Third Workshop on Next Generation Real-Time Embedded Systems

NG-RES 2022, June 22, 2022, Budapest, Hungary

Edited by
Marko Bertogna
Federico Terraneo
Federico Reghenzani

Editors

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

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

Federico Reghenzani
Politecnico di Milano, Italy
federico.reghenzani@polimi.it

ACM Classification 2012
Computer systems organization → Real-time systems; Computer systems organization → Embedded and cyber-physical systems

ISBN 978-3-95977-221-1

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/978-3-95977-221-1.

Publication date
June, 2022

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.

Digital Object Identifier: 10.4230/OASIcs.NG-RES.2022.0

ISBN 978-3-95977-221-1 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Invited Paper

Regular Papers

Third Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2022).
Editors: Marko Bertogna, Federico Terraneo, and Federico Reghenzani
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume collects the papers presented at the third edition of the Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2022). The workshop is co-located with the 2022 edition of the HiPEAC conference and was held on June 22, 2022 in Budapest, Hungary.

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 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 sofware and technologies (e.g. RTOSs, hypervisors, separation kernels, virtualization) for parallel and heterogenous 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 third edition of the workshop three regular papers were accepted, each of which receiving between two and three peer reviews. In addition, we are glad to have an invited paper by Giorgio Buttazzo titled “Can We Trust AI-Powered Real-Time Embedded Systems?”. We would like to thank the authors of the NG-RES 2022 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, Federico Terraneo, and Federico Reghenzani

Third Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2022).
Editors: Marko Bertogna, Federico Terraneo, and Federico Reghenzani
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

General Chair

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

Program Chair

- Federico Terraneo, Politecnico di Milano, Italy
- Federico Reghenzani, Politecnico di Milano, Italy

Program Committee

- Alberto Leva, Politecnico di Milano, Italy
- Ashik Ahmed Bhuiyan, University of Central Florida, United States
- Benny K. Akesson, TNO, Netherlands
- Christine Rochange, Institut de Recherche en Informatique de Toulouse, France
- Filip Markovic, Mälardalen University, Sweden
- Jaume Abella Ferrer, Barcelona Supercomputing Center, Spain
- Lucia Lo Bello, University of Catania, Italy
- Luís Almeida, Universidade do Porto, Portugal
- Jürgen Teich, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
- Marco Solieri, Università di Modena e Reggio Emilia, Italy
- Roberto Cavicchioli, Università di Modena e Reggio Emilia, Italy

Third Workshop on Next Generation Real-Time Embedded Systems (NG-RES 2022).
Editors: Marko Bertogna, Federico Terraneo, and Federico Reghenzani
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Can We Trust AI-Powered Real-Time Embedded Systems?

Giorgio Buttazzo ✉
Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy

Abstract

The excellent performance of deep neural networks and machine learning algorithms is pushing the industry to adopt such a technology in several application domains, including safety-critical ones, as self-driving vehicles, autonomous robots, and diagnosis support systems for medical applications. However, most of the AI methodologies available today have not been designed to work in safety-critical environments and several issues need to be solved, at different architecture levels, to make them trustworthy. This paper presents some of the major problems existing today in AI-powered embedded systems, highlighting possible solutions and research directions to support them, increasing their security, safety, and time predictability.

2012 ACM Subject Classification Computer systems organization

Keywords and phrases Real-Time Systems, Heterogeneous architectures, Trustworthy AI, Hypervisors, Deep learning, Adversarial attacks, FPGA acceleration, Mixed criticality systems

Digital Object Identifier 10.4230/OASIcs.NG-RES.2022.1

Category Invited Paper

1 Introduction

Embedded computing platforms are becoming more complex every day to manage the increasing computational load generated by emerging applications, as autonomous vehicles (cars, trains, drones, aircrafts), advance robotic systems, intelligent appliances, and so on. Such systems are equipped with a variety of sensors that produce a large amount of data, hence demanding for real-time processing and high-performance computing. To provide the required computational power, computer architectures are evolving towards heterogeneous platforms that integrate on the same board, or even on the same chip, multicore processors of different types, field programmable gate arrays (FPGAs), general purpose graphics processing units (GPGPUs), and special co-processors optimized for executing operations on tensors, as tensor processing units (TPUs).

Although new tools and libraries are becoming available every year, developing safety-critical applications on top of such heterogeneous platforms, while providing the required guarantees, is quite difficult due to a number of non-trivial problems. The following list presents just a few of such problems, related to the use of artificial intelligence (AI) in safety-critical applications with real-time constraints.

- The interaction among the various components through the shared resources available on the computing platform (as buses, memories, and I/O devices) generates a significant amount of interference, introducing large and unpredictable delays on the computational activities. Such a large variability in responding to external events makes it is very difficult to provide timing guarantees on the application behavior. This is also a serious problem for the certification of safety-critical software components.
- Programming modern heterogeneous platforms requires a deep knowledge of low-level details of the architecture, which prolongs the developing and testing times.

- Distributing the computational activities in an optimal way between hardware accelerators and processors is not trivial, but it can make a huge difference in the overall system performance, as well as in satisfying the real-time application constraints.
- Deep neural networks (DNNs) are commonly developed and inferred by means of state-of-the-art frameworks (e.g., Tensorflow, Caffe, and PyTorch), which greatly simplify the implementation of new models. Unfortunately, however, none of the current frameworks is specifically optimized to be used in safety-critical environments, nor capable of providing bounded response times. This prevents their use in real-time applications like autonomous driving, where DNNs should have a highly predictable behavior, not only in the functional domain, but also in the time domain, responding within specific deadlines.
- The use of deep learning algorithms and the related frameworks increases the software attack surface, posing serious security issues for the overall system. This problem is exacerbated by the fact that such frameworks usually run on top of rich operating systems, as Linux, which are more vulnerable to cyber attacks.
- In spite of their excellent capabilities in perception tasks, deep neural networks have been shown to be prone to adversarial attacks, i.e., malicious inputs with imperceptible perturbations that force a neural network to produce a wrong output with a high confidence score.
- Similar threats derive from inputs that significantly differ from the distribution of the training set. Predicting the behavior of a neural network on such inputs is not easy and, in some cases, the network could also respond with a wrong output with a high confidence score.
- Finally, since the behavior of a neural network is not explicitly programmed, but encoded in a huge number of parameters, interpreting the output of a neural model and deciding whether its prediction can be trusted is a challenging task.

To address the issues described above, a lot of research is being devoted to support the development of AI-powered applications on top of heterogeneous platforms for safety-critical real-time systems.

The remainder of the paper is organized as follows: Section 2 discusses the problems and preliminary solutions related to architecture issues; Section 3 presents problems and promising solutions related to security issues; Section 4 describes the major threats caused by the use of AI algorithms and some solutions aimed at mitigating them; Section 5 proposes an architectural approach to address all the issues discussed above; and Section 6 states the conclusions and outlines some promising research lines.

2 Architecture issues

To be used in real time, the inference of modern DNN models requires hardware acceleration. This can be achieved by exploiting modern heterogeneous computing platforms equipped by GPUs or programmable hardware, as FPGA. This section discusses the main problems related to such computing platforms and presents existing solutions to them.

2.1 General platform issues

To cope with the different computational requirements of real-time applications, modern heterogeneous computing platforms integrate different processing elements, as multi-core processors of different types, general purpose GPGPUs, FPGAs, and tensor processing units.

The concurrent accesses to shared devices existing on such architectures, as buses, memory controllers, and high-level caches, create a significant interference on the computations, introducing long and variable delays in application tasks.

For instance, Cavicchioli et al. [12] observed significant and variable delays when using GPU acceleration on heterogeneous embedded platforms due to the contention occurring on shared memory, especially for memory-intensive GPU tasks. Restuccia et al. [29] identified some anomalous situations that can arise in an AXI bus arbiter in FPGA-based SoC and proposed a reservation mechanism to prevent this phenomenon and restore fairness during bus transactions.

A timing analysis has also been proposed [28] to bound the execution of periodically-invoked hardware accelerators in nominal conditions. This analysis can be used to configure a latency-free hardware module named AXI Stall Monitor (ASM) to detect and safely solve possible stalls during AXI bus transactions. Efforts have also been devoted to analytically bound the delay experienced by AXI bus transactions issued by hardware accelerators on FPGA [30].

Hardware acceleration typically involves memory-intensive computations. Therefore, an accurate control of the memory traffic is crucial to achieve predictability in the execution of HW-tasks.

Pagani et al. [26] proposed a bandwidth reservation mechanism for AXI-based transactions on FPGAs able to control the bus traffic generated by hardware accelerators. The mechanism, named Memory Budget and Protection Unit (MBPU), aims at "shielding" hardware accelerators from excessive or unpredictable memory interference. MBPUs are installed between AXI master ports and the interconnect, enforcing a given budget of memory transactions within a periodic interval of time. Budgets are recharged in a periodic fashion and are configurable from the CPU via memory-mapped registers. MBPUs also protect the system from unrestricted accesses to memory by HW-tasks: this is accomplished by masking the accesses that fall outside a set of configurable memory address spaces.

2.2 GPU-related issues

Today, the most common way for accelerating DNNs is by executing them on a GPU-based platform. This solution has two main advantages: (i) the response time can be reduced by two orders of magnitude and (ii) the development is supported by standard frameworks. However, GPUs also have disadvantages. First, they are closed systems and multiple tasks are scheduled in a non preemptive fashion. This means that, if the system includes multiple neural networks with different complexity and periodicity requirements, those with shorter periods will be more likely to experience longer delays and higher response time variability. An example of non-preemptive schedule of three DNNs with different execution times and periods is illustrated in Figure 1.

Notice that, in this example, the total GPU utilization is less than one (U = 0.95), but DDN1 is forced to skip the second and the fifth execution instance, because it cannot preempt the execution of DNN3.

To solve this problem, Capodieci et al. [9], in collaboration with NVIDIA, proposed to modify the GPU internal scheduler with a preemptive scheduler based on Earliest Deadline First (EDF) [18], also providing bandwidth isolation by means of a Constant Bandwidth Server (CBS) [3]. Unfortunately, however, this solution is not yet available on commercial NVIDIA GPU platforms.

Other problems with GPU acceleration are due to the high power consumption and their significant weight and encumbrance, which prevent their usage in small embedded systems, as unmanned aerial vehicles (UAVs).

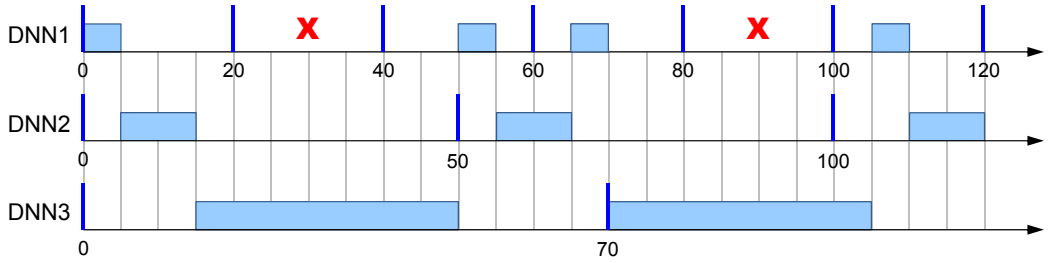

Figure 1 Example of non-preemptive schedule of three DNNs with different execution times and periods. As clear from the figure, DNN1 experiences longer and variable delays.

2.3 FPGA-related issues

An interesting alternative to GPUs for accelerating AI algorithms is provided by FPGAs. They are integrated circuits designed to be configured after manufacturing for implementing arbitrary logic functions in hardware. As such, they exhibit a highly predictable behavior in terms of execution times. In addition, they consume much less power with respect to GPUs and existing commercial platforms are characterized by lower weight, encumbrance, and cost. Hence, they represent an ideal solution for being used on battery-operated embedded systems with size, weight, power and cost (SWaP-C) constraints, as space robots, satellites, and UAVs.

Nevertheless, FPGAs have other problems when used as DNNs accelerators:

- No floating point unit (FPU) is available on the chip, unless it is explicitly programmed by the user, but consuming a significant fraction of the available fabric.
- Programming FPGAs is quite more difficult than programming CPUs or GPUs, and efficient coding requires a deep knowledge of low-level architecture details.
- The frameworks available today for developing AI applications on FPGA-based platforms are less rich and flexible than those available for GPUs, and the same is true for related libraries and tools.
- The overall FPGA area available in medium size SoCs could be insufficient to host more than one DNN, or even a single large DNN.

To overcome the problems outlined above, a lot of research has been carried out in the recent years.

The absence of an FPU is overcome by performing a preliminary parameter quantization to convert floating point numbers into integers with n-bit precision. Several quantization methods have been proposed in the literature [17], including symmetrical, asymmetrical, non-uniform, and statistical. An extreme quantization converts weights into binary numbers using the sign function. Courbariaux, Bengio, and David [14] have shown that a binarized DNN can achieve 98.8% accuracy in classifying the handwritten digits of the MNIST dataset. Other optimization steps (e.g., network pruning and layer fusion) can also be performed, both on GPUs and FPGAs, to reduce the computation time and the memory footprint of trained DNNs while minimizing the loss in accuracy.

To overcome the limitation of the FPGA area, Biondi et al. [6] proposed a programming framework, called FRED[1], to support the design, development, and execution of predictable software on FPGAs. FRED exploits dynamic partial reconfiguration and recurrent execution

[1] See details on `http://fred.santannapisa.it`.

to virtualize the FPGA area, thus enabling the user to allocate a larger number of hardware accelerators than those that could otherwise be fit into the physical fabric. FRED also integrates a tool for automated floorplanning [33] and a set of runtime mechanisms to enhance predictability by scheduling hardware resources and regulating bus/memory contentions [29].

An application targeted by FRED consists of a set of software tasks (SW-tasks) running on the CPU cores that can periodically invoke the execution of hardware accelerators (HW-tasks) to be dynamically programmed on the FPGA. The communication scheme adopted between SW-tasks and HW-tasks is illustrated in Figure 2.

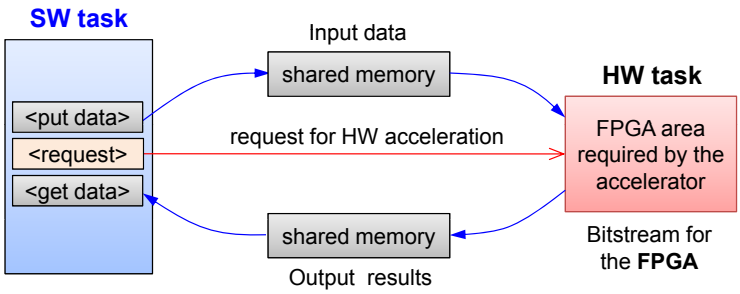

Figure 2 Communication scheme adopted between SW-tasks and HW-tasks in FRED.

The FPGA virtualization is achieved through a timesharing mechanism that replaces inactive accelerators (i.e., those that finished their computation and are waiting for the next activation) with active ones. In this way, the total number of HW-tasks that can run on the FPGA can be much higher than the number of HW-tasks that would statically fit in the physical area available on the fabric. Hence, this mechanism virtualizes the FPGA by creating a virtual area much larger than the physical one. The resulting approach is similar to multitasking, where tasks continuously change their context, or a virtual memory mechanism, where memory pages are swapped between hard disk and dynamic memory.

Thanks to a set of design choices and a proper scheduling infrastructure, resource contention delays experienced by tasks running under FRED are bounded and predictable, and hence they can be estimated to verify the system schedulability.

A full support for FRED has been developed under both FreeRTOS and Linux [25, 24]. The Linux support comes with a user-space daemon and a set of custom kernel drivers to handle the processor configuration port (PCAP) and the shared-memory communication buffers between CPUs and FPGA. A preemptable reconfiguration interface has also been developed by Rossi et al. [31] to achieve a finer control in scheduling the reconfiguration requests and a better control on the reconfiguration delays incurred by HW-tasks.

2.4 Framework issues

DNNs are commonly developed and inferred by means of state-of-the-art frameworks, as Tensorflow, Caffe, and PyTorch. Unfortunately, such frameworks are not optimized for being used in real-time applications and they are not supported by commercial real-time operating systems, as VxWorks and QNX. As a consequence, DNN tasks may be subject to a variable interference.

Casini et al. [10] addressed this problem by modifying the internal scheduler of the TensorFlow framework and adapting it for the SCHED_DEADLINE scheduling class of Linux. Extensive experiments demonstrated the effectiveness of the approach, showing a significant reduction of both average and longest-observed response times of TensorFlow tasks.

Recently, Restuccia and Biondi [27] proposed a set of techniques for accelerating DNNs on FPGA-based platforms with a highly predictable timing behavior under the Vitis AI frameworks by Xilinx. In Vitis AI, the execution of the DNN layers relies on the deep learning processing unit (DPU) core, a hardware accelerator optimized for the execution of convolutional DNNs. Based on an extensive profiling campaign conducted on the Xilinx Zynq Ultrascale+ platform, they proposed an execution model for the DPU employed to derive a response time analysis for guaranteeing real-time applications constraints.

