transmission. Besides possible influences on stability that we are not discussing herein, this is an intuitively necessary means to watch over the sensor and check it stays alive.

Rigorously speaking, we are here limiting the scope to *periodic* EBC, as in the most general theory events are not constrained to occur at multiples of any time quantum (contrary to what we are assuming here right from the explanatory Figure 1). However, given the

clocked nature of any digital computing system, the periodic EBC context is general enough for us. The reader willing to deepen his/her knowledge can refer e.g. to the recent survey [2] and its huge bibliography. To give here just a rapid idea about how beneficial a properly designed EBC can be in terms of saved transmissions, Figure 2 reports a snapshot of the operation of a properly tuned EBC loop in the presence of typical measurement noise; some brief explanatory comments are provided in the caption.



■ **Figure 2** EBC in action: set point and controlled variable (top), control signal (centre) and events (bottom); the transmission saving with respect to fixed-rate control – where these would have to always occur at the fastest pace observed *and needed* during rapid signal variations – is apparent; periodic events when the system is (almost) at rest are due to SoD timeout, and their slow pace would not suffice to keep the loop under proper control in the face of significant *stimuli*.

As just said, however, EBC needs to be designed properly, or disasters can occur owing to the controller not acting timely. From the methodological standpoint, the main issue with EBC (also in the periodic case) is that the classical theory of fixed-rate sampled-data digital control ceases to apply, as the time span in between two subsequent control computations is not constant. In fixed-rate control, ensuring stable and correct operation of the closed-loop system ultimately calls for a proper choice of the sampling period, and there are established techniques for this purpose. In our EBC context, the role of the sampling period is played cooperatively by two actors, namely the time quantum q and the parameters pertaining to event generation (in SoD, the threshold Δ and the timeout N_{to}).

In extreme synthesis, assuming that the control design process follows the very common *modus operandi* to first determine a continuous-time controller and then its digital realisation, obtaining the latter in the (periodic) EBC context means

1. choosing the event generator parameters in such a way that events are generated frequently enough to ensure closed-loop stability and to not excessively deteriorate performance with respect to that ideally provided by the continuous-time controller, while at the same time avoiding too frequent spurious events (owing typically to measurement noise) to not excessively stress communication channels, controller and actuator;
2. converting the continuous-time controller to a discrete-time one suitable for updating its output and state in steps that are not uniformly spaced in time (although distances are quantised) to get the required control algorithm.

An important research domain concerns extending tuning techniques conceived for fixed-rate control realisations, which for industry standard controllers form a large *corpus* as shown e.g. in [22], to serve for EBC. But in addition to this research, that concentrates on synthesising the control algorithm rather than on the underlying architecture, for our purposes it is worth here noticing that EBC quite apparently revolutionises the usage of computing and network resources. An immediately noticeable fact is that the said usage is intrinsically non uniform and can exhibit hard-to-predict bursts, but if one focuses on the real-time context, there is more. Analysing the so emerging *scenario* is the subject of the following sections.

3 New challenges

As mentioned in the introduction, abandoning the “classical” fixed-rate digital control techniques in favour of EBC, alongside the previously outlined advantages, poses new challenges from the technological point of view. Some are in fact variations or enhancements of already known ones, for example in the domain of mixed criticality, while others are specific to the EBC context. Among the relevant ones, we evidence here a wider variability of the control latency, tight requirements in terms of network synchronisation and significant impacts on the schedulability of control tasks. In this section we analyse in detail these issues, compatibly with space limitations.

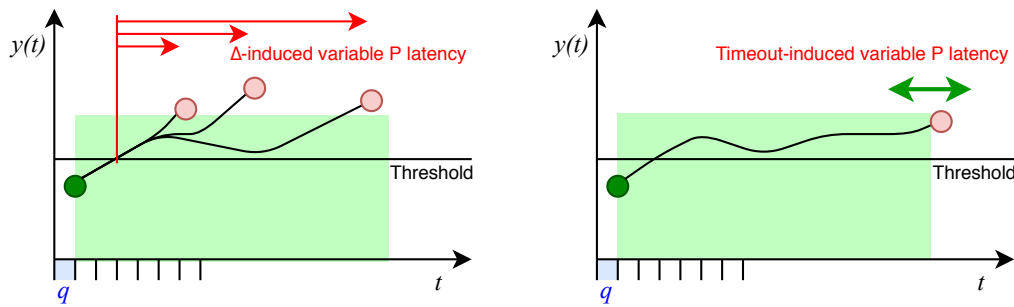
Control latency

Event-based control affects latency “as seen by the plant”, that is, the amount of time since some fact occurs till the controller reacts. Consider, for example, the case of a disturbance applied to the process input: if a fixed-rate control scheme is used, the controller surely reacts to the disturbance starting at the first control step immediately following the time instant when the disturbance effect becomes visible on the controlled variable. From this we have that the control latency, as seen by the process, is bounded and at most equal to the width of one additional time step with respect to the process natural response delay. In the case of EBC, on the contrary, the controller reaction time is affected by various factors, like the mechanism used to generate events, the presence of a timeout, and others.

Following this brief analysis, we can introduce the concept of “cyber-physical control latency”, explicitly denoting the influence on the controller response time of two different components, one attributable to both the network and computing infrastructures - the cyber part - and the other one - the physical one - to the process structure, through the event-triggering mechanism.

In the case of a fixed-rate control system, the contribution to the overall control latency of the physical component is negligible with respect to the cyber one, since the controller will react to any change in the process output within, at most, one time step, irrespectively of the process internal state, the extent of output and measurement noises and so forth. From the point of view of control system’s stability and performance, this represents an good situation: once the controller step rate has been properly computed and estimates of the amount of network-induced delay are available, the impact of the control latency on phase margin (one of the key parameters which allow to describe both control stability and performance) can be straightforwardly computed and, eventually, compensated with *ad-hoc* techniques.

On the other hand, with event-based control systems the situation is more difficult: here, the physical component has a significant contribution on the overall amount of the control latency, also increasing its temporal variability. This physics-induced latency is ascribable



■ **Figure 3** Example of variable physical latency in the case one has to detect when some variable $y(t)$ exceeds a threshold: P latency variability can come from both the SoD threshold Δ (left) and the timeout N_{to} (right); in either case, moreover, the process state has influences the P latency variability by dictating – together with the inputs – the variation rate of $y(t)$.

to different phenomena, two of which are now described: one of them, depicted in the left part of Figure 3, is the rate of variation of the process output variable, determining the time span between the instant when a perturbed movement of the process output arises due to an external disturbance and the one when a corrective control action is applied, this one triggered by the process output variable crossing the threshold inside the event generator.

The other one, shown in the right part of Figure 3, is constituted by process perturbations causing its output variable to have an erratic behaviour, but inside the event-triggering dead band: given that the event-triggering threshold has to be chosen appropriately also considering the maximum tolerable deviation from the reference signal, such phenomena could anyway be detrimental in terms of control performance. In this situation, the amount of time before a corrective control action is applied depends on the characteristics of the timeout mechanism - which is always a good idea to have, to avoid running the closed-loop system in open loop for excessive amounts of time - periodically and unconditionally triggering a new control action: in the worst case, the one when the perturbation begins immediately after the last timeout-triggered control action has been applied, the control latency amounts to one full timeout period. This problem is apparently an EBC peculiarity, relevant for real time control as it pertains to latency.

Synchronisation

When an EBC scheme is employed, communications between sensors and controller and/or controller and actuators can be infrequent, however their timing must be always precise enough to allow them to happen properly. Although in a fixed-rate control application there are plenty of signal fronts to keep all the network nodes synchronised, in an event-based one this may well not be the case: consider, in this respect, that the time between two subsequent control computations, also in the fixed-rate case, is in general far larger than the time scale of network protocol transactions.