3 Security issues

Safety-critical systems that make use of AI algorithms consist of several components with different complexity and requirements.

Consider for example a self-driving car. The functions responsible for steering, throttle modulation, braking, and engine control are highly critical and must satisfy stringent requirements in terms of safety, security, and real-time behavior. As such, they need to be managed by a real-time operating system, that must be certified to guarantee the required safety integrity levels.

On the other hand, high-level functions related to sensory perception, object tracking, and vehicle localization, which heavily rely on AI algorithms, need to be executed on a rich operating system (e.g., Linux) to exploit all the available device drivers, libraries, and development frameworks required for such complex computations. These components are far from being certified and offer a large software surface for cyber attacks.

In 2015, two hackers, Charlie Miller and Chris Valasek, discovered a vulnerability in the Jeep Cherokee, which they exploited to remotely access the vehicle and gain physical control, including steering, braking, turning on the wipers, blasting the radio, and finally, killing the engine to bring the vehicle to a complete stop [15]. They wrote a long paper [19] where they explain how they accessed the CAN bus through the infotainment system, detailing the full attack chain.

Although cyber attacks to non critical components accessible by a wireless network cannot be avoided completely, the security of a system can greatly be enhanced by preventing such attacks from spreading to more critical components. This can be achieved by *isolating* software components with different level of criticality into different execution domains, through the use of a hypervisor.

3.1 Hypervisor-based architecture

A hypervisor is a software layer above the hardware platform able to create and manage multiple execution domains, each hosting a virtual machine with its own operating system.

An example of a hypervisor-based architecture for safety-critical embedded systems is the one proposed by the SPHERE project [7], which supports the creation of multiple virtual machines on the same computing platform, providing both time/memory isolation, security, real-time communication channels, and I/O virtualization to allow different virtual machines to shared the peripheral devices.

Figure 3 shows an example of a hypervisor managing two execution domains with different levels of criticality. One domain hosts a virtual machine running all safety-critical functions on a real-time operating system (RTOS), while the other domain hosts another virtual machine running AI-powered software on the Linux operating system. Such an architecture has been

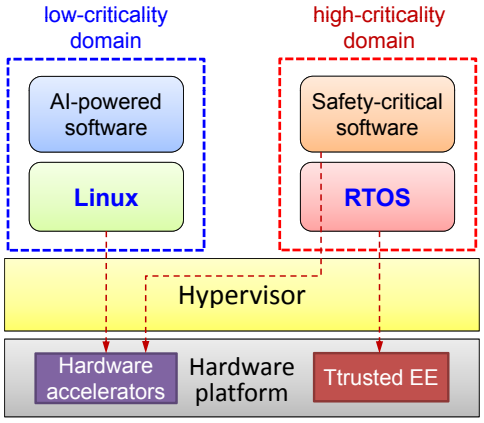

Figure 3 Example of a hypervisor managing two execution domains with different criticality.

successfully implemented and tested on a number of AI-powered control applications using CLARE [2], a novel hypervisor purposely designed to support mixed-criticality real-time systems exploiting AI hardware acceleration in heterogeneous embedded platforms.

3.2 CLARE hypervisor

The CLARE hypervisor is a novel bare-metal (type-1) hypervisor at the core of the CLARE software stack [2]. It integrates cutting-edge mechanisms to host safe, secure, and time-predictable virtual machines that can execute in isolation upon the same hardware platform. CLARE hypervisor follows a fully-static approach with off-line configurations and optimization to allocate the onboard resources to virtual machines. It has been designed to support modern heterogeneous platforms, such as GPGPU- and FPGA-based SoC, to better exploit and control their computational resources. In particular, it provides a number of real-time and security features that make it suitable for safety-critical systems, such as

- Improved key management and attack detection under control flow integrity by pointer authentication code [16].
- Hardware-based isolation exploiting the ARM TrustZone technology to perform key management and provide attack detection and recovery strategies.
- Virtualization of trusted execution environments leveraging ARM TrustZone [13].
- Protection mechanisms at the hypervisor level to control temporal and spatial interference among domains, also preventing side-channel attacks [20].
- I/O device virtualization and I/O related memory contention control, with related latency analysis [11].
- FPGA virtualization to allow multiple domains to exploit hardware accelerators in isolation.

4 AI related issues

Deep neural networks have shown an impressive performance in several recognition tasks, but their suitability for mission-critical applications has been questioned by Szegedy et al. [35] and many other authors [34], who showed that imperceptible perturbations added to an input sample can fool a neural network in perceiving objects that are not present in the input. Such perturbed inputs are called adversarial examples (AEs) and represent a serious

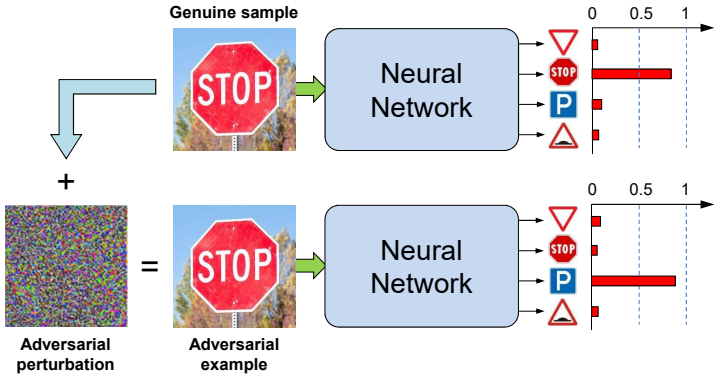

Figure 4 Example in which a neural network is able to correctly classify a genuine image of a stop sign (top). However, the same image can be modified by adding an adversarial perturbation, so that it is classified as a parking sign with a high confidence score (bottom).

threat for the security of AI-based systems. An example of adversarial image is shown in Figure 4, where the picture of a stop sign is perturbed in such a way that it is perceived by the network as a parking sign with a high confidence score.

Although a significant effort has been spent to develop defense methods against adversarial examples [5], the problem remains open and challenging, since these attacks violate the fundamental stationarity assumption of learning algorithms, i.e., that training and testing data are drawn from the same distribution.

The trustworthiness of DNNs is also threatened by genuine inputs characterized by a distribution that is quite different from that of the training samples. Such inputs are referred to as *out-of-distribution* (OoD) samples. Two examples of OoD images are shown in Figure 5.

Figure 5 Two examples of OoD images that could cause a deep neural network to produce a wrong output.

Considering that the prediction score of a DNN can be high in the presence on both AEs and OoD samples, the output score of the best classified class cannot be considered as an indication of the prediction confidence of the model.

Several methods have been proposed in the literature to detect AEs and OoD samples. Two of these methods are presented below with more details.

4.1 Detection by input transformations

One method for detecting AEs relies on the fact that DNN models are usually robust to certain types of input transformations (e.g., translation, rotation, scaling, blurring, noise addition, etc.). This means that, if a genuine image is correctly recognized by a DNN, the

prediction score reduces only slightly when the same image is translated, rotated, or modified with one the mentioned transformations. However, the same is not true for most AEs and it has been observed that they result to be more sensitive to input transformations, which cause a much higher degradation in the prediction score.

This property of AEs has been exploited by some authors [37] to detect whether an input x is adversarial or genuine. If $y = f(x)$ is the top class score produced by the DNN on input x and $y_T = f(T(x))$ is the score produced on the transformed input $T(x)$, a simple detection method is to consider x to be adversarial if the difference $y - y_T$ is higher than a given threshold τ.

Unfortunately, it is possible to generate adversarial examples that are robust to input transformations. To cope with this case, Nesti et al. [22] proposed a new method, called *defense perturbation*, capable of detecting AEs that are robust to input transformations. The defense perturbation is generated by a proper optimization process capable of making robust AEs sensitive again to input transformations. Furthermore, the paper introduces multi-network AEs that can fool multiple DNN models simultaneously, presenting a solution for detecting them.

4.2 Detection by coverage analysis

A different approach for detecting AEs is based on a deeper analysis of the neuron activation values in the different DNN layers. In fact, in order to force a DNN model to classify an input with a desired wrong class, AEs usually cause an overactivation of some neurons in different network layers. To identify such neurons, Rossolini et al. [32] presented a new coverage analysis methodology capable of detecting both adversarial and out-of-distribution inputs.

The approach works in two distinct phases: in a preliminary (off-line) phase, a trusted dataset is presented to the DNN and the neuron outputs, in each layer and for each class, are analyzed and aggregated into a set of covered states, which all together represent a sort of *signature* describing how the model responds to the trusted samples for each given class. Then, at runtime, each new input is subject to an evaluation phase, in which the activation state produced by the input in each layer is compared with the corresponding signature for the class predicted by the network. The higher the number of activation values outside the range observed during the presentation of the trusted dataset, the higher the probability that the current input is not trustworthy. The approach is schematically illustrated in Figure 6.

The nice thing about this approach is that the comparison against the signature allows computing a confidence value c distinct from the prediction score, indicating how much the current prediction can be trusted.

4.3 Interpretability issues

Another problem of complex machine learning models is that they are hardly interpretable by humans. In fact, they encode their input-output function in millions of parameters and, therefore, it is not trivial to understand why a given input produces a certain output. Many people demand for transparency and precise mechanisms as a prerequisite for trust. Especially for safety-critical applications, developers cannot trust a critical decision system if it is not possible to explain the reasons that brought to that decision.

To address this issue, a new branch of research, referred to as *Explainable AI* (XAI) [36], started around 2014 with the goal of reconstructing and representing in a comprehensible fashion the features that caused an AI model to produce a given output.

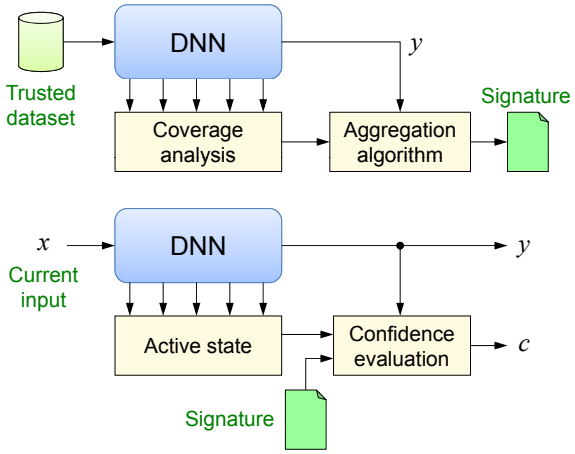

Figure 6 Overview of the coverage-based method proposed in [32]: (top) off-line phase for producing the signature for each layer and each class; (bottom) online detection phase based on comparing the current activation states with the stored signature. It produces a confidence value c indicating how much the current prediction can be trusted.

Thanks to these methods, it has been found that, in some cases, a DNN learned to classify images using features that have nothing to do with the object of interest, but appear frequently in most training samples of a specific class (e.g., water for the class *ship*, snow for the class *wolf*, or even copyright tags present in most of the images of a given class). Such biases in the training data limit the generalization capabilities of a neural model and can cause wrong predictions that would be quite harmful in applications like self-driving cars or medical diagnoses.

Identifying such biases in the training set is the main objective of XAI, which in the last years proposed several methods and tools for making AI decisions more interpretable to humans [21]. For instance, Pacini et al. [23] presented X-BaD, a flexible tool for detecting biases in training sets while providing user-interpretable explanations for DNN outputs. The tool can also be used to compare the performance of different XAI methods.

5 Towards trustworthy AI-based systems

From the considerations presented in the previous sections, it should be clear that, although AI algorithms exhibit a great performance in several perception and control tasks, they have intrinsic weaknesses in terms of safety, security, timing predictability, and certifiability. Does it mean that complex cyber-physical systems cannot take advantage of such an amazing software technology? Fortunately, there is a promising way to exploit the power of modern deep learning algorithms in safety-critical systems.

While we cannot prevent AI algorithms from being attacked or producing unsafe results, we can take a number of countermeasures to prevent them from harming the whole system.

For instance, the level of temporal predictability, as well as the level of security of the whole system can be increased by using a suitable real-time hypervisor capable of isolating the safety-critical components from the AI-powered functions in two separated virtual machines, as described in Section 3. In this way, an attack to the AI domain cannot propagate to the high-criticality domain, which can be protected by exploiting the hardware security features available in modern computer architectures.

To cope with adversarial attacks and OoD samples, the defense methods described in Section 4 are essential to detect both malicious as well as unsafe inputs that would cause a DNN to produce a wrong output. In these cases, the system must react by excluding the attacked AI component from the decision pipeline and switching to a simpler, but safer, backup control module that can bring the system into a safe state. For instance, in a self-driving car, the backup module could take control of the vehicle to stop it at the side of the road.

The approach described above has been undertaken by Biondi et al. [8] for exploiting deep learning models in safety-critical systems. The architecture includes two execution domains illustrated in Figure 7: a high-performance domain, running under Linux, and a safety-critical domain, running under the Erika Enterprise real-time kernel [1], both managed by the CLARE hypervisor [2].

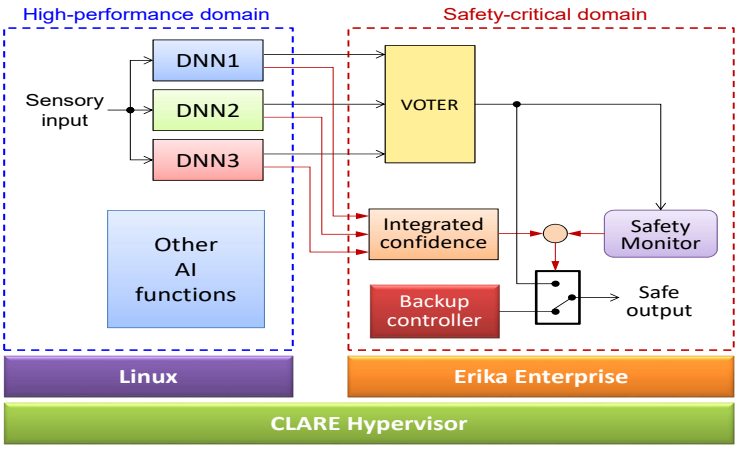

Figure 7 Architecture scheme proposed in [8], including a high-performance domain (running all the AI functions) and a safety-critical domain (running all critical control functions), managed by the CLARE hypervisor.

The high-performance domain is in charge of executing all the AI algorithms and tasks that must run under Linux, whereas the safety-critical domain runs all the vital system functions under the Erika Enterprise real-time kernel [1].

To increase the level of robustness against AEs and OoD inputs, each perceptual function is replicated by three different DNN models trained on different datasets. Each DNN also includes a coverage analyzer to provide a confidence signal (represented by the red arrow coming out from each DNN box). The three confidence signals are then integrated into an overall confidence signal, which is used to decide whether to switch to the safe controller. The three redundant DNN outputs go instead to a majority voter to resolve possible disagreements in the predictions.

An extra safety feature is represented by the presence of a *Safety Monitor*, which is in charge of detecting possible DNN outputs that could have a negative consequence on the controlled system. In fact, even in the presence of a non-malicious input, a DNN could generate an output that is not detected as unsafe by the voter and the confidence integrator, but could cause the controlled system to fail or misbehave. If such a condition is detected by the Safety Monitor, the system control is switched from the high-performance AI controller to the backup controller.

A first version of the architecture has been successfully implemented and tested on a control system for an inverted pendulum. In this case, the Safety Monitor uses a Lyapunov approach to detect when the system state exits a safe region of the state space and enters a margin region in which the stability can still be recovered by the safe controller. When this happens, the control is given to the backup controller. The AI controller is re-enabled when the system state goes back to the safe zone.

Figure 8 illustrates a simplified and qualitative representation of the state space for an inverted pendulum, where the black dot represents the current system state, the safe region is visualized in green, the recoverable region in yellow, and the unstable region in gray.

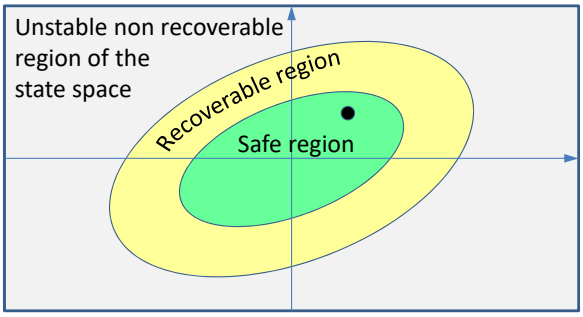

Figure 8 Qualitative representation of the state space for an inverted pendulum. The black dot represents the current system state.

On the same line, Belluardo et al. [4] presented a safe and secure multi-domain software architecture tailored for autonomous driving.

6 Conclusions

This paper presented a set of problems that today prevent the use of deep learning algorithms in mission-critical systems, as self-driving vehicles, autonomous robots, and medical applications. Among them, the most relevant issues are the low predictability of modern heterogeneous architectures and the high vulnerability of AI software to cyber attacks.

Fortunately, the research community is readily reacting to address such problems and some solutions to overcome such limitations have already been proposed at different architecture levels. However, the problems are many and complex, so several issues remain unsolved and require some joint effort from the AI and the real-time research communities.

References

1 Erika Enterprise RTOS. URL: `https://www.erika-enterprise.com/`.
2 The CLARE Software Stack. URL: `https://accelerat.eu/clare`.
3 Luca Abeni and Giorgio Buttazzo. Resource Reservation in Dynamic Real-Time Systems. *Real-Time Systems*, 27(2):123–167, July 2004.
4 L. Belluardo, A. Stevanato, D. Casini, G. Cicero, A. Biondi, and G. Buttazzo. A Multi-domain Software Architecture for Safe and Secure Autonomous Driving. In *Proc. of the 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2021)*, Online event, August 18–20, 2021.
5 Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, December 2018.

6 Alessandro Biondi, Alessio Balsini, Marco Pagani, Enrico Rossi, Mauro Marinoni, and Giorgio Buttazzo. A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS 2016)*, Porto, Portugal, November 29 – December 2, 2016.

7 Alessandro Biondi, Daniel Casini, Giorgiomaria Cicero, Niccolò Borgioli, and Giorgio Buttazzo et al. SPHERE: A Multi-SoC Architecture for Next-generation Cyber-Physical Systems Based on Heterogeneous Platforms. *IEEE Access*, 9:75446–75459, May 2021.

8 Alessandro Biondi, Federico Nesti, Giorgiomaria Cicero, Daniel Casini, and Giorgio Buttazzo. A Safe, Secure, and Predictable Software Architecture for Deep Learning in Safety-Critical Systems. *IEEE Embedded Systems Letters*, 12(3):78–82, September 2020.

9 N. Capodieci, R. Cavicchioli, M. Bertogna, and A. Paramakuru. Deadline-Based Scheduling for GPU with Preemption Support. In *Proc. of the 39th IEEE Real-Time Systems Symposium (RTSS 2018)*, Nashville, Tennessee, USA, December 11–14, 2018. URL: `http://arxiv.org/abs/1312.6199`.

10 Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo. Timing Isolation and Improved Scheduling of Deep Neural Networks for Real-Time Systems. *Software: Practice and Experience*, 50(9):1760–1777, September 2020.

11 Daniel Casini, Alessandro Biondi, Giorgiomaria Cicero, and Giorgio Buttazzo. Latency Analysis of I/O Virtualization Techniques in Hypervisor-Based Real-Time Systems. In *Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2021)*, Online event, May 18–21, 2021.

12 R. Cavicchioli, N. Capodieci, and M. Bertogna. Memory Interference Characterization Between CPU Cores and Integrated GPUs in Mixed-Criticality Platforms. In *Proc. of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2017)*, Limassol, Cyprus, September 12–15, 2017.

13 Giorgiomaria Cicero, Alessandro Biondi, Giorgio Buttazzo, and Anup Patel. Reconciling Security with Virtualization: A Dual-Hypervisor Design for ARM TrustZone. In *Proceedings of the 18th IEEE International Conference on Industrial Technology (ICIT 2018)*, Lyon, France, February 20–22, 2018.

14 Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Proc. of the 29th Conference on Neural Information Processing Systems (NIPS 2015)*, Montreal, Canada, December 7–10, 2015.

15 Blane Erwin. The Groundbreaking 2015 Jeep Hack Changed Automotive Cybersecurity, 2021. URL: `https://fractionalciso.com/the-groundbreaking-2015-jeep-hack-changed-automotive-cybersecurity/`.

16 Giulia Ferri, Giorgiomaria Cicero, Alessandro Biondi, and Giorgio Buttazzo. Towards the Hypervision of Hardware-based Control-Flow Integrity for Arm Platforms. In *Proceedings of the Italian Conference on CyberSecurity (ITASEC 2019)*, Pisa, Italy, February 12–15, 2019.

17 Yunhui Guo. A Survey on Methods and Theories of Quantized Neural Networks. *ArXiv*, abs/1808.04752, 2018.

18 C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):40–61, January 1973.

19 Charlie Miller and Chris Valasek. Remote Exploitation of an Unaltered Passenger Vehicle, August 10, 2015. URL: `http://illmatics.com/Remote%20Car%20Hacking.pdf`.

20 Paolo Modica, Alessandro Biondi, Giorgio Buttazzo, and Anup Patel. Supporting Temporal and Spatial Isolation in a Hypervisor for ARM Multicore Platforms. In *Proceedings of the 18th IEEE International Conference on Industrial Technology (ICIT 2018)*, Lyon, France, February 20–22, 2018.

21 Christoph Molnar. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, 2021. URL: `https://christophm.github.io/interpretable-ml-book/`.

22 Federico Nesti, Alessandro Biondi, and Giorgio Buttazzo. Detecting Adversarial Examples by Input Transformations, Defense Perturbations, and Voting. *IEEE Transactions on Neural Networks and Learning Systems*, August 2021.

23 Marco Pacini, Federico Nesti, Alessandro Biondi, and Giorgio Buttazzo. X-BaD: A Flexible Tool for Explanation-Based Bias Detection. In *Proc. of the IEEE International Conference on Cyber Security and Resilience*, Online event, July 26–28, 2021.

24 M. Pagani, A. Biondi, M. Marinoni, L. Molinari, G. Lipari, and G. Buttazzo. A Linux-Based Support for Developing Real-Time Applications on Heterogeneous Platforms with Dynamic FPGA Reconfiguration. *Future Generation Computer Systems*, To appear.

25 Marco Pagani, Alessio Balsini, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. A Linux-based Support for Developing Real-Time Applications on Heterogeneous Platforms with Dynamic FPGA Reconfiguration. In *Proceedings of the 30th IEEE International System-on-Chip Conference (SOCC 2017)*, Munich, Germany, September 5–8, 2017.

26 Marco Pagani, Enrico Rossi, Alessandro Biondi, Mauro Marinoni, Giuseppe Lipari, and Giorgio Buttazzo. A Bandwidth Reservation Mechanism for AXI-based Hardware Accelerators on FPGAs. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS 2019)*, Stuttgart, Germany, July 9–12, 2019.

27 Francesco Restuccia and Alessandro Biondi. Time-Predictable Acceleration of Deep Neural Networks on FPGA SoC Platforms. In *Proc. of the 42nd IEEE Real-Time Systems Symposium (RTSS 2021)*, Online event, December 7–10, 2021.

28 Francesco Restuccia, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Safely preventing unbounded delays during bus transactions in FPGA-based SoC. In *Proceedings of the 28th Annual Int. Symposium on Field-Programmable Custom Computing Machines (FCCM 2020)*, Fayetteville, Arkansas, USA, May 3–6, 2020.

29 Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Is Your Bus Arbiter Really Fair? Restoring Fairness in AXI Interconnects for FPGA SoCs. *ACM Transactions on Embedded Computing Systems*, 18(5-51):1–22, October 2019.

30 Francesco Restuccia, Marco Pagani, Alessandro Biondi, Mauro Marinoni, and Giorgio Buttazzo. Modeling and analysis of bus contention for hardware accelerators in FPGA SoCs. In *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, Online event, July 7–10, 2020.

31 Enrico Rossi, Marvin Damschen, Lars Bauer, Giorgio Buttazzo, and Jörg Henkel. Preemption of the Partial Reconfiguration Process to Enable Real-Time Computing with FPGAs. *ACM Transactions on Reconfigurable Technology and Systems*, 11(2):10:1–10:24, November 2018.

32 Giulio Rossolini, Alessandro Biondi, and Giorgio Buttazzo. Increasing the Confidence of Deep Neural Networks by Coverage Analysis. In *arXiv:2101.12100 [cs.LG]*, January 2021. `arXiv:2101.12100`.

33 Biruk Seyoum, Alessandro Biondi, and Giorgio Buttazzo. FLORA: FLoorplan Optimizer for Reconfigurable Areas in FPGAs. *ACM Transactions on Embedded Computing Systems*, 18(5-73):1–20, October 2019.

34 Tom Simonite. AI Has a Hallucination Problem That's Proving Tough to Fix, March 12, 2018. URL: `https://www.wired.com/story/ai-has-a-hallucination-problem-thats-proving-tough-to-fix/`.

35 Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. of the 2nd International Conference on Learning Representations (ICLR 2014)*, Banff, AB, Canada, April 14–16, 2014. URL: `http://arxiv.org/abs/1312.6199`.

36 The Royal Society. Explainable AI: the basics – policy briefing, 2019. URL: `https://royalsociety.org/-/media/policy/projects/explainable-ai/AI-and-interpretability-policy-briefing.pdf`.

37 Shixin Tian, Guolei Yang, and Ying Cai. Detecting Adversarial Examples through Image Transformation. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, New Orleans, Louisiana, USA, February 2—7, 2018.

Multi-Requirement Enforcement of Non-Functional Properties on MPSoCs Using Enforcement FSMs – A Case Study

Khalil Esper
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Stefan Wildermann
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Abstract

Embedded system applications usually have to meet real-time, energy or safety requirements on programs typically concurrently executed on a given MPSoC target platform. Enforcing such properties, e.g., by adapting the number of processors allocated to a program or by scaling the voltage/frequency mode of involved processors, is a difficult problem to solve, especially with a typically large varying environmental input (workload) per execution. In a previous work [4], we formalized the related enforcement problem using (a) finite state machines to model enforcement strategies, (b) discrete-time Markov chains to model the uncertain environment determining the system's workload, and (c) the system response that defines the feedback for the reactive enforcer. In this paper, we apply that approach to specify and verify multi-requirement enforcement strategies and assess a case study for enforcing two independent requirements at the same time, i.e., latency and energy consumption. We evaluate and compare different enforcement strategies using probabilistic verification for the use case of an object detection application.

2012 ACM Subject Classification Computer systems organization → Multicore architectures; Theory of computation → Linear logic; Theory of computation → Modal and temporal logics; Hardware → Finite state machines; Computer systems organization → Self-organizing autonomic computing; Theory of computation → Verification by model checking; Mathematics of computing → Probabilistic representations

Keywords and phrases Runtime Requirement Enforcement, Verification, Finite State Machine, Markov Chain, Energy Consumption, Probabilistic Model Cheking, PCTL, MPSoC

Digital Object Identifier 10.4230/OASIcs.NG-RES.2022.2

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

1 Introduction

Many, particularly embedded system applications come along with different requirements that should be met during their execution on many-core systems. One example are temperature constraints because of a limited power budget. Causes of variations include interference with other applications, i.e., due to shared resources. In addition, the workload induced by the environment input data[1] can vary significantly and with high uncertainty.

[1] This paper takes care of the uncertainty of input from the environment that is typically not under the control of a system, thus the influence is exogenous. The uncertainty of execution state on an MPSoC platform is typically caused by sharing of resources on an MPSoC platform, thus endogenous. This problem can be treated systematically by techniques for isolating application programs dynamically at run-time such as invasive computing [18] and is therefore not treated here.

As a solution, *run-time requirement enforcement (RRE)* techniques such as [19] have emerged. Such techniques adjust a set of configurations like voltage/frequency setting or the degree of parallelism in reaction to observed changes in the system state or input workload. Different techniques have been proposed to implement run-time managers for dynamic adaptation of program execution [5, 7, 11, 12]. Many of them have drawbacks that either no formal guarantees can be given regarding their effectiveness of holding the specified requirements or they make simplifying assumptions regarding the controller or the many-core system under control. To overcome such disadvantages, we use *finite state machines (FSMs)* to formally specify enforcement strategies, as they are expressive in specification and computation, which helps in formulating complex strategies. Furthermore, they do not make any restricting or simplifying assumptions regarding the system-under-control, and finally they can be analyzed using formal verification techniques [4].

In this respect, we proposed a methodology [4] to verify the satisfaction of requirements on non-functional properties of program executions on MPSoCs which are controlled by FSM-based RREs. A requirement is often specified by a corridor of allowed or desired values of a non-functional property of program execution, e.g., a latency or power consumption corridor. In order to quantitatively assess a RRE strategy, we specify and formally verify important verification goals, e.g., whether a RRE is able to either strictly or at least loosely stay within the bounds of one or a set of requirements. Furthermore, how many subsequent program executions could violate a given set of requirements before eventually returning into specified requirement corridors. After introducing a *discrete-time Markov chain (DTMC)* for modeling the variation of environmental input, we model the *requirement response* of an MPSoC when executing an application program as a function that specifies for each combination of environment state and enforcer output whether and which requirements are fulfilled and which are violated.

Based on the concatenation of formal models, we can define stochastic verification problems using *Probabilistic Computation Tree Logic (PCTL)* [1], a probabilistic variant of CTL, and apply stochastic model checking, i.e., PRISM [9] to analyze and compare different enforcer strategies quantitatively.

Unlike other approaches that enforce only one requirement at a time [16, 2, 14, 15, 13], our method [4] can be used to enforce more than one requirement – although this was not shown there. In this work, we therefore use the methodology in [4] to analyze enforcers for multiple independent requirements at a time. As an example, we propose multi-requirement enforcement strategies to satisfy more than one requirement (e.g., latency and energy requirements) at a time. We then assess a case study to evaluate and compare between different strategies using probabilistic formal verification based on the PRISM model checker.

The remainder of this paper is structured as follows. Section 2 introduces the formal models for specifying the enforcement FSM and the requirement response. Section 3 describes the evaluation of a proposed set of uni- and multi-requirement enforcement FSMs based on an image processing case study and discusses the verification results. Finally, in Section 4, we conclude this work.

2 Run-time Requirement Enforcement (RRE)

The enforcement of requirements should be achieved even under a variation of environmental inputs. Such an input can be described for each discrete periodic execution k of a program by an environment feature vector $x(k) \in \mathcal{X}$, where $\mathcal{X}$ denotes the *environment space*. *Run-time requirement enforcement (RRE)* techniques such as [19, 3] can be used to enforce the

satisfaction of a set of requirements during each execution even in the presence of input variation. In the following, we term such an assignment of resources and their parametrization to execute a given program a configuration c and the set of possible system configurations to execute a program *configuration space* C. We consider *Feedback-based RRE* techniques such as [7] that react to a violation or satisfaction of a non-functional requirement φ based on feedback from the *system-under-control* and adapting the configuration $c(k+1)$ for the next execution $(k+1)$ accordingly [4]. Figure 1 illustrates the proposed model described in the following.

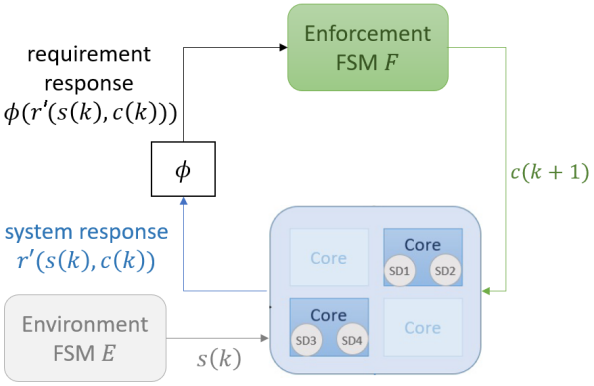

Figure 1 Schematic illustration of feedback-based RRE [4].

2.1 Enforcement FSM F

Feedback-based RREs can be modeled by finite state machines. Following [4], a respective enforcement FSM is defined as follows:

▶ **Definition 1.** *An* enforcement FSM (F) *is a deterministic finite state machine (Moore machine) that can be described by a 6-tuple* $(Z, z_0, I, \delta, C, \gamma)$ [4]*:*

- Z *is a finite set of states.*
- $z_0 \in Z$ *is the initial state.*
- I *is the input alphabet.*
- δ *is the transition relation:* $\delta \subseteq I \times Z \times Z$ *with* (i, z, z') *representing a transition from state* z *to state* z' *under input* i*.*
- C *is the output alphabet, also called configuration space.*
- γ *is the output function, which maps each state to the output alphabet:* $\gamma : Z \to C$*.*

An enforcement FSM is *uni-requirement* when it enforces one requirement $(h = 1)$, and it is called *multi-requirement* when it enforces more than one requirement $(h > 1)$.

2.2 Requirement Enforcement

A MPSoC platform, or *system-under-control* is abstracted by a single function called *system response function* $r : \mathcal{X} \times C \to \mathbb{R}^h$ [4]. The *system response* at execution k is considered as a vector of h relevant execution qualities $r(x(k), c(k)) = (o_1(k), \ldots, o_h(k))$, e.g., corresponding to observed latency, power consumption, etc., during the kth execution.

In [17], requirements on non-functional execution qualities o_j, $j = 1, \ldots, h$ are specified using intervals, e.g., resulting in two propositions φ_j^{LB} and φ_j^{UB}

$$\varphi_j(o_j(k)) = \varphi_j^{LB}(o_j(k)) \wedge \varphi_j^{UB}(o_j(k)) = (LB_{o_j} \leq o_j(k)) \wedge (o_j(k) \leq UB_{o_j}) \tag{1}$$

where LB_{o_j} and UB_{o_j} denote a lower, respectively upper bound on the execution quality o_j.