Another fact worth noticing, as testified by the major industry standards [17, 14, 28]¹, is that Time Division Multiple Access (TDMA) or even polling-based access schemes are widely adopted when dealing with communication networks for control applications. Since in periodic EBC adopting TDMA (polling would make no sense) implies that a slot must be

¹ There are exceptions like e.g. CAN, but mostly limited to vehicular applications and wired settings, where the problem of minimising the radio listening consumption does not exist.

reserved for each *possible* transmission, bandwidth saving can be jeopardised unless some slack reclamation technique is in place for non time-critical traffic to opportunistically occupy unused slots.

The consequent need for the originators of that traffic to carry out a reliable clear channel assessment apparently tightens the synchronisation needs. Also, in fixed-rate control missing one sample of the controlled variable is an information loss immediately cured by the next one. In EBC there can be no “next one” for a long time, causing highly undesired behaviours. As such, this is an example of pre-existing problem exacerbated by EBC.

Schedulability of control tasks

When dealing with task schedulability in a real-time system, the presence of event-based control tasks in a task set can have non negligible effects in terms of schedule feasibility and system overload. Before better analysing these effects, let us consider, for a comparison, the case of a task set containing only fixed-rate control tasks: here, to each controller (or group of controllers) corresponds a periodic task with a fixed and known execution period and, from the schedulability point of view, the problem is the “classic” one of finding a feasible schedule for a given set of tasks. On the other hand, event-based techniques, due to their underlying principle of acting on the plant “only when needed”, pose some concerns for what regards the scheduling of CPU resources: each control task maintains its requirements in terms of guaranteed periodic execution, but its contribution to the overall CPU load is not constant, since it will be executed only sporadically and for a limited amount of time.

When EBC tasks are present, then, the feasibility of a task schedule enters the “cyber-physical” domain: the same physical phenomena that affect the control latency, shown in the previous point, can strongly determine the time distribution of an EBC task. In this regard, we also introduce the concept of “event storm”: it may happen, during the normal operation of the system, that a large number control tasks, if not *all* of them, are simultaneously woken up due as a consequence of some external physical phenomena causing the generation of controller events. When an event storm occurs, CPU utilisation factor suddenly spikes up to a value which can be greater than 100%, causing some tasks to miss their deadline.

The key point of such a phenomenon lays in the fact that an event storm is caused by some event happening in the physical world, and these events are substantially not predictable: this means that new techniques to ensure proper task schedulability in presence of EBC applications have to be devised. Similarly to what can be done with TDMA slots, however, the aperiodic nature of the EBC tasks brings into play also some advantages, allowing to achieve better CPU and network utilisation by re-assigning the otherwise unused CPU time.

Summing up, the EBC context complicates *a priori* architecture sizing, because utilisation bursts can be much higher than in fixed-rate control, and the inherently sporadic (but possibly transiently concentrated) events, make it hardly possible to figure out hyper-periods to ground task allocation upon. Said otherwise, EBC strongly affects – to not say just breaks – the customary connection between real-time and periodic tasks, turning the exception of a latency-constrained non-periodic task – at least as long as control is the purpose – into the normal case.

4 Proposals

The sporadic behaviour of EBC tasks, occupying network and computing resources only when some corrective control action has to be applied, can be favourably exploited to improve the performances of the computing infrastructure they are based on, for example exploiting the

approach proposed in [20]. In this section we develop our treatise by analysing the possible earnings that can be gathered when dealing with schedulability of either CPU or network resources.

Concerning CPU utilisation, the underlying principle of EBC stating that the controller is run “only when needed” results in the fact that is no more necessary to have an always running periodic task for each controller: although the constraint of ensuring enough CPU time to each (event-based) control task is still present - to not affect the stability and performance of the closed-loop system - there is now the possibility to re-allocate the otherwise unused CPU time to other tasks whenever the corresponding controller is in idle state. This, evidently, is not the case with fixed-rate controllers: since, in this case, the closed-loop system to which they belong is not designed to be run in open loop - or, at least, the safety of such a behaviour is not guaranteed -, a value for the control action has to be computed at each time step, even when the control error is zero. However, as described in Section 3, event-based control tasks can also have detrimental effects on the feasibility of a CPU schedule in occurrence of event storms. In this respect, two different approaches are possible: the first one, conservative, proceeds by considering all the event-based controllers as fixed-rate ones and then requiring to the schedulability analysis to guarantee that the overall task set never reaches a CPU utilisation factor greater than 100%. With this approach, an event storm simultaneously activating all the control tasks does not have a destructive impact on the overall system performance, while keeping the possibility of reusing empty CPU slots for other non-critical tasks.

The second approach, applicable when some degradation in the control performance is tolerable, calls for a subdivision of the event-based control tasks in two sets, whether a degradation of control performance is acceptable or not. The outcomes of this subdivision, then, provide some room to safely undersize the required computing resources by making the scheduling system structured such that, in case of CPU overload - i.e. due to an event storm - the tasks associated to the control loops accepting a performance degradation are not always assigned their CPU time, leaving computing resources to the other, more critical, tasks.

The same principle of re-allocating the otherwise unused time slots can be applied to network resources too. Data transmission on control networks is often managed through TDMA schemes so as to have an almost constant and known in advance (cyber) control latency: since each control task is uniquely assigned a transmission slot, all the variable delays introduced by collisions and access contention are automatically removed. Especially with battery-operated wireless devices, using event-based control techniques coupled with a well-synchronised TDMA scheme allows to reach remarkable energy savings, enhancing the devices’ operating time and reducing the maintenance costs. These advantages, however, are counterbalanced by a poor utilisation of the wireless transmission medium: the fact that each transmission slot is uniquely assigned to a control task means that it cannot be reused whenever the event-based control system is in the idle state, a situation which happens quite often and for significant periods of time. To avoid wasting this precious bandwidth, various techniques can be used. One possibility is implementing a slack reclamation technique, making each network node capable of detecting, for each time slot, if the slot assignee is effectively transmitting data: if not, that otherwise empty slot can be reused for other transmissions. For such a mechanism to be feasible, however, a very precise synchronisation among all the nodes is required, such that the residual synchronisation error is well below the width of a time slot. Another observation has to be made about which kind of data can be effectively exchanged through the re-used slots: given how this mechanism works, data exchange through these slots is affected by a wide temporal variability in terms of available

bandwidth and latency, both depending on how many slots are effectively available in a transmission round. Given these characteristics, then, data exchanged through otherwise empty slots cannot have strict requirements in terms of real-time performance: this, however, leaves room for all the data transmissions serving ancillary functionalities always present in an industrial plant, such as non-critical monitoring of process parameters, and signalling and so on.

Another way for a more efficient utilisation of the available bandwidth from the event-based control tasks is abandoning the TDMA scheme in favour of a CSMA (Carrier Sense Multiple Access) one: instead of having uniquely-assigned time slots also for the event-based control tasks, resulting in the aforesaid bad utilisation of the available bandwidth, the communication mechanism can be made such that all the packets containing data for event-based control are exchanged through opportunistic time slots, where more than one network node attempts to transmit its payload. It is not a mystery that this scheme has effects on the control latency: lost the guarantees given by a TDMA scheme, the value of the network-induced latency has to be determined on a probabilistic basis in terms of “worst case cyber-physical latency”. This expression poses the accent on the fact that the overall transmission latency is both due to network characteristics and physical phenomena affecting the access contention to the transmission medium and the generation of events starting a data transmission. If we go deeply into the problem, however, we have to observe that control applications are more tolerant to latency issues with respect to others like, for example, signal processing ones: while in the second case a non-tight latency bound can disrupt the final results (think to the case of an FFT task: a change in the sample arrival rate shifts the resulting spectrum), in a control loop the value of latency bounds appears more indirectly, in terms of stability degree, absence of oscillations, small response time deterioration, and so forth. A hybrid approach is also possible, by subdividing a transmission round in two parts: the first one managed through a TDMA scheme for all the event-based control tasks whose execution is somehow critical and requires for strict bounds on the variability of transmission latency and the remaining one accessed with a CSMA technique, for all the tasks able to tolerate a wider variability of the transmission latency.