The system response $r(x(k), c(k)) = (o_1(k), \ldots, o_h(k))$ is finally mapped to a *requirement response* using a requirement response function

$$\begin{aligned}\phi\left(o_1(k), \ldots, o_h(k)\right) := \big(&\varphi^{LB}(o_1(k)), \varphi^{UB}(o_1(k)), \ldots, \\ &\varphi^{LB}(o_h(k)), \varphi^{UB}(o_h(k))\big) \in \{0,1\}^{2h}\end{aligned} \tag{2}$$

This binary vector serves as the input to the enforcement FSM F, and thus the input alphabet is $I \subseteq \{0,1\}^{2h}$. Based on the requirement response, F computes as a reaction the next configuration $c(k+1) \in C$ to enforce the desired non-functional properties for the next execution.

2.3 Environment FSM E

In order to apply verification techniques on a proper enforcement of requirements, we need a formal model of the environment that influences the system-under-control. Thus, the environment is modeled using a discrete-time Markov Chain called *environment FSM* [4]. However, the number of possible input values can be very large, possibly leading to a state explosion. In [4], we proposed to partition the environment space $\mathcal{X}$ into equivalence classes or partitions p leading to the same requirement response ϕ as a potential solution.

▶ **Definition 2.** *An* environment finite state machine E *is a discrete-time Markov chain (DTMC) defined by a 3-tuple* (S, a, Δ) [4]*:*

- S *is the finite set of states. Each* $s \in S$ *is assigned exactly one partition* $p \in P$ *of the environment space* $\mathcal{X}$*.*
- $a : S \to [0,1]$ *is a function that assigns each state* $s \in S$ *its initial state probability* $a(s)$*.*
- $\Delta \in [0,1]^{|S| \times |S|}$ *is a transition probability matrix.*

3 Multi-Requirement Enforcement Case Study

In the following, we present a simple image processing application as shown in Figure 2. The job of the object detection algorithm is to detect a given object in each image frame by applying a SIFT feature matching algorithm. Subsequently, we present different RRE variants and verify a number of PCTL verification goals related to h requirements to be enforced using probabilistic model checking.

3.1 Object Detection Application

The object detection application, shown in Figure 2, is an image processing application that conducts a pipelined processing of a periodic input image stream. The goal of this application is to detect a given object in each image frame by applying a scale-invariant feature transform (SIFT) matching algorithm [10]. The application consists of an actor chain. Each actor processes one input image frame k at a time. The image source (IS) actor reads in the input images periodically at a defined 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 the edges and corners in an image. After that, the SIFT orientation (SO) actor applies 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 x of a given image evenly into each actor.

For the following experiments, let each of the periodic executions of each SD actor have a latency requirement $\varphi_L = \varphi_L^{LB} \wedge \varphi_L^{UB} = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ which is typical in real-time systems and an energy consumption requirement $\varphi_{En} = \varphi_{En}^{LB} \wedge \varphi_{En}^{UB} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$, for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ. Intuitively, $\varphi_L^{LB} = (0 \text{ ms} \leq o_L)$ and $\varphi_{En}^{LB} = (0 \text{ mJ} \leq o_{En})$ are always satisfied.

For the enforcement of such requirements, 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 $\langle n, m \rangle$. Upon each execution, the output of each SD actor is then sent to the SIFT matching (SM) actor to detect common features between the given reference object and the features in 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 delivered by an image destination (ID) actor.

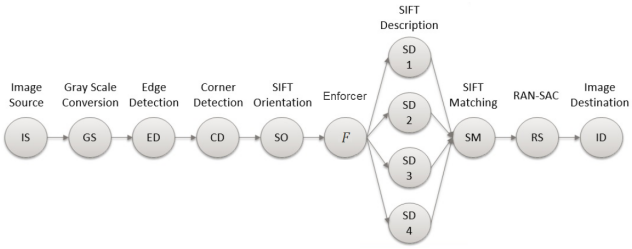

Figure 2 Object detection algorithm implemented as a graph of actors for pipelined processing of streams of images.

3.2 Specifying Enforcement FSMs

In the following, we introduce and compare five enforcement FSMs exemplarily, each having as many states as configurations $|C|$, thus $Z = \{0, \cdots, 31\}$, the input $i \in I = \{0,1\}^{2h} = \{0,1\}^4$ with $i = \phi(r'(s,c)) = \phi(o_L, o_{En}) = ((LB_{o_L} \leq o_L), (o_L \leq UB_{o_L}), (LB_{o_{En}} \leq o_{En}), (o_{En} \leq UB_{o_{En}}))$, an assumed initial state $z_0 = 17$ and the configuration space C of cardinality $|C| = 32$.

- *1-step latency-requirement enforcement FSM* F_1: [3] proposes a technique called 1-step enforcement that decreases, resp. increases the current state, respectively configuration by exactly one step in case of a satisfaction (φ_L), resp. violation ($\overline{\varphi_L}$) of a latency requirement to be enforced. A corresponding enforcement FSM $F_1 = (Z, z_0, I, \gamma, C, \delta_1)$ is shown in Figure 3. It has the following transition relation δ_1:

$$z(k+1) = \begin{cases} 0 & \varphi_L \wedge (z(k) = 0) \\ z(k) - 1 & \varphi_L \wedge (z(k) \neq 0) \\ z(k) + 1 & \overline{\varphi_L} \wedge (z(k) \neq 31) \\ 31 & \overline{\varphi_L} \wedge (z(k) = 31) \end{cases} \tag{3}$$

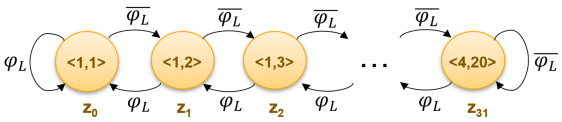

Figure 3 1-step latency-requirement enforcement FSM F_1 that only utilizes φ_L. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *1-step energy-requirement enforcement FSM* F_2: Decreases, resp. increases the current state, respectively configuration reflecting the next lower, resp. higher power by exactly one step in case of a satisfaction (φ_{En}), resp. violation ($\overline{\varphi_{En}}$) of a energy requirement to be enforced. A corresponding enforcement FSM $F_2 = (Z, z_0, I, \gamma, C, \delta_2)$ is shown in Figure 4. It has the following transition relation δ_2:

$$z(k+1) = \begin{cases} 0 & \overline{\varphi_{En}} \wedge (z(k) = 0) \\ z(k) - 1 & \overline{\varphi_{En}} \wedge (z(k) \neq 0) \\ z(k) + 1 & \varphi_{En} \wedge (z(k) \neq 31) \\ 31 & \varphi_{En} \wedge (z(k) = 31) \end{cases} \tag{4}$$

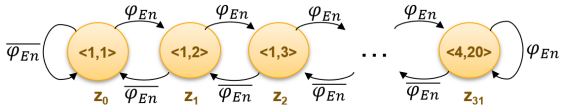

Figure 4 1-step energy-requirement enforcement FSM F_2 that only utilizes φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *1-step multi-requirement enforcement FSM* F_3: Stays in the current state, respectively configuration if both requirements are satisfied or none of the requirements are satisfied, and decreases the current state by exactly one step if only φ_{En} is violated and increases if only φ_L is violated. A corresponding enforcement FSM $F_3 = (Z, z_0, I, \gamma, C, \delta_3)$ is shown in Figure 5. It has the following transition relation δ_3:

$$z(k+1) = \begin{cases} 0 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) = 0) \\ z(k) - 1 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) \neq 0) \\ z(k) & ((\varphi_L \wedge \varphi_{En}) \vee (\overline{\varphi_L} \wedge \overline{\varphi_{En}})) \wedge (z(k) \neq 0) \\ z(k) + 1 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) \neq 31) \\ 31 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) = 31) \end{cases} \tag{5}$$

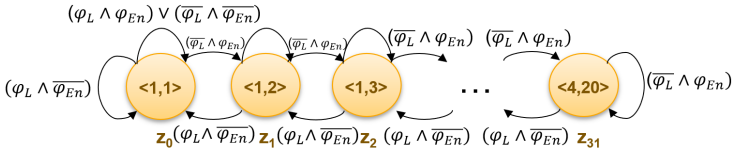

Figure 5 1-step multi-requirement enforcement FSM F_3 for enforcing two single-bound requirements φ_L and φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *Latency-oriented multi-requirement enforcement FSM* F_4: Decreases the current state by one step when only φ_{En} is violated, increases the current state by two steps when φ_L is violated, and stays in the same state otherwise. A corresponding enforcement FSM $F_4 = (Z, z_0, I, \gamma, C, \delta_4)$ is shown in Figure 6. It has the transition relation δ_5:

$$z(k+1) = \begin{cases} 0 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) = 0) \\ z(k) - 1 & (\varphi_L \wedge \overline{\varphi_{En}}) \wedge (z(k) \neq 0) \\ z(k) & (\varphi_L \wedge \varphi_{En}) \wedge (z(k) \neq 0) \\ z(k) + 2 & \overline{\varphi_L} \wedge (z(k) < 30) \\ 31 & \overline{\varphi_L} \wedge (z(k) \geq 30) \end{cases} \tag{6}$$

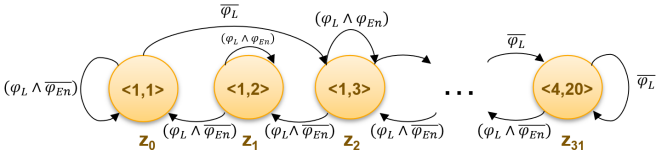

Figure 6 Latency-oriented multi-requirement enforcement FSM F_4 for enforcing two single-bound requirements φ_L and φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

- *Energy-oriented multi-requirement enforcement FSM* F_5: Decreases the current state by two steps when φ_{En} is violated, increases the current state by one step when only φ_L is violated, and stays in the same state otherwise. A corresponding enforcement FSM $F_5 = (Z, z_0, I, \gamma, C, \delta_5)$ is shown in Figure 7. It has the following transition relation δ_4:

$$z(k+1) = \begin{cases} 0 & \overline{\varphi_{En}} \wedge (z(k) \leq 1) \\ z(k) - 2 & \overline{\varphi_{En}} \wedge (z(k) > 1) \\ z(k) & (\varphi_L \wedge \varphi_{En}) \wedge (z(k) \neq 0) \\ z(k) + 1 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) \neq 31) \\ 31 & (\overline{\varphi_L} \wedge \varphi_{En}) \wedge (z(k) = 31) \end{cases} \tag{7}$$

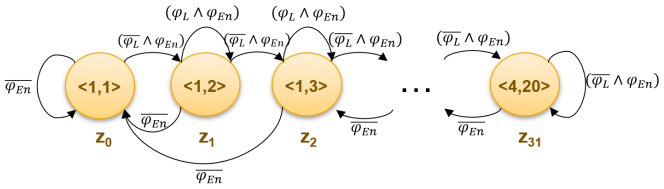

Figure 7 Energy-oriented multi-requirement enforcement FSM F_5 for enforcing two single-bound requirements φ_L and φ_{En}. Annotated to each state is the output configuration $c = (n, m)$, consisting of number n of powered cores and power mode m.

3.3 Verification Goals Specification

As we do not want to reason about or verify the satisfaction of verification goals during single runs or single traces of input stimuli, but depending on a DTMC model of the typically uncertain environment (i.e., E), we can use PCTL [1], a probabilistic variant of CTL, to specify stochastic verification goals. In the following, $\mathcal{P}_{=?}[\varphi_j]$ denotes the probability of satisfying a proposition φ_j.

For verification, we will use PRISM [9], a probabilistic model checker, to perform verification of a number of interesting verification goals on enforcers.

Finally, LTL formulas have a *bounded* variant in PCTL [6], which adds an upper bound λ on the number of successive steps or iterations in our model. In our previous work [4], we proposed and formulated a set of interesting verification goals using temporal logic. In this work, we use one of them to compare between the proposed enforcement FSMs. That being: $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi)]$ which denotes the probability of a requirement φ to never hold in any of λ consecutive executions. We will use this verification goal to verify the proposed enforcement strategies for the latency requirement $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L)]$, the energy consumption requirement $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_{En})]$, and finally both requirements together $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L \vee \neg\varphi_{En})]$.

3.4 Deriving Environment FSMs E

We first partition the environment space $\mathcal{X}$ into a set of partitions P, by computing the system response $o = (o_L, o_{En}) = r(x, c)$ for all different inputs $x \in \mathcal{X}$ for each $c \in C$, and then deriving the partitions p of inputs $x \in \mathcal{X}$ that result in the same requirement response $\phi(r(x, c))$ for each $c \in C$.

For the above example application, the authors in [20] proposed to perform a design space exploration (DSE) to compute the maximum number of features $x_{max}(UB_{o_L}, c)$ that can be processed within a given deadline UB_{o_L} in configuration c, and a single state enforcer that proactively (rather than reactively) chooses upon arrival of the kth frame always the configuration $c \in C$ consuming the minimal amount of energy for that input. A similar DSE is conducted for the energy consumption o_{En} for an energy consumption upper bound $UB_{o_{En}}$.

Based on this information, we can compute the requirement response $\phi(r(x, c))$ for each input $x \in \mathcal{X}$ and configuration $c \in C$ for each execution k as follows:

$$\phi(r(x,c)) = \begin{cases} (1,1,1,1) & (\varphi_L^{LB}, \varphi_L^{UB}, \varphi_{En}^{LB}, \varphi_{En}^{UB}) \\ (1,1,1,0) & (\varphi_L^{LB}, \varphi_L^{UB}, \varphi_{En}^{LB}, \overline{\varphi_{En}^{UB}}) \\ (1,0,1,1) & (\varphi_L^{LB}, \overline{\varphi_L^{UB}}, \varphi_{En}^{LB}, \varphi_{En}^{UB}) \\ (1,0,1,0) & (\varphi_L^{LB}, \overline{\varphi_L^{UB}}, \varphi_{En}^{LB}, \overline{\varphi_{En}^{UB}}) \end{cases} \tag{8}$$

Based on this partitioning, and using a frame sequence R with $|R| = 1{,}000$ frames, we follow the procedure explained in [4] to obtain the environment FSM E_1, shown in Figure 8, which is specified after computing the partitions based on a latency requirement $\varphi_L = \varphi_L^{LB} \wedge \varphi_L^{UB} = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ and an energy consumption requirement $\varphi_{En} = \varphi_{En}^{LB} \wedge \varphi_{En}^{UB} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$, for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ. Intuitively, $\varphi_L^{LB} = (0 \text{ ms} \leq o_L)$ and $\varphi_{En}^{LB} = (0 \text{ mJ} \leq o_{En})$ are always satisfied.

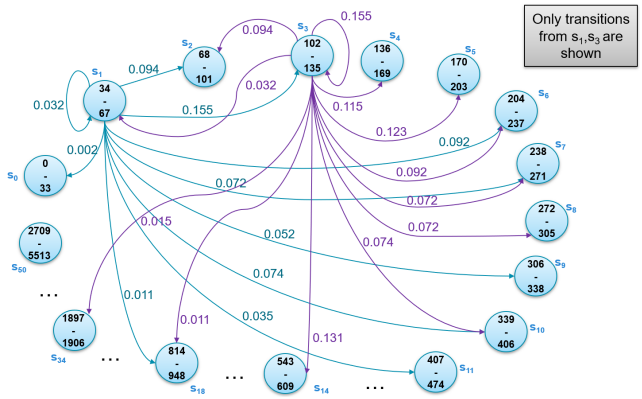

Figure 8 Resulting environment FSM E_1 and for an image processing algorithm for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ generated from a trace R of $|R| = 1000$ frames. The highest encountered number of features x (workload to be processed) per frame in the trace R is 5,513.

3.5 Verification Results

We specified the enforcement and environment FSMs for verification by the PRISM model checker using the PRISM modeling language [8].

- $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L)]$ is the probability of φ_L to never hold in any of λ consecutive executions. Figure 9 plots this probability for increasing values of λ for all five introduced enforcement FSMs. We notice that F_2 and F_5 have the highest probabilities of violating the latency requirement among all other enforcement FSMs. This is plausible, since F_2 only utilizes the energy requirement response as an input, and the multi-requirement enforcement FSM F_5 is energy-oriented.
  We also notice that F_4 is superior regarding satisfying the satisfaction of latency requirement, since it is a latency-oriented multi-requirement enforcement FSM that jumps two steps forward when φ_L is violated. Finally, even F_1 utilizes only the latency requirement response to transition between states, it has a higher probability of violating the latency requirement than F_3 and F_4, because it goes backwards one step when φ_L is satisfied, unlike the multi-enforcement FSMs F_3 and F_4 which stay in the same state when φ_L is satisfied.
- $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_{En})]$ is the probability of φ_{En} to never hold in any of λ consecutive executions. Figure 10 plots this probability for increasing values of λ for all five introduced enforcement FSMs. We notice that F_5 has the lowest probability for violating the energy requirement, because it is a multi-requirement FSM that is oriented for energy saving where it goes

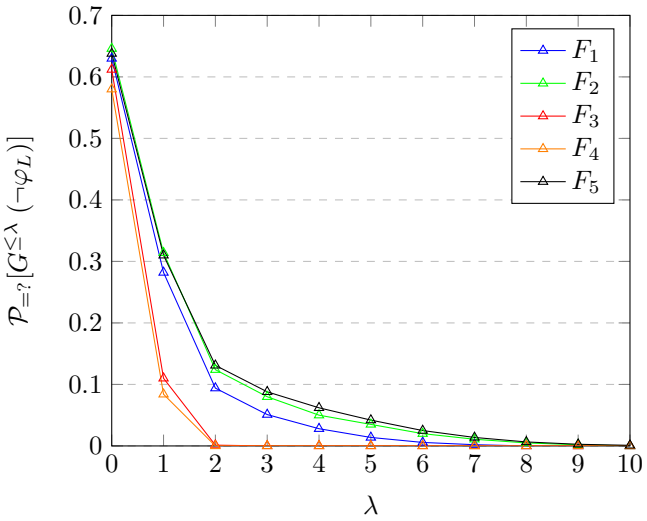

Figure 9 Probability that φ_L never holds in any of λ consecutive executions ($\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L)]$) where $\varphi_L = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms.

backward two steps when φ_{En} is violated. Next comes F_2, which only utilizes the energy requirement response as an input. We also notice that other enforcement FSMs violate the energy requirement with higher probabilities than F_2, F_5, since they either do not consider the energy requirement response like F_1 or they do not prioritize it over the latency requirement φ_L like F_3 and F_4. Finally, we notice that the latency-oriented multi-requirement FSM F_4 has the highest probability of violating the energy requirement. Even though F_4 considers the energy requirement response, F_1 still has a lower probability in violating φ_{En}. This shows the need for systematic methods to generate enforcement FSMs that satisfy given requirements.

- Finally, we analyze the probability of φ_L or φ_{En} to never hold in any of λ consecutive executions by computing $\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L \vee \neg\varphi_{En})]$. Figure 11 plots this probability for increasing values of λ for all introduced enforcement FSMs. As can be observed, F_3 exhibits the lowest probability of violating any requirement. This is because it has the third lowest probability of violating the energy requirement φ_{En} and the second lowest probability of violating the latency requirement φ_L, see Figure 9 and Figure 10. F_2 and F_5 show the next best behavior after F_4, because they have the lowest probability of violating the energy requirement φ_{En}, and at the same time showing a comparable probability concerning the latency requirement, see Figure 9 and Figure 10. Finally, although F_4 is the best regarding the satisfaction of the latency requirement, it is the worst in satisfying both requirements together, as it has a very low probability of satisfying the energy requirement φ_{En}, which outweighs the latency requirement satisfaction, see Figure 10.

4 Conclusion and Future Work

In this paper, we proposed multi-requirements enforcement strategies for enforcing multiple non-functional requirements at a time, like latency and energy consumption. We formulated several verification goals that ask for the probability of violating the latency, energy or both requirements for a consecutive number of steps. Doing so allows to formally verify and

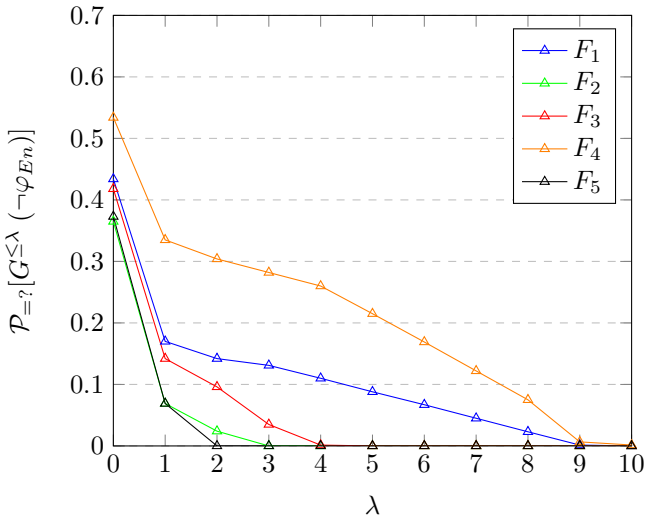

Figure 10 Probability that φ_{En} never holds in any of λ consecutive executions ($\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_{En})]$) where $\varphi_{En} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$ for an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ.

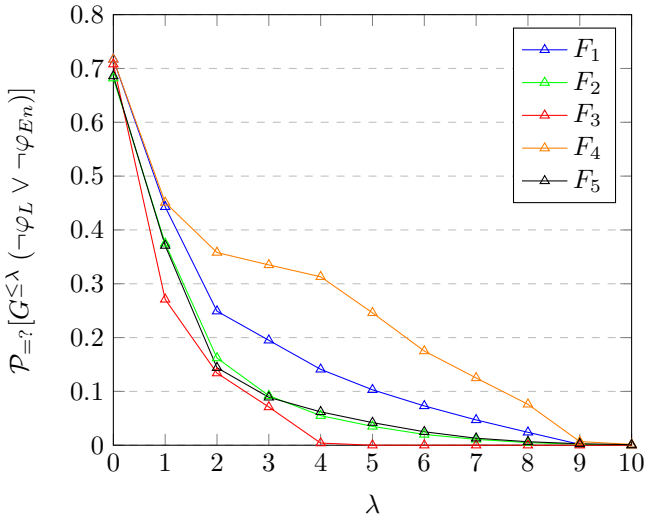

Figure 11 Probability that φ_L or φ_{En} never holds in any of λ consecutive executions ($\mathcal{P}_{=?}[G^{\leq\lambda}\ (\neg\varphi_L \vee \neg\varphi_{En})]$) where $\varphi_L = (LB_{o_L} \leq o_L) \wedge (o_L \leq UB_{o_L})$ and $\varphi_{En} = (LB_{o_{En}} \leq o_{En}) \wedge (o_{En} \leq UB_{o_{En}})$, for a latency lower bound $LB_{o_L} = 0$ ms and an upper bound (deadline) $UB_{o_L} = 80$ ms and an energy consumption lower bound $LB_{o_{En}} = 0$ mJ and an energy consumption upper bound $UB_{o_{En}} = 500$ mJ.

quantitatively compare different FSMs for requirement enforcement. We used PRISM for such probabilistic verification problems. Our evaluation shows that it is very difficult to understand which FSMs are superior with respect to individual and which for combined enforcement of multiple requirements. In the future, we would like to work on techniques for the automatic generation of enforcement FSMs with multiple probabilistic verification goals (a multi-objective optimization problem) for the enforcement of one or multiple requirements at a time.

References

1. Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the Logical Characterisation of Performability Properties. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9–15, 2000, Proceedings*, volume 1853 of *Lecture Notes in Computer Science*, pages 780–792. Springer, 2000.
2. Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime Enforcement for Reactive Systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *Lecture Notes in Computer Science*, pages 533–548. Springer, 2015.
3. Khalil Esper, Stefan Wildermann, and Jürgen Teich. A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems. In *Second Workshop on Next Generation Real-Time Embedded Systems, NG-RES@HiPEAC 2021, January 20, 2021, Budapest, Hungary*, volume 87 of *OASIcs*, pages 1:1–1:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
4. Khalil Esper, Stefan Wildermann, and Jürgen Teich. Enforcement FSMs – Specification and Verification of Non-Functional Properties of Program Executions on MPSoCs. In *19th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE'21)*, 2021.
5. Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *Proceedings of the 36th International Conference on Software Engineering*, pages 299–310, 2014.
6. Hans Hansson and Bengt Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
7. 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 Computer Society, 2015.
8. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Quantitative analysis with the probabilistic model checker PRISM. *Electron. Notes Theor. Comput. Sci.*, 153(2):5–31, 2006.
9. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification – 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.
10. David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.
11. Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Automated control of multiple software goals using multiple actuators. In Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman, editors, *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 373–384. ACM, 2017.
12. Sumit K. Mandal, Ganapati Bhat, Janardhan Rao Doppa, Partha Pratim Pande, and Ümit Y. Ogras. An energy-aware online learning framework for resource management in heterogeneous platforms. *ACM Trans. Design Autom. Electr. Syst.*, 25(3):28:1–28:26, 2020.
13. Srinivas Pinisetty, Partha S. Roop, Vidula Sawant, and Gerardo Schneider. Security of pacemakers using runtime verification. In *16th ACM/IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2018, Beijing, China, October 15-18, 2018*, pages 51–61. IEEE, 2018.
14. Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard von Hanxleden. Runtime enforcement of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 16(5s):178:1–178:25, 2017.

15 Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Stavros Tripakis, and Reinhard von Hanxleden. Runtime enforcement of reactive systems using synchronous enforcers. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10–14, 2017*, pages 80–89. ACM, 2017.

16 Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

17 Jürgen Teich, Michael Glaß, Sascha Roloff, Wolfgang Schröder-Preikschat, Gregor Snelting, Andreas Weichslgartner, and Stefan Wildermann. Language and Compilation of Parallel Programs for *-Predictable MPSoC Execution Using Invasive Computing. In *10th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip, MCSOC 2016, Lyon, France, September 21–23, 2016*, pages 313–320. IEEE Computer Society, 2016.

18 Jürgen Teich, Jörg Henkel, Andreas Herkersdorf, Doris Schmitt-Landsiedel, Wolfgang Schröder-Preikschat, and Gregor Snelting. Invasive computing: An overview. In Michael Hübner and Jürgen Becker, editors, *Multiprocessor System-on-Chip - Hardware Design and Tool Integration*, pages 241–268. Springer, 2011.

19 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 *A Journey of Embedded and Cyber-Physical Systems*, pages 125–149. Springer, 2021.

20 Jürgen Teich, Behnaz Pourmohseni, Oliver Keszöcze, Jan Spieck, and Stefan Wildermann. Run-Time Enforcement of Non-Functional Application Requirements in Heterogeneous Many-Core Systems. In *25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, January 13–16, 2020*, pages 629–636. IEEE, 2020.

Overlapping-Horizon MPC: A Novel Approach to Computational Constraints in Real-Time Predictive Control

Alberto Leva
DEIB, Politecnico di Milano, Italy

Simone Formentin
DEIB, Politecnico di Milano, Italy

Silvano Seva
DEIB, Politecnico di Milano, Italy

Abstract

Model predictive control (MPC) represents the state of the art technology for multivariable systems subject to hard signal constraints. Nonetheless, in many real-time applications MPC cannot be employed as the minimum acceptable sampling frequency is not compatible with the computational limits of the available hardware, *i.e.,* the optimisation task cannot be accomplished in one sampling period. In this paper we generalise the classical receding-horizon MPC *rationale* to the case where $n > 1$ sampling intervals are required to compute the control trajectory. We call our scheme *Overlapping-horizon MPC* – OH-MPC for short – and we numerically show its attitude at providing a tunable trade-off between optimisation quality and real-time capabilities.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Information systems → Process control systems

Keywords and phrases real-time control, model predictive control

Digital Object Identifier 10.4230/OASIcs.NG-RES.2022.3

1 Introduction and background

In modern control applications, Model Predictive Control (MPC) is the approach of election to optimal feedback regulation of multivariable (possibly nonlinear) systems subject to input and output constraints [1]. The key ingredient of all MPC strategies is the so-called *Receding-Horizon* policy, hereinafter RH-MPC for short. According to this policy, at each control step occurring at fixed period, the control signal trajectory is computed by considering its (predicted) impact on the state of the system over a future time window of finite length. Then, only the first sample of the optimal control sequence is applied to the system, whereas the whole input trajectory is re-optimised over a moved prediction horizon at the following sampling instant [9].

Despite the uncountable successful applications of such an approach [6], practical implementations of MPC schemes still present tough challenges for many real-world applications [5], among which: integration with data-driven model learning procedures [11], automatic tuning of MPC weights [8] and efficient on-line optimisation [12]. In particular, the latter represent a strong limitation for all applications where limited computational resources are available and a minimum sampling frequency is fixed by the dynamics at hand, in that the optimisation task *must* be accomplished in one period. Indeed, the expansion of MPC toward real-time systems on the one hand, and large-scale problems on the other, make the problem of allotting computational resource timely far more relevant than it was in the past, see e.g. [10].

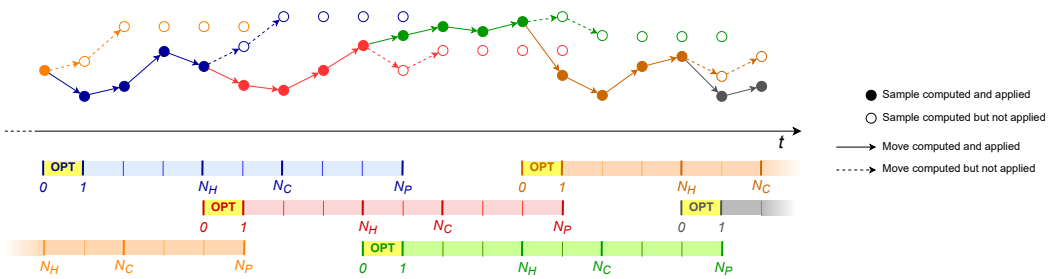

Figure 1 Role of the hold horizon N_H assuming that the optimisation problem (OPT) can be solved in one step; since in this particular example $N_H < N_C$, not all the computed control moves are applied; solid circles denote the control sequence actually fed to the process.

In [7], a suitable MPC scheme was proposed to overcome the above issues, by allowing the control action to be updated only "sporadically", standing on the assumption that an inner control layer is available to regulate the system, while the outer MPC is running in open loop. Another approach is to embed the predictive controller within a general event-based/asynchronous framework (see, e.g., [3] and [13]), thus limiting the number of computations but, at the same time, making the analysis unnecessarily complicated for a system whose signals are periodically sampled and with no network-related problems, like event-triggering or communication delays [14]. Moreover and most important, in sporadic and event-based strategies, when a new optimisation is required, this still must be done *within one control step*.

In order to deal with the problem of limited resources, in this work we propose a *generalisation* of the standard RH control scheme, for the case where $n > 1$ samples are required to perform the optimisation over the selected prediction horizon N_P. The proposed generalised scheme will be named *Overlapping-Horizon MPC* (OH-MPC) hereafter, in that its key idea is to start optimising the new input sequence while the previous control is applied – along this interval the old and new sequences are thus "overlapped", whence the name – and switch to the updated control trajectory as soon as this is available. In particular, by defining a *hold horizon* N_H as the number of samples of the computed input sequence that is actually applied to the system, we will discuss that, for $2n-1 \leq N_H \leq N_P$, we can span with continuity (of course quantised in steps) between two *extrema*. One is the classical RH-MPC policy; the other (that we name herein *Open-Loop MPC* or OL-MPC for short) consists of applying the entire sequence of control samples as coming from the optimisation over the prediction window. In addition and most important, then, while spanning in between the said *extrema* we can always comply with the computational constraints. Finally, we will show that, if $n = 1$, our generalisation reduces to the traditional RH-MPC scheme, with all the related properties. In a nutshell, to summarise, we can outline our proposal by the following statements.

- Taking RH-MPC and OL-MPC as *extrema*, we introduce and exploit an additional degree of freedom in the form of applying only a part (N_H) of the computed control horizon.
- We build on this to propose a methodology for addressing the case in which the solution of the optimisation problem necessarily spans more than one control time step as dictated by the physics of the problem, which apparently makes RH-MPC infeasible.

The remainder of the paper is organised as follows. In Section 2, the notion of hold horizon N_H for a predictive control scheme is defined and the OH-MPC scheme is introduced and explained. Then, Section 3 discusses the presented ideas, also in a view to providing some

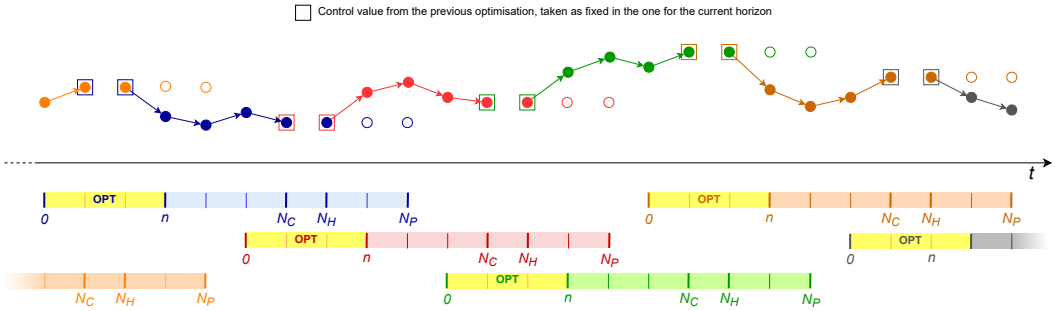

Figure 2 Horizons when the optimisation problem (OPT) cannot be solved in one step; here we set $N_C < N_H$ – hence all the computed control moves are applied – just for the sake of variety and completeness; again, solid circles denote the control sequence actually fed to the process.

application-oriented motivation for the additional degrees of freedom introduced. Section 4 provides a benchmark numerical example, to show that OH-MPC represents the best trade-off when limited computational resources are available. The paper is ended by some concluding remarks.