5 Related work

On the systems and control front, EBC dates back to pioneering works such as [3], where the idea of lightening the control network load was proposed and developed on a significantly heuristic basis. Methodological studies on the properties of such newly introduced loops came in the following decades and yielded neat results, see e.g. [19, 27], while the influences of the EBC framework on the synthesis of controllers came into the research *arena* [15, 16]. The presence of EBC also required new models for the network as seen “externally” by controllers in terms of dynamic systems [30]. At the same time, pilot and research-targeted realisations started appearing – see for example [10] and several analogous works – paving the way to addressing real industrial cases [9, 8].

As already said, a recent and complete survey on the overall subject is [2], while another one more geared to industrial applications can be found in [11]. Considering this huge research *corpus*, the main conclusions for the purpose of our research is that powerful analysis methods are nowadays available for EBC, but the intertwined effects of event triggering mechanism and control algorithm are still being explored, so that in fact a systematic approach to tuning event-related parameters together with those of a control law is still in its infancy – especially if industry-standard solutions are sought, specifications are tight, the cost of resources makes rules out over-provisioning *a priori*, or any combination of the above.

On the technological side, the problem of resource scheduling in presence of both period and aperiodic tasks has been already analysed in the past - see, for example, works like [18] and [13]. On this basis, a good starting point for future works aiming to both improvements to EBC task schedulability and a more efficient utilisation of CPU time left free by idle EBC tasks is constituted by the current state of the art on the schedulability of sporadic tasks, with works like [7, 21, 6].

From the network scheduling point of view, instead, EBC represents a completely new use case for the current state of the art, for a variety of reasons. The first one is the management of latency: while in fixed rate control the addition of one time step to the estimated - or measured - cyber latency is a correct overbound for the total cyber-physical latency, with EBC this assumption cannot be held true anymore due to the presence of a strong physical component influencing the total control latency. To this extent, Figure 3 shows two notable cases. Coming to the transmission protocols for control networks, EBC ideally requires for schemes allowing for non periodic data transmission but without queues: an unusual requirement for a transmission protocol from both the cyber and cyber-physical points of view. The *rationale* for such a requirement resides in the fact that, for a control system, a measurement sample arriving “late” to the controller becomes useless, since it conveys information about an old state of the process, which in the meantime has surely changed. From this point of view, a network protocol without queues dropping the old packets of favour of newer ones represents a more effective situation. Given such a *scenario*, the presence of a valid scheme for the synchronisation among the network nodes allows for the implementation of suitable protocols. To this aim, works like [26] and its successive extensions [24, 25] provide a valuable ground for future developments.

On a wider perspective, the current research activity concentrates on the IIoT paradigm and on event-based wireless communication [1, 29] but, to the best of the authors’ knowledge, the problem of cyber-physical latency is hardly mentioned, let alone of a formalisation of the connected problems.

6 Conclusions

Using event-based techniques for process control brings in numerous advantages, especially when battery-operated wireless sensors and actuators are involved. However, from the technological point of view, applying such techniques in a real-time context poses new and important challenges: in this paper we have briefly analysed these issues with a focus on both computational and network resources, showing the impact of EBC tasks on the feasibility of a CPU schedule and the existing trade-offs between energy and bandwidth saving. Another relevant point is constituted by control latency, which becomes more dependent on the physical phenomena inherent with the process being controlled. To this aim, we have introduced the concept of “cyber-physical control latency” and detailed the nature of its cyber and physical components.

Following the analysis of these new challenges, we have outlined some solutions allowing for a safe implementation of EBC techniques in a real-time context, also pointing towards a better utilisation of both CPU and network resources through slack-reclamation techniques. Future work in this direction points towards a deeper analysis of the issues here presented followed by the devise of adequate methods for reclaiming the otherwise unused resources.

References

- 1 G. Aceto, V. Persico, and A. Pescape. A survey on information and communication technologies for Industry 4.0: state-of-the-art, taxonomies, perspectives, and challenges. *IEEE Communications Surveys & Tutorials*, 21(4):3467–3501, 2019.
- 2 E. Aranda Escolástico, M. Guinaldo, R. Heradio, J. Chacon, H. Vargas, J. Sánchez, and S. Dormido Bencomo. Event-based control: A bibliometric analysis of twenty years of research. *IEEE Access*, 8:47188–47208, 2020.
- 3 K.E. Årzén. A simple event-based PID controller. In *Proc. 14th IFAC World Congress*, volume 18, pages 423–428, Beijing, China, 1999.
- 4 S.A. Ashraf, I. Aktas, E. Eriksson, K.W. Helmersson, and J. Ansari. Ultra-reliable and low-latency communication for wireless factory automation: from LTE to 5G. In *Proc. 21st IEEE International Conference on Emerging Technologies and Factory Automation*, Berlin, Germany, 2016.
- 5 K.J. Åström. Event based control. In A. Astolfi and L. Marconi, editors, *Analysis and Design of Nonlinear Control Systems*, pages 127–147. Springer, Berlin, 2008.
- 6 S. Baruah, V. Bonifaci, G. D’Angelo, H. Li, A. Marchetti Spaccamela, S. Van Der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *Journal of the ACM*, 62(2):14:1–14:33, 2015.
- 7 S. Baruah and N. Fisher. The partitioned scheduling of sporadic real-time tasks on multiprocessor platforms. In *Proc. 2005 International Conference on Parallel Processing Workshops*, pages 346–353, Oslo, Norway, 2005.
- 8 T. Blevins, D. Chen, S. Han, M. Nixon, and W. Wojsznis. Process control over real-time wireless sensor and actuator networks. In *Proc. 17th IEEE International Conference on High Performance Computing and Communication*, 2015.
- 9 T. Blevins, M. Nixon, and W. Wojsznis. Event based control applied to wireless throttling valves. In *Proc. 1st International Conference on Event-based Control, Communication, and Signal Processing*, Kraków, Poland, 2015.
- 10 J. Chacón, J. Sánchez, L. Yebra, A. Visioli, and S. Dormido. Experimental study of two event-based PI controllers in a solar distributed collector field. In *Proc. 12th European Control Conference*, pages 626–631, Zürich, Switzerland, 2013.
- 11 M. Dotoli, A. Fay, M. Miśkiewicz, and C. Seatzu. Advanced control in factory automation: a survey. *International Journal of Production Research*, 55(5):1243–1259, 2017.
- 12 Y. Wei et Al. RT-wifi: Real-time high-speed communication protocol for wireless cyber-physical control applications. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 140–149, Nashville, Tennessee, 2013.
- 13 G. Lipari and G. Buttazzo. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *Journal of Systems Architecture*, 46(4):327–338, 2000.
- 14 S.M. Hassan, R. Ibrahim, K. Bingi, T.D. Chung, and N. Saad. Application of wireless technology for control: A WirelessHART perspective. *Procedia Computer Science*, 105(supplement C):240–247, 2017.
- 15 D. Henriksson and A. Cervin. Optimal on-line sampling period assignment for real-time control tasks based on plant state information. In *Proc. 44th IEEE Conference on Decision and Control*, pages 4469–4474, Seville, Spain, 2005.
- 16 B. Hensel, J. Ploennigs, V. Vasyutynskyy, and K. Kabitzsch. A simple PI controller tuning rule for sensor energy efficiency with level-crossing sampling. In *Proc. 9th IEEE International Multi-Conference on Systems, Signals and Devices*, pages 1–6, Chemnitz, Germany, 2012.
- 17 D. Jansen and H. Buttner. Real-time ethernet: the EtherCAT solution. *Computing and Control Engineering*, 15(1):16–21, 2004.
- 18 K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of period and sporadic tasks. In *Proc. 12th IEEE Real-Time Systems Symposium*, pages 129–139, San Antonio, TX, USA, 1991.

- 19 J. Lunze and D. Lehmann. A state-feedback approach to event-based control. *Automatica*, 46(1):211–215, 2010.
- 20 M. Maggio, F. Terraneo, and A. Leva. Task scheduling: A control-theoretical viewpoint for a general and flexible solution. *ACM Transactions on Embedded Computing Systems*, 13(4):1–22, 2014.
- 21 M. Marouf, L. George, and Y. Sorel. Schedulability analysis for a combination of non-preemptive strict periodic tasks and preemptive sporadic tasks. In *Proc. 17th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1–8, Kraków, Poland, 2012.
- 22 A. O’Dwyer. *Handbook of PI and PID controller tuning rules*. Imperial college press, London, United Kingdom, 2009.
- 23 S. Seva, C.E. Lukaschewsky Mauriziano, W. Fornaciari, and A. Leva. A low energy FPGA platform for real-time event-based control. In *Proc. 1st Workshop on Next Generation Real-Time Embedded Systems*, Bologna, Italy, 2020.
- 24 F. Terraneo, A. Leva, S. Seva, M. Maggio, and A. V. Papadopoulos. Reverse flooding: Exploiting radio interference for efficient propagation delay compensation in WSN clock synchronization. In *2015 IEEE Real-Time Systems Symposium*, pages 175–184, Rome, Italy, 2015.
- 25 F. Terraneo, P. Polidori, A. Leva, and W. Fornaciari. TDMH-MAC: Real-time and multi-hop in the same wireless MAC. In *Proc. 39th IEEE Real-Time Systems Symposium*, pages 277–287, Nashville, TN, USA, 2018.
- 26 F. Terraneo, L. Rinaldi, M. Maggio, A. V. Papadopoulos, and A. Leva. FLOPSYNC-2: Efficient monotonic clock synchronisation. In *Proc. 35th IEEE Real-Time Systems Symposium*, pages 11–20, Rome, Italy, 2014.
- 27 Y. Wang, W.X. Zheng, and H. Zhang. Dynamic event-based control of nonlinear stochastic systems. *IEEE Transactions on Automatic Control*, 62(12):6544–6551, 2017.
- 28 Xuepei Wu and Lihua Xie. On the wireless extension of PROFINET networks. In *Proc. 2019 IEEE VTS Asia Pacific Wireless Communications Symposium*, pages 1–5, Singapore, 2019.
- 29 H. Yang, K. Zhang, K. Zheng, and Y. Qian. Leveraging linear quadratic regulator cost and energy consumption for ultra-reliable and low-latency IoT control systems. *IEEE Internet of Things Journal*, 7(9):8356–8371, 2020.
- 30 X.M. Zhang, Q.L. Han, and Bao-Lin B.L. Zhang. An overview and deep investigation on sampled-data-based event-triggered control and filtering for networked systems. *IEEE Transactions on Industrial Informatics*, 13(1):4–16, 2016.

M2OS-Mc: An RTOS for Many-Core Processors

David García Villaescusa 

University of Cantabria, Santander, Spain
garciavd@unican.es

Mario Aldea Rivas 

University of Cantabria, Santander, Spain
aldeam@unican.es

Michael González Harbour 

University of Cantabria, Santander, Spain
mgh@unican.es

Abstract

A current trend of industrial systems is reducing space, weight and power (SWaP) through the allocation of different applications on a single chip. This is enabled by the continued improvement of semiconductor technology which allows the integration of multiple cores in a single processor chip, as the processors are prevented to continue increasing their clock rate due to the “power-wall”. The use of Commercial-Off-The-Shelf (COTS) multi-core processors for real-time purposes presents issues due to the shared bus used to access the shared memory. An alternative to the use of multi-core processors are the many-core processors with tens to hundreds of processors in the same chip, using different scalable ways to interconnect their cores. This paper presents the adaptation of the M2OS Real-Time Operating System (RTOS) and its simplified Ada run-time for mesh-based many-core processors. This RTOS is called M2OS-mc and has been tested on the *Epiphany III* many-core processor (referred in this paper simply as *Epiphany*), a many-core which has 16 cores connected by a Network-on-Chip (NoC) consisting of a 4x4 2D mesh. In order to have a synchronized way to send messages between tasks through the NoC independently of the core where they are being executed, we provide *sampling port* communication primitives.

2012 ACM Subject Classification Computer systems organization → Real-time operating systems

Keywords and phrases M2OS, Many-Core, Real-Time, Parallella, Epiphany, Network-on-Chip, Operating System, RTOS

Digital Object Identifier 10.4230/OASICS.NG-RES.2021.5

Funding FEDER funds (AEI/FEDER, UE) under Grant TIN2017-86520-C3-3-R(PRECON-I4).

David García Villaescusa: Graduate Grant Program of the University of Cantabria, Spanish Government.

1 Introduction

In the past, the evolution of processors was mostly related to frequency improvement but since the processors reached a power consumption too high to dissipate, the designers have been improving the processor’s performance by having more processing cores executing in the same chip: the multi-core era begun. Multi-cores provide not only better energetic efficiency but a greater performance-per-cost. The applications can be parallelized, being divided into sections that can be executed simultaneously, to take advantage of all of cores in the same multi-core chip.

Multi-core processors with few cores have a shared bus for communications among their cores and the shared memory, as shown in Figure 1. When the number of cores increases, the shared bus becomes a bottleneck and different communication strategies are used. In these processors with a high number of cores, called many-cores, a common alternative is the use of a Network-on-Chip (NoC) based on a 2D mesh, as shown in Figure 1 for the *Epiphany*



© David García Villaescusa, Mario Aldea Rivas, and Michael González Harbour; licensed under Creative Commons License CC-BY

Second Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2021).

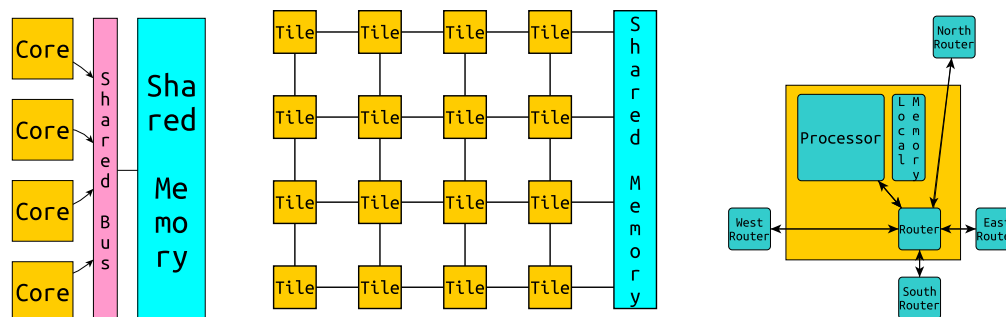
Editors: Marko Bertogna and Federico Terraneo; Article No. 5; pp. 5:1–5:13



OpenAccess Series in Informatics

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

processor [16]. This network has a predictable delay on the communications between two neighbor cores. It also requires less wires than a shared bus and its power consumption is linear with the number of cores. Other NoC topologies have been proposed in other architectures like the torus of Kalray [6] or the ring used by Intel [12].



■ **Figure 1** Generic multi-core topography (left), *Epiphany*'s topography (middle) and eCore architecture (right).

The many-core mesh connects the tiles using the typical configuration shown in Figure 1, with the core, its local memory and a router connecting the tile's core with the neighbor routers in the mesh.

As the many-core processors seem to be the immediate future of COTS processors, there is a need to have suitable software platforms that allow the execution of hard real-time applications on such architectures. In order to fulfill that requirement, this paper presents the port of the M2OS RTOS to *Epiphany*, a many-core which has 16 cores connected by a 4x4 2D mesh, with each core having a 32-kilobyte local memory.