2 Overlapping-horizon MPC

In this section, we present the OH-MPC policy in general, and outline the corresponding algorithm. Let the system $\mathcal{S}$ to control be described in the discrete time domain by

$$x(t+1) = f(x(t), u(t)), \tag{1}$$

where $x \in \mathbb{R}^n$ represents the state vector, while f denotes a nonlinear function of the past state and input $u \in \mathbb{R}^m$. The standing assumption of the work is the following.

▶ **Assumption 1.** *The sampling time T_s of the application is dictated by the physical control problem, thus cannot be changed, and is so short that $n > 1$ samples are required to solve a state-feedback optimal control problem for* (1).

In such a situation, not infrequent in the applications, the standard receding horizon policy cannot be applied. In this section, we will therefore derive our scheme as a generalisation of traditional MPC, to deal with this specific – yet potentially critical – case.

The bove said, in its most general form the OH-MPC problem is stated as

$$\begin{aligned} \min_{u(n),\dots,u(N_c)} \quad & \frac{1}{N_p - n} \sum_{h=n}^{N_p} \mathcal{L}(x(h), u(h)) \\ \text{subject to :} \quad & x(t+1) = f(x(t), u(t)),\ t = 1, \dots, N_p, \\ & u(i) = \bar{u}(N_p - n + i),\ i = 1, \dots, n, \\ & x(j) \in \mathcal{X},\ u(j) \in \mathcal{U},\ j = n, \dots, N_p \end{aligned} \tag{2}$$

As can be seen, the objective of the control strategy is to minimise a (possibly economic [4]) cost, expressed by the time average of a nonlinear function $\mathcal{L}(\cdot, \cdot)$ of states and inputs over a prediction horizon N_p, so that x and u are constrained to belong to some feasibility sets, called $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{U} \subset \mathbb{R}^m$, respectively.

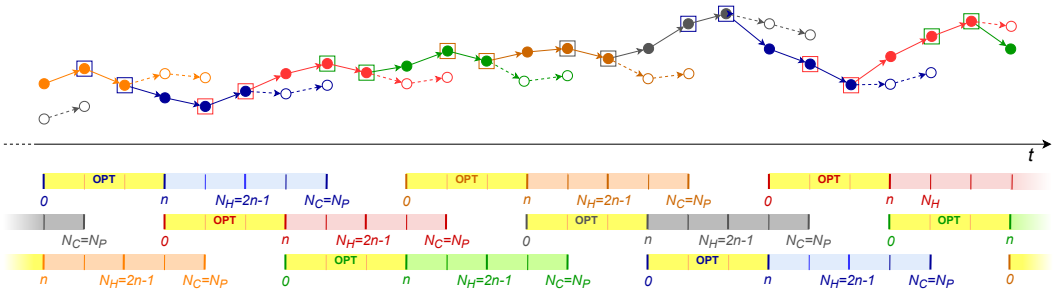

Figure 3 Horizons when the optimisation problem (OPT) cannot be solved in one step – particular but interesting case in which the optimisation problem can use all the available computational capability (we set here $N_C = N_P$ for simplicity, as this is a quite frequent choice); here too, solid circles denote the control sequence actually fed to the process.

The *key feature* of OH-MPC is that, since we assume n samples are required to return the optimisation results, the first $n-1$ samples of the input trajectory are set as the latest $n-1$ ones delivered at the previous iteration, namely, $\bar{u}(N_p-n+1),\ldots,\bar{u}(N_p)$. Such a sequence is used within the optimisation routine to estimate (through the model f) the starting trajectory of the state x and then set the initial condition, i.e., the predicted value of $x(n)$, for the input sequence to be optimised, that is, $u(n),\ldots,\bar{u}(N_c)$, where N_c denotes the *control horizon*. We further assume that, when $N_c < N_p$, the sequence $u(N_c+1),\ldots,u(N_p)$ is constant and equal to the most recent control $u(N_c)$.

Within this framework, there arises the need to introduce an additional degree of freedom N_H, called *hold horizon*, as the number of the computed control samples that are actually *applied* to the system. The role and meaning of the newly introduced hold horizon N_H is visually explained in Figure 1 for the traditional case of $n = 1$.

Even in this *scenario*, where the solution can be made available in one step, one might decide to apply a subset of the computed controls. For instance, in open-loop MPC (OL-MPC, hereafter), where the control input is applied in open-loop and updated only after the end of the prediction horizon, one might decide to apply only N_H samples of u (out of N_c) and then rerun the optimisation earlier. However, a typical choice of the hold horizon in OL-MPC with $N_C = N_P$ is $N_H = N_C$, namely, the input is optimised over the whole horizon and all the outcoming samples are applied to the system.

The choice of N_H becomes particularly interesting when $n > 1$ (see Figure 2). In fact, we can here highlight that the hold horizon must satisfy

$$2n-1 \le N_H \le N_C. \tag{3}$$

The upper bound is encountered in those situations like OL-MPC where, no matter how large n is, the control system is run in open-loop (and, typically, $N_C = N_P$). The lower bound might become instead a rather restrictive constraint, in that it limits the minimum amount of input samples that have to be injected into the system in open-loop, due to the computational constraints. The limit case $N_H = 2n-1$, where the control action is updated at the maximum possible frequency (dictated by n) is visually illustrated in Figure 3.

The overall strategy – that should now be clearly qualified as a generalisation of standard MPC, an aspect that will be further discussed in the next section – can be summarised as per Algorithm 1.

Algorithm 1 The OH-MPC algorithm.

```
   /* Problem acquisition & setup                                    */
 1 acquire the guaranteed No. n of steps to optimize;
 2 acquire the horizons N_P, N_C, N_H;
 3 acquire the model of the problem, i.e., L(·,·), f(·,·);
 4 acquire the initial conditions x(0), ū(t), t = N_P − n + 1, ..., N_P;
   /* Execution                                                      */
 5 while control system is running do
 6     solve (2) and collect u(n), ..., u(N_c);
 7     set u(i) = u(N_C), i = N_C + 1, ..., N_P;
 8     compose the sequence

           u(t) = { ū(N_P − n + t),   t = 1, ..., n,
                  { u(t),             t = n, ..., N_H

 9     apply the composed sequence to the system S in (1);
10     when the sequence ends, thus the new optimisation time is reached, set x(N_H) as
        the new x(0) and ū(t) = u(t), t = 1, ..., N_P;
11 end
```

3 Discussion and motivation

We devote this section to briefly discuss the possibilities opened by the OH-MPC policy, also providing – compatibly with the proposal-oriented scope of this paper – some operational motivation for its adoption.

To this end, we first observe that by suitably setting the involved horizons, OH-MPC specialises to both known problems and new ones, of importance discussed below, to exploit the introduced additional degrees of freedom. In particular, if $n = 1$ is feasible, the following staements hold true.

- With $n = 1$ and $N_H = 1$, OH-MPC apparently reduces to the classical RH-MPC problem.
- With $n = 1$ and $N_H = N_P$, the so-called "open-loop MPC" (OL-MPC) is obtained; here, we call this "one-step-compute" OL-MPC to stress the condition on n.
- In the latter case, taking N_P as given, choosing $N_C < N_P$ is the one degree of freedom to reduce the size of the optimisation problem.

This said, let us briefly review alternatives to the proposed OH-MPC in its application case of election, i.e., when $n = 1$ is not feasible, and for reasons too long to discuss herein, OL-MPC is not considered reliable enough. These alternatives are substantially two. One can either downsize the optimisation problem by reducing the prediction and/or the control horizon, or replace the said problem with local equivalents (for example, linearising in the vicinity of conveniently chosen operating points) that require less effort to be solved.

More interesting, and motivating for the presented research, is therefore to study the case in which $n = 1$ is infeasible, but at the same time – once again for a variety of possible reasons that we are not treating in this paper – the optimisation problem must be solved *as is*. Here, OH-MPC can be fruitfully exploited to provide the needed additional degrees of freedom.

With reference to Figure 2, we assume for the scope of this work that optimisations occur only when some horizon elapses, i.e., we exclude for the moment techniques involving event-triggered optimisations like the sporadic one above.

This said, first the hold horizon N_H can be made both larger and smaller than the control one N_C. This allows to use the former to dictate the (constant) cadence of optimisations, while - for the latter - one can take the value dictated by the optimisation problem definition.

Second, when an optimisation is in progress, the control samples applied to the process come from the previous one, which – as will be shown – is tendentiously better than e.g. just holding the control signal till a new sequence is ready.

Third, once n is reliably obtained e.g. by profiling techniques aimed at WCET (Worst Case Execution Time) estimation [2], N_H allows to optimally use all the available resources. In detail, setting $N_P = N_C$ and $N_H = 2n - 1$ allocates all the computation time to perform the maximum number of optimisations – as illustrated in Figure 3 – compatibly with the time needed to compute one.

4 A proof-of-concept example

We now show a simulation case study – deliberately minimalistic – to witness the usefulness of OH-MPC. The system to be controlled is the linear time-invariant plant

$$\begin{cases} x_1(t+1) &= 0.9 \cdot x_1(t) + 0.1 \cdot u(t) \\ x_2(t+1) &= 0.6 \cdot x_1(t) + 0.4 \cdot x_2(t) \\ y(t) &= x_2(t), \end{cases} \tag{4}$$

while the control problem we wish to solve is of the form in (2), with

$$\mathcal{L}(x(t), u(t)) = x(t)^T Q x(t) + u(t)^T R u(t),$$

$R = 1$, and

$$Q = \begin{bmatrix} 10 & 0 \\ 0 & 100 \end{bmatrix}.$$

In the addressed problem, we also set a constraint on the value of the control variable u, that must lie within the [-1,1] range. No constraints on x are given, namely $\mathcal{X} \equiv \mathbb{R}^2$. We consider the achieved closed-loop properties in terms of *disturbance rejection* using different model predictive controllers, via an experiment with a unitary matched load disturbance applied to the process at $t = 0$ and removed at $t = 30$. The prediction horizon N_P is set equal to 9 steps.

Case study no. 1. Let us assume that the computation time needed for the solution of the control problem amounts to three steps, thus $n = 3$.

In order to fairly assess the performance of OH-MPC, we consider a comparison among the following strategies, all with $N_P = N_C$:

1. an *oracle (in fact infeasible) solution*, given by an **OL-MPC** approach, where we assume – contrary to the OH-MPC hypothesis – that the solution can be computed in one step, that is $n = 1$, and $N_H = N_P = N_C$;

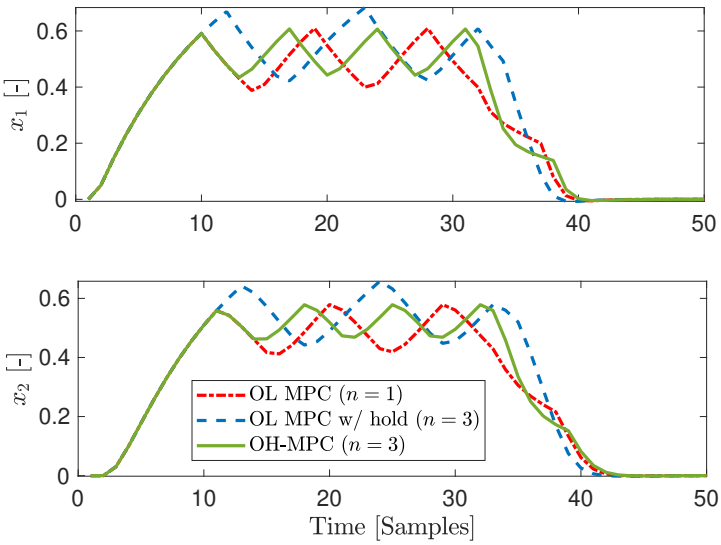

Figure 4 *Case study no. 1.* Time responses of state x to a step disturbance with different MPC strategies: OL-MPC (unfeasible), OL-MPC with final n-step holding and OH-MPC (feasible). OH-MPC clearly provides the best feasible performance.

2. a *baseline feasible solution*, in which the constraint $n = 3$ is taken into account, named **OL-MPC w/ hold**. This solution is obtained starting from the one above, but, at the beginning of a new optimization, the control variable is held equal to the latest available control sample for $n - 1$ steps, waiting for the new solution;
3. the **OH-MPC** approach with $N_H = N_P = N_C$.

The time histories of the state trajectories are illustrated in Figure 4, where it can be clearly seen that the OH-MPC solution remains limited as compared to the baseline one, even if it is not able to reach the attenuation level provided by OL-MPC. It is however worth stressing that this OL-MPC is calculating the optimal solution in one step only, which is infeasible under the assumption that $n = 3$.

The fact that OH-MPC can be considered as a good trade-off solution (namely, the best alternative if the constraint $n = 3$ is active) is further confirmed by comparing the value of the optimal cost

$$J = \sum_{t=1}^{N} x(t)^T Q x(t) + u(t)^T R u(t),$$

where N denotes the length of the whole experiment, in Figure 5.

Case study no. 2. Considering the same system of the previous example, we now suppose that the designer's desire is to use all the available computational power to run the controller at the maximum frequency, however under the physical constraint that the computation time needed for the solution of the control problem amounts to five steps ($n = 5$), which can be considered even more challenging than the previous situation, if $N_P = 9$. Again, we consider a comparison with the *baseline* strategy OL-MPC w/hold, but now – to use all the available time – $N_H = 2n - 1 = 9$.

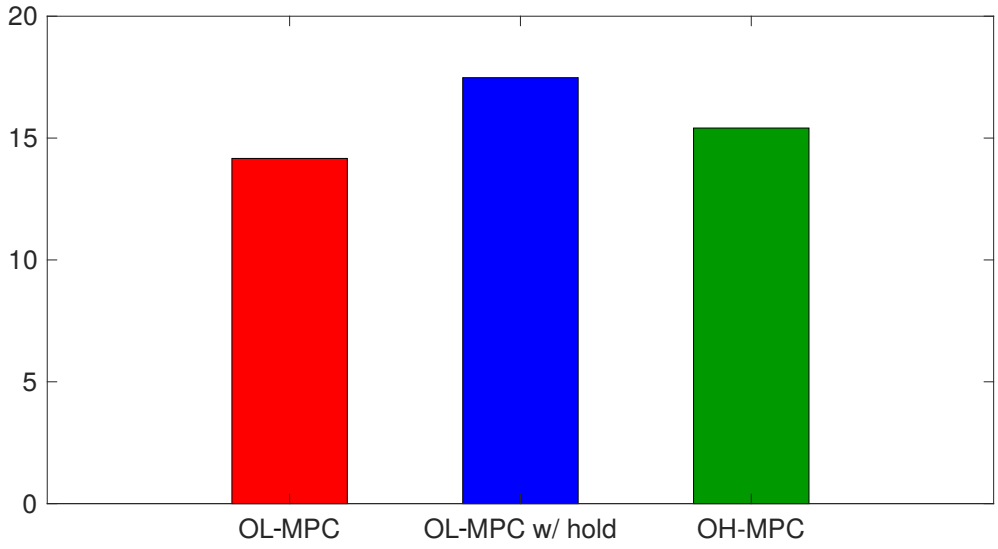

Figure 5 *Case study no. 1.* Cost J for different MPC strategies: OL-MPC (unfeasible), OL-MPC with final $(n-1)$-step holding and OH-MPC (feasible). OH-MPC clearly provides the best feasible performance.

The time histories of the state trajectories are illustrated in Figure 6, where it can be observed that the OH-MPC solution remains better than the baseline also in such a critical scenario. A confirmation of this fact can be found in Figure 7, where the cost J is highlighted, thus confirming OH-MPC gets closer to the ideal situation.

Summing up, OH-MPC appears to yield intermediate results between the (infeasible) one-step-computing solution ($n = 1$) and the baseline one based just on holding the last control value.

5 Conclusions and future work

We presented an MPC scheme, named *Overlapping-Horizon* MPC to comply with the case in which the optimisation problem cannot be solved in one control time step, or said otherwise, its solution can be guaranteed to terminate only within a number $n > 1$ of such steps. The addressed case is potentially critical and occurs whenever computational resource limitations can be relevant, whence the usefulness of OH-MPC in real-time control systems.

Although the presented research is still at a preliminary stage, OH-MPC definitely exhibits interesting properties, in particular not requiring to modify the optimisation problem with respect to its "original" formulation, nor to alter the sampling time with respect to the value dictated by the control system physics (another desirable propoerty in the real-time case).

Future work will be directed toward a formal analysis of the OH-MPC scheme, possibly articulating the study per characteristics of the controlled system and/or cost function, as well as toward an engineered realisation, suitable for implementation and testing on real plants.