M2OS is a small and efficient real-time kernel supporting the non-preemptive one-shot task model [17] [1]. The implementation presented in this paper follows the multikernel paradigm [4], with a different RTOS image running in each core. The small footprint of M2OS makes its adaptation to this architecture feasible. M2OS has been ported to the *Epiphany* many-core processor, in such a way that the resulting M2OS-mc is aimed at running in any other 2D-mesh many-core platform with a minimum amount of changes. We have developed a mechanism to allow exchanging messages between the different tasks of the system, independently of the core where they are executing. This mechanism uses the *sampling port* implementation presented at Section 5.

M2OS-mc, as well as M2OS, is written in Ada as this language has specialized features supporting low-level real-time, safety-critical and embedded systems programming”¹.

A typical Ada application executed on M2OS in the *Epiphany* processor is composed of several tasks running in the different cores, with one or more tasks in each core. Tasks in the same core are executed under the one-shot non-preemptive scheduling policy implemented by M2OS. The tasks' messages between tasks allocated in different cores will travel through the NoC.

To take advantage of the parallel architecture of the underlying many-core, the response to an external event is typically performed by several tasks (running on the same or different cores) which are activated in sequence (a task's predecessor activates the next task in the sequence and provides its input data).

¹ https://en.wikibooks.org/wiki/Ada_Programming

M2OS alongside M2OS-mc are available on-line at the website ², and are distributed under a GPL license.

The paper continues by analyzing the related work in Section 2. In Section 3 the *Epiphany* processor is introduced. Section 4 discusses the properties of M2OS and exposes its adaptation to a many-core. Synchronization and message exchange between tasks are described and evaluated in Section 5. Finally, Section 6 shows the paper conclusions and future work. An appendix is included with the *sampling port* interface and a consumer-producer test code.

2 Related work

The NoC concept is not something new. It was already presented by Benini in 2002 [5] and it soon got the real-time community's attention [10].

There have been some projects that brought many-core platforms and real-time operating systems together:

- P-SOCRATES [9], whose purpose is to develop an entirely new designed framework from the conceptual design of the system functionality to its physical implementation, to facilitate the deployment of standardized parallel architectures in all kinds of systems. The tasks follow the OpenMP task model. From that project Erika3, from Erika Enterprise [7], has been developed. It is an RTOS that uses a single image per computer cluster and has a memory footprint of just a few kB. This RTOS runs in the Kalray MPPA-255.
- eSol has developed a many-core real-time OS called eMCOS [8] with a distributed micro-kernel architecture implemented. This micro-kernel is allocated at the cores with minimal functions while more advanced operations are performed through server cores. It claims to support a wide variety of architectures in which *Epiphany* is not included.
- Altamary ported RTEMS [2] for the *Epiphany* processor on a *Parallella* board similar to the one used in this project. As we will see later, the *Parallella* board has both local memory for each core and a global shared memory. This modified version of RTEMS can be placed in both types of memory. When RTEMS is placed in the shared memory the system is significantly slower than when it is placed in local memory. However, in the latter case it only leaves 5kB for the applications. RTEMS is a relatively complex RTOS that implements several scheduling algorithms.

Several studies have been done for theoretical 2D mesh NoCs [11] [13]. These studies perform scheduling analysis using theoretical many-core processors. With the availability of an RTOS such as M2OS-mc more realistic scheduling analysis could be carried out in the future.

3 Epiphany

The *Epiphany* processor is integrated in the *Parallella* [14] development board which has the size of a credit card and needs just 5W to work. Apart from the *Epiphany* processor it also has an ARM dual-core processor (Zynq), which is the central processor on the *Parallella* board. It combines an ARM dual-core Cortex-A9 with Xilinx programmable logic. *Zynq* has an Ubuntu adaptation (Parabuntu) that is used as operating system. The Parabuntu OS is used to send the executables to the *Epiphany* cores (*eCores*) and it also starts each *eCore* execution.

² <https://m2os.unican.es>

The *Epiphany* processor is a many-core designed by Adapteva with 16 cores connected by a NoC placed in a 4x4 2D mesh as Figure 1 shows, where every square is a tile that contains a router connected to the neighbor tiles and the execution core. Each core of the *Epiphany* is an *eCore*, whose architecture is also designed by Adapteva, that executes its instructions in order, with a frequency of 600 MHz. It consists of an integer ALU, floating-point unit, a debug unit an interrupt controller, a general purpose program sequencer and a 64-word general purpose register file. Each core has 32kB of local memory. The architecture is supported by the GCC compiler and has libraries for OpenMP and MPI.

The design could grow as it has been shown with a 1024 cores version [15]. Unfortunately, the *Epiphany V* is not available in any development board.

Any *eCore* can access the local memory of the rest of the *eCores* using a range of special global addresses. The synchronized message interface explained in Section 5 takes advantage of that. An *eCore*'s local memory can be written and read without any hardware limitations but the memory size. The process of writing to another *eCore*'s memory is 8 times faster than reading. This is due to the fact that the *Epiphany* processor has independent networks for reading and writing between cores and the one used for writing is much faster.

The *Parallella* board has a shared memory that can be accessed by the *eCores*. This memory access is much slower than a memory access between *eCores* so this method is considered too slow, although it could be useful for other functionalities as it is shown in Section 4.5.

It can be said that the *Parallella* board is a good platform for experimenting with the development of RTOS for mesh-based many-cores.

4 M2OS

M2OS [17] [1] is a small real-time operating system that allows running multitasking applications in small microcontrollers with scarce memory resources. This is the case of the *Epiphany* processor, where each of its *eCores* has a 32 kB local memory.

M2OS implements a simple scheduling policy based on non-preemptive one-shot tasks, which requires a very small memory footprint. This policy allows the same stack area to be shared by all the tasks and, consequently, the system only needs to allocate a stack area large enough to fit the largest task stack.

M2OS is written in Ada and it is the base of a simplified Run-Time System for the GNAT Ada compiler. This RTOS has been developed for *Arduino Uno* and *STM32F4*. M2OS is intended to be easily ported to different platforms. All the hardware dependent part is encapsulated in a Hardware Abstraction Layer (HAL), which is the only code that has to be modified to port the kernel to a new platform.

The new HAL written for the *Epiphany* uses the *Epiphany Library* (`e-lib`) to perform low-level functions that are specific to this architecture, such as interrupt and timer management. An Ada interface has been implemented for those functions of the `e-lib` library that are required by the M2OS kernel. As a result, the `e-lib` library must be included in the linking instruction.

A deeper analysis the *Epiphany*'s implementation will now be exposed.

4.1 Building and loading the application

M2OS is an RTOS written in Ada so taking advantage that the *eCore* architecture is supported by the GNU compiler collection `gcc`, we have compiled it for the *eCore*, therefore achieving support for the Ada and C languages.

One executable file is generated for each of the *eCores*. The executable file generated includes both the M2OS and the user code. This executable must be loaded into the different *eCores* by the *Zynq* processor. Each *eCore* starts its execution individually when the *Zynq* processor sends the corresponding signal.

The linker script used to build the M2OS applications places all the data and code in the local memory of the *eCores*.

A set of scripts has been produced to automate the cross-compilation of the applications that will run under M2OS and to load the generated executables to the *Parallela* board.

4.2 HAL

The HAL of M2OS has been implemented for the *eCore*'s architecture. This layer includes the basic support for context switch, interrupt and hardware timer handling.