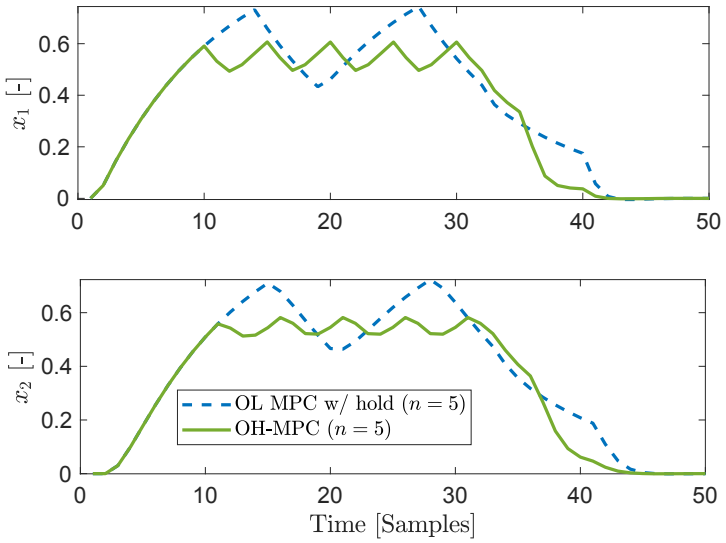

Figure 6 *Case study no. 2.* Time responses of state x to a step disturbance with different MPC strategies: OL-MPC with final n-step holding and OH-MPC.

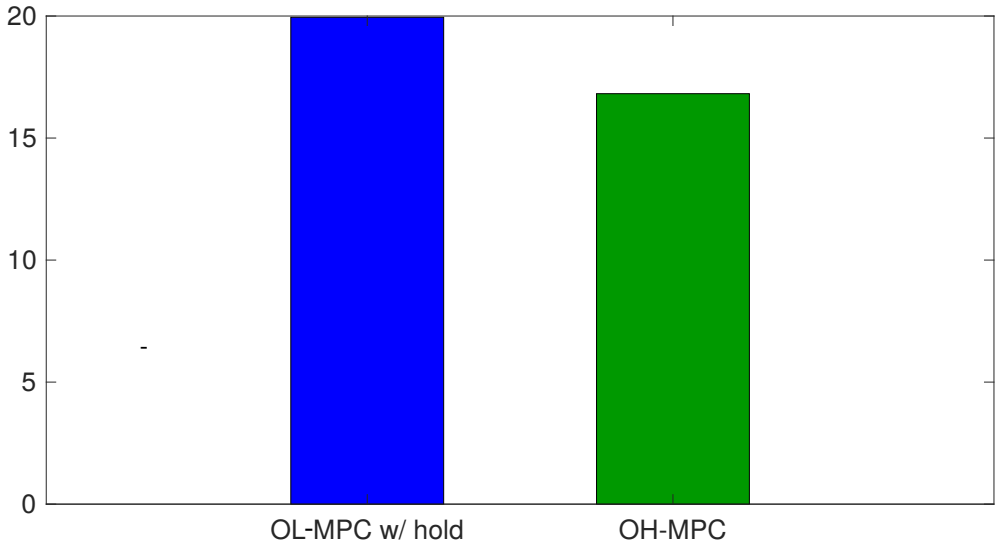

Figure 7 *Case study no. 2.* Cost J for different MPC strategies: OL-MPC with final $(n-1)$-step holding and OH-MPC (feasible).

References

1. F. Allgöwer and A. Zheng. *Nonlinear model predictive control*. Birkhäuser, 2012.
2. A. Bonci, S. Longhi, G. Nabissi, and G.A. Scala. Execution time of optimal controls in hard real time, a minimal execution time solution for nonlinear SDRE. *IEEE Access*, 8:158008–158025, 2020.
3. F.D. Brunner, W. Heemels, and F. Allgöwer. Robust event-triggered MPC for constrained linear discrete-time systems with guaranteed average sampling rate. *IFAC-PapersOnLine*, 48(23):117–122, 2015.
4. M. Ellis, H. Durand, and P.D. Christofides. A tutorial review of economic model predictive control methods. *Journal of Process Control*, 24(8):1156–1178, 2014.
5. M.G. Forbes, R.S. Patwardhan, H. Hamadah, and R.B. Gopaluni. Model predictive control in industry: Challenges and opportunities. *IFAC-PapersOnLine*, 48(8):531–538, 2015.
6. S.K. Lahiri. *Multivariable predictive control: Applications in industry*. John Wiley & Sons, 2017.
7. A. Leva, F.M. Benzi, V. Magagnotti, and G. Vismara. Sporadic Model Predictive Control. *IFAC-PapersOnLine*, 50(1):4887–4892, 2017.
8. A. Lucchini, S. Formentin, M. Corno, D. Piga, and S.M. Savaresi. Torque vectoring for high-performance electric vehicles: an efficient MPC calibration. *IEEE Control Systems Letters*, 4(3):725–730, 2020.
9. J. Mattingley, Y. Wang, and S. Boyd. Receding horizon control. *IEEE Control Systems Magazine*, 31(3):52–65, 2011.
10. P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin. DMAC: Deadline-miss-aware control. In *Proc. 31st Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
11. D. Piga, M. Forgione, S. Formentin, and A. Bemporad. Performance-oriented model learning for data-driven MPC design. *IEEE control systems letters*, 3(3):577–582, 2019.
12. S.J. Wright. Efficient convex optimization for linear MPC. In *Handbook of model predictive control*, pages 287–303. Springer, 2019.
13. J. Yoo and K.H. Johansson. Event-triggered model predictive control with a statistical learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.
14. D. Zhang, P. Shi, Q. Wang, and L. Yu. Analysis and synthesis of networked control systems: A survey of recent advances and challenges. *ISA transactions*, 66:376–392, 2017.

Ahead-Of-Real-Time (ART): A Methodology for Static Reduction of Worst-Case Execution Time

Daniele Cattaneo
DEIB, Politecnico di Milano, Italy

Gabriele Magnani
DEIB, Politecnico di Milano, Italy

Stefano Cherubin
School of Computing, Edinburgh Napier University, UK

Giovanni Agosta
DEIB, Politecnico di Milano, Italy

Abstract

Precision tuning is an approximate computing technique for trading precision with lower execution time, and it has been increasingly important in embedded and high-performance computing applications. In particular, embedded applications benefit from lower precision in order to reduce or remove the dependency on computationally-expensive data types such as floating point. Amongst such applications, an important fraction are mission-critical tasks, such as control systems for vehicles or medical use-cases. In this context, the usefulness of precision tuning is limited by concerns about verificability of real-time and quality-of-service constraints. However, with the introduction of optimisations techniques based on integer linear programming and rigorous WCET (Worst-Case Execution Time) models, these constraints not only can be verified automatically, but it becomes possible to use precision tuning to automatically enforce these constraints even when not previously possible. In this work, we show how to combine precision tuning with WCET analysis to enforce a limit on the execution time by using a constraint-based code optimisation pass with a state-of-the-art precision tuning framework.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Software and its engineering → Compilers; Mathematics of computing → Mathematical software

Keywords and phrases Approximate Computing, Precision Tuning, Worst-Case Execution Time

Digital Object Identifier 10.4230/OASIcs.NG-RES.2022.4

Funding This work is supported by the European Commission and the Italian Ministry of Economic Development (MISE) under the EuroHPC TEXTAROSSA project (G.A. 956831).

1 Introduction

In critical and mixed-critical applications, at least some of the tasks that compose the system workload need to respect strict quality-of-service constraints, particularly in terms of latency. These constraints may be expressed in terms of deadlines, and a maximum probability of missing them. To ensure that deadlines are respected, worst-case execution time (WCET) analysis can be used. In tasks that heavily rely on floating point arithmetic, it is possible to improve the execution time (and other extra-functional properties such as energy-to-solution) by applying approximate computing techniques such as precision tuning [4]. This technique enables trade-offs between computation precision and the aforementioned extra-functional properties, by allowing some or all the computations to be performed using different data types than the ones specified in the application source code. While this kind of transformation is usually performed manually by embedded system developers, it is an error prone operation, and it is difficult to manually gauge the right data type for each operation even for an

experienced developer, when the operation chains are long. However, recent developments in compiler-assisted precision tuning [5] have introduced not only tools to perform the data type adaptation, but also to explore the vast design space opened by the ability to provide different data types for different code fragments, via integer linear programming (ILP) [2].

Specifically, this last approach involves the construction of a mathematical model of the program being compiled, which is then fed into a linear programming solver to produce the final data type assignments. The mathematical model computes a parametric estimation of the relative execution time slowdown and of the quantisation error, with respect to a fully floating-point-based implementation. These estimates are not useful in general to gauge the real error or execution time of the generated program, because the per-instruction coefficients employed are ratios rather than absolute errors or instruction timings.

However, in principle, if we replace such coefficients with values corresponding to the actual error being inserted by a computation or its actual execution time in clock cycles, the optimiser will gain visibility to a realistic estimation of these metrics. Therefore, this enables optimising the program for a given maximum error or execution time. Of these two metrics, the maximum error is often an overestimate that is not fine-grained enough to accurately predict the actual error on realistic data. This problem arises from the fact that error estimates not only depend on the specific data type, but on the data itself [5]. On the other hand, execution time only depends on the instruction selection performed by the compiler and the microarchitecture of the target processing unit, which are both deterministic factors under our control. As a result, conservative estimates of the execution time are often accurate down to an acceptable error percentage [11].

Additionally, optimising for a target execution time is useful in real-time systems, where error-tolerant tasks that must be completed under a certain deadline are plentiful and common. For instance, closed-loop control algorithms of vehicles or weapons often must rely on inherently noisy data from sensors such as gyroscopes or LIDAR systems [8]. Therefore imprecisions in the output are acceptable as long as they are not significant with respect to the input data itself. Our approach can be combined with WCET analysis to statically ensure that the execution time of a task stands below a given boundary at a low cost in terms of error (under 1%) – or, conversely, to prove that precision tuning is not enough to ensure meeting a timing constraint.

Contribution

In this work we adapt state-of-the-art optimisation-based precision tuning techniques taking into account the real execution time of an example application and constrain it to a given upper bound, a new methodology which we call Ahead of Real Time (ART) optimisation. To that end, we provide a theoretical model that can be used to construct such an optimiser.

We demonstrate the practicality of our approach by applying it to a subset of the PolyBench [15] benchmark suite. We empirically demonstrate that the execution time estimated by our methodology matches within a reasonable margin of accuracy (under 30%) the actual execution time on a microcontroller core representative of the hardware used in safety-critical applications, and that our approach allows to meet a timing deadline with a low loss of precision, below 1%.

Organisation of the paper

The rest of this paper is organised as follows. In Section 2 we discuss related works in the field of precision tuning and WCET analysis for meeting timing constraints. Subsequently, in Section 3 we discuss the mathematical models underlying our solution and in Section 4 we show experimental data that validates the methodology as a whole. Finally, we summarise our conclusions and discuss follow-up work in Section 5.

2 Related Works

Approximate computing is a broad field that is attracting a large amount of effort from research groups worldwide. Its increasing relevance is a consequence of the growing spread of error-tolerant applications in different domains, as well as of the rise of energy cost of ICT systems, which threatens to grow to over 20% the total world energy demand by 2030 [9]. As a result, a wide range of hardware and software techniques are being scrutinised. While the full discussion of this topic goes well beyond the scope of this work, a recent survey by Stanley-Marbell *et al.* provides an overview of the most relevant approaches [14]. Within approximate computing, precision tuning is a technique that lends itself to a wide applicability, as it can be employed whenever a computation is performed using data types that are wider than the actual application needs, as well as to automated application, since the compiler, appropriately instructed as to the actual precision needed for the results, can automatically infer the minimum data type and width, and then explore the cost of switching between different data types to obtain an optimal solution. Once more, a full discussion of the topic would require too much space for this work, so we refer the interested reader to a recent survey that goes into greater detail on precision tuning and the tools that support it [4].

Broadly speaking, precision tuning approaches can be classified according to [4] as *static* or *dynamic* depending on whether dynamic compilation is used to improve the accuracy of the precision needs by taking into account variations in the workload, or not. The dynamic approach is not suitable for critical and mixed-critical scenarios, since by nature it alters the execution time whenever a dynamic compilation is performed. Thus, we constrain our discussion to *static precision tuning*.

Within the techniques that are more appropriate for critical and mixed-critical embedded systems, another taxonomic division occurs between approaches that leverage custom hardware and those that address microcontrollers. The main difference is that in the former case the target output is a hardware description language, in the latter the target is embedded C or assembly code. While hardware-oriented tools are certainly relevant, for the purpose of this work we limit our scenario to the more common case of systems built out of off-the-shelf microcontrollers provided by semiconductor manufacturers such as Texas Instruments, ST Microelectronics, or Nordic Semiconductors.

To address this scenario, static precision tuning tools gather the information required to apply their optimisations to the code without requiring extensive testing, but rather through static analyses. Among them, the most representative of the state of the art are *Precimonious* [13], *Daisy* [7], and *TAFFO* [6], which are all candidates for use in embedded systems scenarios. Of these, Daisy operates as a source-to-source compiler, which can be considered a drawback, since it may prevent information from the source from reaching the compiler optimisation phases directly, possibly introducing overheads. Precimonious and TAFFO operate as LLVM plugins, thus providing a greater degree of integration. However, Precimonious public development has not progressed since 2016, making it incompatible with modern releases of the LLVM compiler – it requires LLVM 3, whereas TAFFO can work with recent versions of the compiler framework, including both versions 11 and 12. Therefore, we select TAFFO as the baseline tool for the work presented here.

Regarding the WCET estimation methodologies, a large amount of work is available from the literature. A good taxonomy can be found in [1], where the state of the art in the field is thoroughly analysed. In particular, it is possible to distinguish *static analysis* and *measurement-based* methodologies, as well as *hybrid* approaches on one hand, and *deterministic* and *probabilistic* approaches on the other. These can be combined to form six possible different methodologies.

In practice, though, *measurement-based deterministic timing analysis* (MBDTA) is most commonly employed in the industry, followed by *static deterministic timing analysis* (SDTA), which is used for simpler hardware and software systems. MBDTA still has limitations in that it requires good input data set, and, from the point of view of our work, the need to perform measurements makes it unfeasible in the exploration of a huge design space. While probabilistic methods are gaining increasing momentum [3], *static* methods are still comparatively less developed than measurement-based ones. Therefore the probabilistic approach is less suitable for our purpose.

In conclusion, the need to analyse a huge number of solutions in the design space, and the relative immaturity of static probabilistic timing analysis leads us to choose SDTA as the basic methodology for the WCET analysis performed in this work. Yet, the considerations and the proposed methodology would fit well with any static timing analysis, as long as the analysis method could be used as a constraint in the integer linear programming approach used to solve the design space exploration problem.

3 Proposed model and methdology

In this section, we show how approximate computing can be used to enable the trade-off between numerical precision and WCET. We achieve this by applying precision tuning through ILP model optimisation.

We demonstrate the effectiveness of our approach by implementing it within a compiler-based precision tuning tool – TAFFO. First, we briefly introduce TAFFO and the state-of-the-art ILP model on which our new methodology is based upon. Then, we describe how the ART-ILP model is adjusted and modified to provide realistic execution time estimates and optimisations. Finally, we discuss how to exploit the ART methodology to leverage the precision-WCET trade-off.

3.1 The architecture of TAFFO

TAFFO is a state-of-the art precision tuning toolkit based on the LLVM compiler framework [10]. TAFFO is independent from the program source language due to its analyses being based on the LLVM-IR intermediate language, and it supports automatic tuning using both floating point and fixed point data types. It consists of five independent passes, which take the form of a loadable plugin for LLVM-based compilers. The pass-based architecture allows TAFFO to be expandable, easy to use and robust.

The TAFFO tool requires the programmer to define some contextual information related to the value ranges of the inputs and the extent of the area of code that needs to be tuned. This information is inserted through annotation of the source code. The first pass of TAFFO, called *Initializer*, reads such annotations and converts them in the internal data structures required by the rest of TAFFO.

From the user-provided information, TAFFO then analyses the program to conservatively derive the numerical intervals each variable in the program will have at runtime. This pass is called the *Value Range Analysis* or VRA. The information derived by the VRA is then

used to determine which reduced-precision data type to use for each variable, a procedure called *Data Type Allocation* (DTA). The DTA can operate based on two different algorithms: a peephole-based algorithm which always chooses the fixed-point data type with the highest valid point position for each variable, and a new optimiser based on ILP techniques [2]. This step is able to optimally mix floating point and fixed point data types by exploiting a mathematical model of how changes to the precision mix affect the speedup and the output error. The software uses the Google OR-Tools C++ framework [12] as model solver backend.

Down in the pipeline, the *Conversion* pass is responsible for applying the data type changes on the program being tuned, and finally the *Feedback Estimator* pass statically analyses the error using state-of-the-art estimation methods [5].