- **Context switch.** Under the simple scheduling policy implemented in M2OS the context switch only requires resetting the Stack Pointer to the base position and setting the program counter.
- **Interrupts.** The global interrupts can be enabled, disabled and checked for their status. This implementation was developed thanks to the `e-lib` library.
- **Core identification.** The `e-lib` provides primitives for core identification. This service was not included in the M2OS HAL because it is specific of architectures with more than one core. It is part of M2OS-mc now.
- **Spinlock.** The `e-lib` provides spinlocks to be used among the different cores for non-blocking mutual exclusion synchronization (called “mutex” in the `e-lib` terminology).
- **System timer.** It follows the “ticker” approach that requires the periodic programming of a hardware timer. In our implementation one of the two *eCore*'s timers is used to generate an interrupt each 1ms. This interrupt is used to account for the system time. The timer, driven by a 600MHz clock, can only be programmed in one-shot mode, which requires it to be reprogrammed at each execution of the interrupt handler.
- **System clock.** It stores a counter of each system timer interrupt in a 32 bit integer. It has a 1 ms resolution.
- **High precision clock.** Our implementation of M2OS in *Epiphany* provides a high precision clock by reading the actual value of the hardware timer. This clock has a precision of 1.667 ns and is suitable for intervals up to 1ms (when the system timer resets the value).

4.3 Clock synchronization

Epiphany applications are launched from the *Zynq* processor by loading the application code corresponding to each *eCore* and sending, sequentially, the start signals to the different *eCores* of the system. In consequence, each *eCore* starts its execution at a different instant. For a real-time operating system the timer synchronization at each component is very useful for time awareness, to avoid having significant timer gaps between tasks executing at different *eCores*. For this purpose M2OS synchronizes all the timers during the start up of the RTOS.

This clock synchronization process is conducted by a master *eCore*, which is the last one to be started (the 0x0 *eCore* in the current version). Upon initialization, every other *eCore* has to wait for the master to send a message containing the value of its timer. After receiving this synchronization message, each *eCore* updates its own timer with the received value plus the time the message needs to be generated and transmitted through the NoC and the time spent by the *eCores* executing the required instructions.

4.4 Performance metrics

Different tests are done to measure various mechanisms implemented in a single *eCore*, which are time measurement, context switch, application size and mutex usage.

The tests in this section execute the required actions a thousand times. This number was chosen to achieve short execution times in which there was no interference from the system timer, which produces an interrupt every millisecond.

- **Reading the clock.** Knowing the time needed for reading the clock is required to get more precise times for the rest of the tests. The result of this test is shown in Table 1. Since the minimum time to read the clock is 81 cycles, from this point we have subtracted this value from all the measurements involving the clock.
- **Mutex.** The time required to lock or release a mutex is constant, as shown in Table 1.
- **Context Switch.** The time needed to perform a context switch on an *eCore* is calculated with an M2OS generic test run in the M2OS-mc. The results when using a `delay until` operation are shown in Table 1. The context switch has been tested in depth divided in activation and suspension tests, as shown in Table 2. The set of tests consists of:
 - **Activation tests.** Latency since one task opens a suspension primitive and suspends itself until the activated task executes (suspension object, protected object entry without parameters or protected object entry with one parameter).
 - **Suspension tests.** Latency since a task suspends itself until another task executes. Times are measured for different suspension primitives (delay until, suspension object, protected object entry without parameters or protected object entry with one parameter).
- **Application size.** The output of the `size` linux command for 2 applications, one with 6 periodic tasks and another one with 2 periodic tasks is shown in Table 3. Each of those tasks just put a message on the console, set a boolean to true, calculate the time of the next activation and delay until that time. It can be seen that the amount of tasks has a small impact on the size of the application.

■ **Table 1** Latencies for reading the clock and operating a mutex.

Test	Max	Min	Avg
Clock Read	81 cycles	81 cycles	81 cycles
Lock Mutex	211.7 ns	211.7 ns	211.7 ns
Release Mutex	133.4 ns	133.4 ns	133.4 ns

■ **Table 2** Context switch tests.

Activation Tests	Max	Min	Avg
Suspension object	593.5 ns	593.5 ns	593.5 ns
Protected object entry without parameter	698.5 ns	698.5 ns	698.5 ns
Protected object entry with one parameter	736.8 ns	736.8 ns	736.8 ns
Suspension Tests	Max	Min	Avg
Delay until	596.8 ns	596.8 ns	596.8 ns
Suspension object	345.1 ns	345.1 ns	345.1 ns
Protected object entry without parameter	540.1 ns	433.4 ns	453,4 ns
Protected object entry with one parameter	548.4 ns	548.4 ns	548.4 ns

■ **Table 3** Results of the size command for two applications with 6 and 2 tasks, respectively.

text	data	bss	dec	hex	filename
10914	1244	528	12686	318e	six_tasks
10226	1244	208	11678	2d9e	two_tasks

4.5 Console

The console output in M2OS is performed by the console driver, which has to be implemented for each architecture M2OS is ported to. In the *Parallella* board the system console is managed by the *Zynq* processor. The *eCores* do not have direct access to the system console.

The solution adopted is that every *eCore* writes in a reserved local memory space that is read by the *Zynq* processor. The reserved memory space of every *eCore* will be used as a circular buffer into which `Put_Line` commands write text. The buffer is designed such that a line is never divided. When the final address of the designated area is reached, the next write operation will be done at the beginning of its reserved region, erasing the oldest line or lines. In that way we emulate the behavior of a console. No console input has been implemented.

The console output is thereby printed in the user's terminal by a specifically-developed software executed at the *Zynq*, which shows the *eCores* consoles content by reading the fixed local memory of each *eCore* where the console driver writes the desired console output.

In case the *Zynq* tries to read from the memory assigned to a non-initialized *eCore* it gets content lacking any meaning but the system will not crash.

5 Inter-task messages

A typical application running on the many-core processor consists of a number of end-to-end flows (e2e). Each e2e is a set of tasks (in the same or different *eCores*) that responds to the same periodic or sporadic event. These tasks must have a way to communicate between them and a mechanism for waking up the next task in the flow at the end of each execution. This requires a way to communicate between tasks in different *eCores* in a synchronized manner.

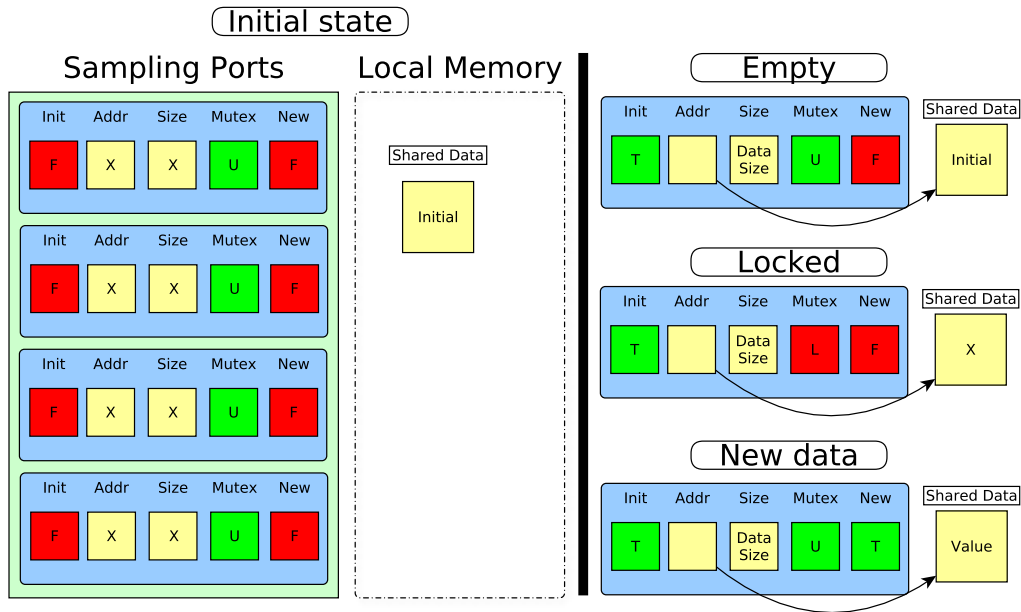
The chosen synchronization mechanism is inspired in the ARINC's *sampling ports* (SP) and the Ada implementation performed by Garrido [3]. Only one message can be held at a determined *sampling port*. This means that any new message written in a *sampling port* will overwrite the previous stored message. Several *sampling ports* can be mapped at the same *eCore*.