3.2 The ART-ILP model

In the intermediate representation of a compiler, a program is described in terms of a control flow graph, where each node is called a basic block and contains a list of instructions. This kind of representation is not directly suitable for modelling the execution time of a program or its error-tolerance, a different formulation is needed. In the following we focus on the execution time, and we present the model used by the DTA pass of TAFFO.

Let us consider a single basic block B, represented as a list of instructions. There are various kinds of instructions, but for the purposes of precision tuning we only consider mathematical instructions and *cast* or type conversion instructions. These are the only instructions that are affected by the precision tuning optimisation. Typically, cast instructions are inserted only when a variable in the intermediate representation needs to be converted from one type to another. Without loss of generality, we consider all mathematical instructions to have a single data type, which applies to all of the operands and its result value. Due to this constraint, which casts are present in the program only depends on the data type assignment of each mathematical instruction.

From these considerations we can begin building a mathematical model describing a program, specifically an *integer linear programming* (ILP) problem. ILP problems have the following form:

$$\begin{aligned} & k_{1,1}x_1 + k_{2,1}x_2 + \cdots + k_{n,1}x_n && \in [l_1, u_1) \\ & k_{1,2}x_1 + k_{2,2}x_2 + \cdots + k_{n,2}x_n && \in [l_2, u_2) \\ & \cdots && \cdots \\ & k_{1,m}x_1 + k_{2,m}x_2 + \cdots + k_{n,m}x_n && \in [l_m, u_m) \end{aligned}$$

$$\min \quad \textstyle\sum_i^n w_i x_i.$$

The first set of disequalities are called the *constraints*, while the final expression is called *objective function*, and represents the quantity that the optimiser must attempt to minimise. Each constant $k_{i,j}$ and w_i is called a *coefficient* or *weight*. The goal of the optimiser is to find an assignment to each *variable* x_i that both satisfies the constraint and minimises the objective function. Additionally, each x_i must be an integer.

Now, in order to exploit such a model for precision tuning, we introduce multiple sets of variables that represent every possible type choice for each instruction. For each mathematical instruction a, and for each data type t, we introduce a variable $x_{a,t} \in [0, 1]$ that represents the choice of using the given data type for that instruction. Each type choice is mutually exclusive, and as a result we must introduce the following constraints:

$$\textstyle\sum_t x_{a,t} = 1 \quad \forall i \in B.$$

In order to take into account the execution time in the optimisation, such variables must appear in the objective function. As a minimum, we must introduce the following term:

$$T_{m,B} = \sum_{i \in M(B),d} \text{time}(i,t) \times x_{i,t}$$

where $\text{time}(i,t)$ is the average execution time of instruction i with data type t, and $M(B)$ is the set of mathematical instructions in M. Therefore, T_m is the execution time of all mathematical instructions in a given basic block.

This partial expression of the execution time must be augmented by a second term for the execution time devoted to cast operations. In fact, an excessive amount of casts may counterbalance any advantage provided by lowering precision. Therefore, before each mathematical instruction, we insert in our model additional *virtual cast* instructions, used to represent the execution time of casts whenever they are needed. To take into account the varying data types between two instructions i and i' and the casts needed on the operands, we introduce a constraint for each possible pair of different types t, t' with this form:

$$x_{i,t} + x_{i',t'} \leq y_{i,t,i',t'} + 1.$$

The $y_{i,t,i',t'}$ variable will be set to 1 during the optimisation process if a cast is necessary. Therefore, in the objective function the time required for performing casts is expressed by the following term:

$$T_{c,B} = \sum_{i,i' \in M(B)} \sum_{t,t':t \neq t'} \text{time}(i,t,t') \times y_{i,t,i',t'}.$$

An additional term in the objective function represents the error, in terms of a representation-independent metric called the *IEBW*, which we won't describe here because it's not involved in our improvements to the existing methodology. We denote this term as E_B. In the objective function, the three terms $T_{c,B}$, $T_{m,B}$ and E_B are summed together and their balance is determined by two weights, W_1 and W_2, referring respectively to the execution time component and the error component. Therefore, the objective function for optimising a basic block B appears as follows:

$$\min \quad W_1 \left(T_{c,B} + T_{m,B}\right) \frac{1}{N_1} - W_2 E_B \frac{1}{N_2}.$$

Two parameters N_1 and N_2 are added to normalise the weights of the two terms (time and error) to make them comparable. The values of N_1 and N_2 are equal to the maximum possible estimated execution time and error respectively.

3.3 The ART approach

The model we have just described only involves simple basic blocks, which only represent straight-line pieces of code without control structures such as loops, conditional statements or branches. The execution time of a serial program can be modelled in a fairly simple way. Let us denote with $\text{time}(B)$ the time required for executing a basic block B, and with N_B the number of times the basic block is executed in a given execution trace E. Therefore, the execution time of E is the following:

$$\text{time}(E) = \sum_{B} N_B \times \text{time}(B).$$

On an in-order CPU architecture such as a microcontroller architecture, the execution time of a basic block B can be modelled with good accuracy as the sum of the individual execution times of each instruction i in the basic block:

$$\text{time}(B) = \sum_{i \in B} \text{time}(i).$$

Notice that in this work we do not consider out-of-order and multicore architectures, and we also ignore the effect of instruction and data caches.

In the linear programming model we described in Section 3.2, we further categorised the instructions in a basic block in three sets: mathematical instructions M, represented by x variables in the model, cast instructions C, represented by y variables in the model, and other instructions which do not appear in the model. Therefore, from a solution to the ILP model – which consists of assignments to the model's variables – we can estimate the execution time of a basic block with the following expression:

$$\text{time}(B) = T_{m,B} + T_{c,B} + T_{B \setminus (M \cup C)}.$$

This formulation adds a constant factor $T_{B \setminus (M \cup C)}$ that represents the execution time of instructions that are neither arithmetical instructions or cast instructions, and are not affected by the optimisation process. When also accounting the execution of an entire program, we must estimate the worst-case or upper-bound N_B for each basic block in the program, which we call $\max(N_B)$. This can be done in a conservative way by well-known control flow static analysis techniques, which are commonplace for WCET analysis [11].The estimation for the execution time thus becomes:

$$\text{time}(E) = \sum_{B} \max(N_B)(T_{m,B} + T_{c,B} + T_{B \setminus (M \cup C)}).$$

Notice that this expression is indeed in the form acceptable for a linear constraint. Therefore, we can statically impose a limit on the worst-case execution time (WCET) of a program by introducing the following constraint in the linear programming model:

$$\text{time}(E) \leq T_{max}.$$

4 Experimental Evaluation

To experimentally evaluate the ART methodology in practice, we performed a set of experiments aimed at testing the quality of the execution time estimation.

As example applications, we chose some selected benchmarks from the PolyBench/C suite, version 4.2.1 [15]. This benchmark suite consists of several programs written in the C programming language that encompass a large variety of computational kernels. Of the entire set of benchmarks, we chose the ones with the highest execution time variance depending on the optimisation parameters: *2mm*, *3mm*, *covariance*, *lu* and *nussinov*. The benchmarks are unmodified, exception done for the addition of the required annotations for TAFFO.

Hardware-wise, the platform targeted for the estimation was a STM3220G-EVAL ST Microelectronics embedded evaluation board, with a 120 MHz Cortex-M3 ARM processor, 128 KB of on-chip internal RAM, and 2 MB of external RAM.

The experiment was conducted as follows. First, the number of clock cycles required for every instruction was profiled on the embedded board by running a specifically-designed software. These metrics were intentionally increased by a fixed percentage (25%) to take

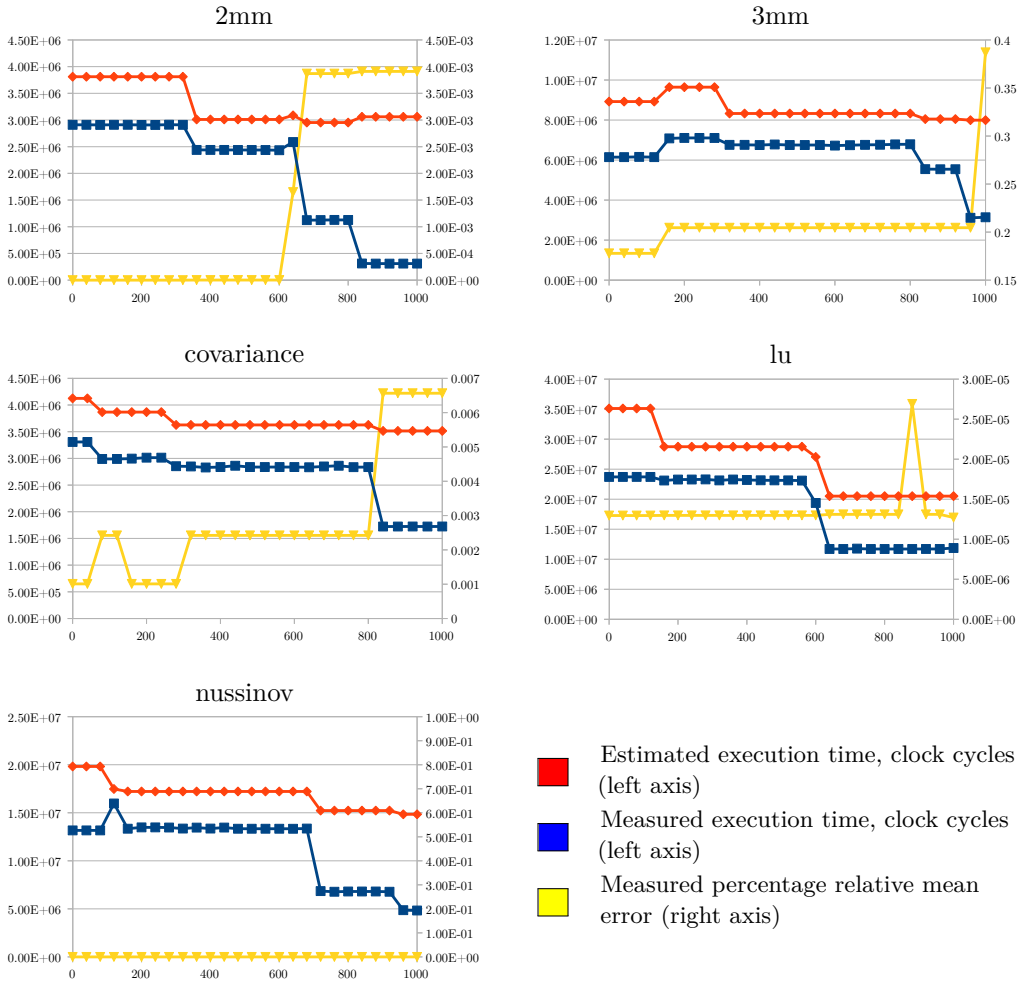

Figure 1 Time and error measurements compared with the time estimates provided by the model used in the ART methodology. On the horizontal axis is the value of the W_1 optimisation parameter. On the the two vertical axes, left to right, are clock cycles and percentage relative mean error.

into account the fact that additional instructions may be introduced by later program transformations performed in the compiler. This data is stored in a configuration file suitable for usage by the optimiser.

At this point, each of the benchmarks was compiled both without using TAFFO, and with TAFFO. Both compilations were performed using LLVM clang version 12.0.0. For what concerns the TAFFO compilations, each benchmark was compiled 25 times, every time with a different setting regarding the weight of the mathematical and cast execution component W_1 and the precision component W_2. We call these separate compilations *versions*. The initial value of W_1 was zero, and each subsequent compilation increased W_1 by 40 until reaching the value of 1000. W_2 was derived from W_1 via the equation $W_2 = 1000 - W_1$. Each version of each benchmark (included the non-mixed-precision version) is then run on the aforementioned embedded board. No supporting operating system is used except for the lightweight hardware abstraction layer provided by the manufacturer of the board. The execution time of each run and the output data from the computation performed by the benchmark is logged by means of the built-in serial port.

During the compilation of the mixed-precision versions, the TAFFO Data Type Allocation pass also computes the estimated execution time of the program.

In Section 4 we show, for each benchmark, the real and estimated execution times in clock cycles, and the percentage relative mean error in the output. The estimated execution time is consistently overestimated with respect to the real execution time. We believe this is due to two factors. Firstly, the 25% margin added to the cycle count of every instruction, which however is intentional to provide a safety margin. Secondly, the maximal basic block execution counts N_B are themselves overestimated by the static analyses we perform, based on the *scalar evolution* pass of LLVM.

Secondly, we observe that the execution time prediction is consistent with the measured execution time: speedups happen exactly when they are predicted by the model. The estimate of the amount of speedup with the increase of W_1 is however underestimated. This is primarily due to the overestimation of the N_B parameters, as we mentioned, since the ratio of overestimation is not consistent for each basic block. However, in general these are not issues for what concerns WCET estimation, as an overestimation is better than an underestimation in this context.

Finally, we observe that the error either remains constant or gradually increases with W_1 – or more properly, with the decreasing of W_1. Some momentary irregularities are observed in the *covariance* and *lu* benchmarks. This happens when the error and execution time terms of the objective function have similar values, due to the N_1 and N_2 normalisation parameters. In general, the error is lower than 1% for all benchmarks. This is consistent from the behaviour we expect from the integer-linear-programming-based optimiser.

From the data we can conclude that the ART methodology is effective for WCET optimisation, as the estimated execution time is indeed reflective of real execution time, and it is also conservative enough to provide an acceptable margin for handling perturbances such as non-maskable interrupts or other higher-priority concurrent tasks.

5 Conclusions

In this work we introduced and described the ART methodology, a way to exploit precision tuning to enforce worst-case execution time constraints on a given computational kernel or program. This methodology has been implemented as part of the TAFFO precision tuning framework, based on LLVM and the Google OR-Tools toolkit, and has been evaluated on an embedded-systems board by exploiting the PolyBench benchmark suite. The results highlighted the approach's ability to enforce a constraint on the worst-case execution time automatically by adjusting the precision of the data types used in the program.

Future improvements to this work encompass the usage of a similar methodology to also enforce a given boundary on the precision loss. Additionally, follow-up development include the development of a model that also supports out-of-order architectures, data and instruction caches, and parallel applications and architectures.

References

1 Jaume Abella et al. Wcet analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10. IEEE, 2015.

2 Daniele Cattaneo, Michele Chiari, Nicola Fossati, Stefano Cherubin, and Giovanni Agosta. Architecture-aware precision tuning with multiple number representation systems. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 673–678, 2021. doi:10.1109/DAC18074.2021.9586303.

3 Francisco J. Cazorla et al. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Comput. Surv.*, 52(1), February 2019. doi:10.1145/3301283.

4 Stefano Cherubin and Giovanni Agosta. Tools for reduced precision computation: a survey. *ACM Computing Surveys*, 53(2), April 2020. doi:10.1145/3381039.

5 Stefano Cherubin, Daniele Cattaneo, Michele Chiari, and Giovanni Agosta. Dynamic precision autotuning with TAFFO. *ACM Trans. Archit. Code Optim.*, 17(2), May 2020. doi:10.1145/3388785.

6 Stefano Cherubin, Daniele Cattaneo, Michele Chiari, Antonio Di Bello, and Giovanni Agosta. TAFFO: Tuning assistant for floating to fixed point optimization. *IEEE Embedded Systems Letters*, 2019. doi:10.1109/LES.2019.2913774.

7 Eva Darulova et al. Sound mixed-precision optimization with rewriting. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '18, pages 208–219, 2018. doi:10.1109/ICCPS.2018.00028.

8 Hai-Tao Fang and De-Shuang Huang. Noise reduction in lidar signal based on discrete wavelet transform. *Optics Communications*, 233(1):67–76, 2004. doi:10.1016/j.optcom.2004.01.017.

9 Nicola Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 561(7722):163–167, 2018.

10 Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proc. Int'l Symp. on Code Generation and Optimization*, 2004.

11 Sung-Soo Lim, Young Hyun Bae, Gyu Tae Jang, Byung-Do Rhee, Sang Lyul Min, Chang Yun Park, Heonshik Shin, Kunsoo Park, Soo-Mook Moon, and Chong Sang Kim. An accurate worst case timing analysis for risc processors. *IEEE Transactions on Software Engineering*, 21(7):593–604, 1995. doi:10.1109/32.392980.

12 Laurent Perron and Vincent Furnon. OR-Tools. URL: https://developers.google.com/optimization/.

13 Cindy Rubio-González et al. Precimonious: Tuning assistant for floating-point precision. In *Proc. Int'l Conf. on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 27:1–27:12, November 2013. doi:10.1145/2503210.2503296.

14 Phillip Stanley-Marbell et al. Exploiting errors for efficiency: a survey from circuits to applications. *ACM Computing Surveys (CSUR)*, 53(3):1–39, 2020.

15 Tomofumi Yuki. Understanding PolyBench/C 3.2 kernels. In *International workshop on Polyhedral Compilation Techniques (IMPACT)*, 2014.