In order to avoid a task from blocking the *eCore* we advise application developers to make a periodic polling using a `delay until` operation that allows other tasks to use the *eCore* while polling the *sampling port*. The polling period must be considered to calculate the response time. An example of this approach is shown in Listing 3 in Appendix A.

5.1 Implementation

The implemented synchronized messages take advantage of the fact that the local memory of any *eCore* is accessible from every other *eCore*.

In our implementation there is a fixed number of *sampling ports* per *eCore*, configurable at system configuration time. They are implemented as an array of `sampling port` records that are placed in the same predefined memory location of each *eCore*'s local memory. Each *sampling port* is identified by the identifier of the core where it is allocated and its index in the array.



■ **Figure 2** *Sampling port states.* We show all the *sampling ports* in the initial state (left) and a single *sampling port* at each of the other states (right).

The *sampling port* record includes the followings fields:

- **Init.** Boolean used to know if the *sampling port* has been initialized or not.
- **Mutex.** The spinlock to protect the content.
- **Size.** The size of the protected content.
- **Addr.** Where the content is located using a global memory address of the *eCore*'s memory.
- **New.** Flag to know whether the content has been modified since the last read operation.
- **Core.** Core where the SP is initialized. This is required by the mutex.

M2OS initializes the **Mutex** of each *sampling port* and sets its **Init** field to **False**. The access to the *sampling port* is protected by the spinlock. Any operation on the *sampling port* must lock the spinlock and release it afterwards.

To wake-up a task, the *sampling port* used for that purpose must be polled periodically by the task, waiting for the **New** field to be set to **True**.

This implementation requires 32 bytes per *sampling port*, in addition to a user-allocated memory area for the message.

5.2 Interface

The interface developed for the *sampling ports* is shown at Listing 1 in Appendix A. It provides the following functions:

- **Init_Sampling_Port.** This function returns the identifier of the initialized *sampling port*. It receives the host *eCore*, the *sampling port* index at the *eCore* and the address and size of the shared data. The function initializes the port causing its state to transition from the “initial” state to the “empty” state, passing through the “locked” state. All these states are shown in Figure 2. If the *sampling port* is already initialized it returns `Null_SP_Id`. No writing or reading operation over the *sampling port* are allowed until this initialization is performed, and consequently those operations will return an error indication in that case.

- **Get_Sampling_Port.** A function that returns the identifier of a *sampling port* given the host *eCore* and its index identifier. If the *sampling port* has not been initialized it returns `Null_SP_Id`.
- **Write_Sampling_Port.** A procedure that writes the shared data of the *sampling port* parameter. The content is marked as new. The content to be copied into the shared data is located at the address given as a parameter and has the size also indicated as a parameter. This procedure performs the transition of the *sampling port* from the “empty” (or “new data”) state to the “locked” state and then to the “new data” state. The states are shown in Figure 2. If the write operation has been successful the output parameter is set to `True`. It will be set to `False` otherwise (when trying to write into an “initial” state *sampling port* or when the size of the item is larger than the size defined at the *sampling port*).
- **Read_Sampling_Port.** This procedure copies the shared content of a *sampling port* into an address supplied as a parameter. The field `New` is set to `False`. During the read operation the *sampling port* is in the “locked” state passing to the “empty” state once the operation finishes. If the read has been successful the output parameter is set to `True`, it will be set to `False` otherwise (when trying to read from an uninitialized *sampling port*, when the size defined at the *sampling port* is larger than the size of the item where the value is going to be written, or when no data is written yet).

5.3 Usage example

The functionality of the *sampling ports* is described with an example that follows the typical producer/consumer pattern. The producer is a periodic task that produces a data item and writes it in a *sampling port*. This *sampling port* is used by the consumer task to wait for new data and process it.

The consumer, shown at Listing 2 in Appendix A, declares and initializes the data to be shared and then initializes the *sampling port*. Thereafter it continuously iterates waiting for a new value to arrive at the *sampling port* and consuming it. The wait operation is implemented as a periodic poll of the *sampling port*.

The producer, shown at Listing 3 in Appendix A, must wait until the *sampling port* is initialized. This is done by periodically polling the *sampling port* until it is initialized. Then it periodically iterates producing a new data item and writing it to the *sampling port*. New content is written to the *sampling port* regardless of whether the previous content has been read or not.

In this example it can be noticed that the consumer is the one that initializes the *sampling port*, because it is located in its own *eCore* and, since writing through the NoC is eight times faster than reading, we decided to implement the most efficient model.

5.4 Tests

M2OS has a battery of tests that have been successfully passed for the *Epiphany* architecture. These tests include stack management, scheduling, timing events and task handling. To this battery set, we have added other tests such as measuring the latency of sending messages between tasks through the network. These tests will be analyzed in the following lines.

Table 4 shows the time required to execute the operations described in Section 5.2. The times for the `Get_Sampling_Port` and the `Write_Sampling_Port` operations are measured for a *sampling port* allocated in an *eCore* at one hop distance from the calling task. It can be seen that both writing and reading a *sampling port* have a linear increment in relation to the size of the message.

■ **Table 4** *Sampling port* latencies for different message sizes.

Latency of	Max	Min	Avg
Init_Sampling_Port	556.8 ns	556.8 ns	556.8 ns
Get_Sampling_Port	146.7 ns	146.7 ns	146.7 ns
Write_Sampling_Port (8 bytes)	596.8 ns	596.8 ns	596.8 ns
Read_Sampling_Port (8 bytes)	873.5 ns	873.5 ns	873.5 ns
Write_Sampling_Port (40 bytes)	1247 ns	1237 ns	1237 ns
Read_Sampling_Port (40 bytes)	2364 ns	2364 ns	2364 ns

The process of sending a message requires locking and unlocking a remote mutex. The latencies therefore depend on the latencies in the NoC, which in turn depend on the distance, in hops, the message needs to travel. A test sending an 8-byte message comparing different locations and different hop distances is shown in Table 5, where it can be seen that the timing is also linear in relation to the distance. We have performed the same test for reading a message from another *eCore* placed at different distances.

■ **Table 5** *Write_Sampling_Port* and *Read_Sampling_Port* latencies for different hop distances in the NoC.

Write	Max	Min	Avg	Read	Max	Min	Avg
1 hop	596.8 ns	596.8 ns	596.8 ns	1 hop	873.5 ns	873.5 ns	873.5 ns
2 hops	631.8 ns	631.8 ns	631.8 ns	2 hops	958.5 ns	958.5 ns	958.5 ns
3 hops	666.8 ns	666.8 ns	666.8 ns	3 hops	1044 ns	1044 ns	1044 ns
4 hops	701.8 ns	701.8 ns	701.8 ns	4 hops	1129 ns	1129 ns	1129 ns
5 hops	736.8 ns	736.8 ns	736.8 ns	5 hops	1214 ns	1214 ns	1214 ns
6 hops	771.8 ns	771.8 ns	771.8 ns	6 hops	1299 ns	1299 ns	1299 ns

In order to complete the performance analysis of the *sampling ports*, a round-trip scenario has been created. This scenario involves two cores with one task and one *sampling port* in each. The task in the first core sends a message to the *sampling port* allocated in the second core. A task waiting for that message sends it back to the *sampling port* allocated in the initial core. Table 6 shows the latencies measured for this round trip for different distances between the cores. In this test, when either of the two tasks has to wait for a message sent through a *sampling port* it does so by spinning continuously, so that there are no context switches or delays.

We can see that there is a dependency on the size of the message, and that the dependency on the distance between the *eCores* is linear.

■ **Table 6** Round-trip latencies for messages of 8 bytes (left) and 40 bytes (right).

8 bytes	Max	Min	Avg	40 bytes	Max	Min	Avg
1 hop	2794 ns	2322 ns	2777 ns	1 hop	5846 ns	5568 ns	5615 ns
2 hops	2787 ns	2366 ns	2776 ns	2 hops	5841 ns	5579 ns	5603 ns
3 hops	2904 ns	2479 ns	2882 ns	3 hops	6051 ns	5720 ns	5768 ns
4 hops	2944 ns	2559 ns	2936 ns	4 hops	6160 ns	5786 ns	5845 ns
5 hops	3062 ns	2656 ns	3036 ns	5 hops	6161 ns	5770 ns	5866 ns
6 hops	3132 ns	2731 ns	3097 ns	6 hops	6190 ns	5891 ns	5903 ns

6 Conclusions and future work

We have ported the M2OS to the *Epiphany* many-core, with an implementation designed to allow the adaptation for other many-cores. We have also implemented a synchronization mechanism inspired on the ARINC-653 *sampling ports*.

The resulting port, called M2OS-mc, has passed the whole battery of tests included in M2OS, so we can conclude the port works. Tests to check the functionality and performance of the developed synchronization mechanism have also been developed and the system has passed them. The performance metrics done for the *sampling ports* shows an acceptable efficiency.

At the current stage, M2OS-mc is a fully functional prototype however, our intention is to continue its development in several aspects:

- Develop a new kind of synchronization port inspired on the ARINC *queuing ports*. This kind of port implements a fixed-size queue of data and allows a consumer task to suspend on an empty port until new data is written, which would avoid the need for polling.
- Develop a system model that allows us to perform a schedulability analysis of the applications using M2OS-mc for *Epiphany*.
- Develop task allocation algorithms that allow us to improve the response times of the end-to-end flows that form the applications.
- Extend the application model by allowing some *eCores* to execute parallel workloads programmed with OpenMP while other *eCores* execute the real-time tasks.

References

- 1 Mario Aldea-Rivas and Héctor Pérez-Tijero. Proposal for a new ada profile for small micro-controllers. *Ada Lett.*, 38(1):34–39, July 2018. doi:10.1145/3241950.3241955.
- 2 Hesham Almatary. *Operating System Kernels on Multi-core Architectures*. PhD thesis, University of York, January 2016. URL: <http://etheses.whiterose.ac.uk/12959/>.
- 3 Jorge Garrido Balaguer, Juan Rafael Zamorano Flores, and Juan Antonio de la Puente Alfaro. Arinc-653 inter-partition communications and the ravenscar profile. *Ada Letters*, 35(1):38–45, 2015. URL: <http://oa.upm.es/42418/>.
- 4 Andrew Baumann, Paul Barham, Rebecca Isaacs, and Tim Harris. The multikernel: A new os architecture for scalable multicore systems. In *22nd Symposium on Operating Systems Principles*. Association for Computing Machinery, Inc., October 2009. URL: <https://www.microsoft.com/en-us/research/publication/the-multikernel-a-new-os-architecture-for-scalable-multicore-systems/>.
- 5 Luca Benini and Giovanni Micheli. Networks on chips: A new soc paradigm. *Computer*, 35:70–78, February 2002. doi:10.1109/2.976921.
- 6 Benoundefinedt Dupont de Dinechin and Amaury Graillat. Network-on-chip service guarantees on the kalray mppa-256 bostan processor. In *Proceedings of the 2nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems*, AISTECS '17, page 35–40, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3073763.3073770.
- 7 Erika Enterprise. Erika3. [Online; accessed 29-January-2020].
- 8 eSol. Scalable and High-performance Real-Time OS available for various types of processors. [Online; accessed 29-January-2020].
- 9 Xiongli Gu, Peng Liu, Mei Yang, Jie Yang, Cheng Li, and Qingdong Yao. An efficient scheduler of rtos for multi/many-core system. *Computers & Electrical Engineering*, 38(3):785–800, 2012. The Design and Analysis of Wireless Systems and Emerging Computing Architectures and Systems. doi:10.1016/j.compeleceng.2011.09.009.

- 10 Salma Hesham, Jens Rettkowski, Diana Goehringer, and Mohamed A. Abd El Ghany. Survey on real-time networks-on-chip. *IEEE Trans. Parallel Distrib. Syst.*, 28(5):1500–1517, May 2017. doi:10.1109/TPDS.2016.2623619.
- 11 Leandro Soares Indrusiak. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Systems Architecture*, 60(7):553–561, 2014. doi:10.1016/j.sysarc.2014.05.002.
- 12 James Jeffers and James Reinders. *Intel Xeon Phi coprocessor high performance programming*. Newnes, 2013.
- 13 Borislav Nikolic, Sebastian Tobuschat, Leandro Indrusiak, Rolf Ernst, and Alan Burns. Real-time analysis of priority-preemptive nocs with arbitrary buffer sizes and router delays. *Real-Time Systems*, 55, June 2018. doi:10.1007/s11241-018-9312-0.
- 14 Andreas Olofsson. Parallella reference manual.
- 15 Andreas Olofsson. Epiphany-v: A 1024 processor 64-bit risc system-on-chip. *arXiv preprint arXiv:1610.01832*, 2016.
- 16 Andreas Olofsson, Tomas Nordström, and Zain Ul-Abdin. Kickstarting high-performance energy-efficient manycore architectures with epiphany. *CoRR*, 2014. arXiv:1412.5538.
- 17 Mario Aldea Rivas and Hector Perez Tijero. Leveraging real-time and multitasking ada capabilities to small microcontrollers. *Journal of Systems Architecture*, 94:32–41, 2019. doi:10.1016/j.sysarc.2019.02.015.

A Listings

```

type SP_Index is range 0 .. Sampling_Ports_Per_Core-1;
type SP_Id is private;

function Init_Sampling_Port (C : in E_Lib.Core; Id : in SP_Index;
    Addr : in System.Address; Size : Interfaces.Unsigned_32)
    return SP_Id;

function Get_Sampling_Port (C : in E_Lib.Core; Id : in SP_Index)
    return SP_Id;

function Write_Sampling_Port (SP : in SP_Id;
    Orig : in System.Address; Orig_Size : in Interfaces.Unsigned_32)
    return Boolean; -- Successful

procedure Read_Sampling_Port (SP : in SP_Id;
    Dest : in System.Address; Dest_Size : in Interfaces.Unsigned_32;
    Successful : out Boolean; Is_New : out Boolean);

```

■ Listing 1 *Sampling port* Interface


```

task Consumer is
  -- Declare and initialize data
begin
  SP := Init_Sampling_Port (Current_Core, SP_Index, Data'Addr, Data'
    Size);
  loop
    loop
      Read_Sampling_Port(SP, Data'Address, Data'Size, Success, Is_New
        );
      if not Success then
        -- Error;
      end if;
      exit when Is_New;
      Next_Polling_Period := Next_Polling_Period + Period;
      delay until Next_Polling_Period;
    end loop
    -- Consume data
  end loop;
end Consumer;

```

■ Listing 2 Consumer

```

task Producer is
begin
  loop
    SP := Get_Sampling_Port (Core_Target, SP_Index);
    exit when SP /= Null_SP_Id;
    Next_Polling_Period := Next_Polling_Period + Period;
    delay until Next_Polling_Period;
  end loop;
  loop
    -- Produce new data
    Write_Sampling_Port(SP, Data'Address, Data'Size, Success);
    if not Success then
      -- Error
    end if;
    Next_Activation := Next_Activation + Task_Period;
    delay until Next_Activation;
  end loop;
end Producer;

```

■ Listing 3 Producer

