The AMPERE Project*:

A Model-driven development framework for highly Parallel and EneRgy-Efficient computation supporting multi-criteria optimization

Eduardo Quiñones, Sara Royuela
Barcelona Supercomputing Center
Barcelona, Spain
eduardo.quinones@bsc.es

Claudio Scordino, Paolo Gai
Evidence S.r.l.
Pisa, Italy
claudio@evidence.eu.com

Luis Miguel Pinho, Luis Nogueira
Instituto Superior de Engenharia do Porto
Porto, Portugal
lmp@isep.ipp.pt

Jan Rollo
Sysgo
Prague, Czech Republic
jro@sysgo.com

Tommaso Cucinotta, Alessandro Biondi
Scuola Superiore Sant'Anna
Pisa, Italy
t.cucinotta@santannapisa.it

Arne Hamann, Dirk Ziegenbein
Bosch
Stuttgart, Germany
arne.hamann@de.bosch.com

Hadi Saoud, Romain Soulat
Thales Research & Technology
Palaiseau, France
hadi.saoud@thalesgroup.com

Björn Forsberg, Luca Benini
Eidgenössische Technische Hochschule Zürich
Zurich, Switzerland
bjoernf@iis.ee.ethz.ch

Gianluca Mando, Luigi Rucher
Thales Italy
Florence, Italy
gianluca.mando@thalesgroup.com

***Abstract*—The high-performance requirements needed to implement the most advanced functionalities of current and future *Cyber-Physical Systems* (CPSs) are challenging the development processes of CPSs. On one side, CPSs rely on model-driven engineering (MDE) to satisfy the non-functional constraints and to ensure a smooth and safe integration of new features. On the other side, the use of complex parallel and heterogeneous embedded processor architectures becomes mandatory to cope with the performance requirements. In this regard, parallel programming models, such as OpenMP or CUDA, are a fundamental brick to fully exploit the performance capabilities of these architectures. However, parallel programming models are not compatible with current MDE approaches, creating a gap between the MDE used to develop CPSs and the parallel programming models supported by novel and future embedded platforms.**

The AMPERE project will bridge this gap by implementing a novel software architecture for the development of advanced CPSs. To do so, the proposed software architecture will be capable of capturing the definition of the components and communications described in the MDE framework, together with the non-functional properties, and transform it into key parallel constructs present in current parallel models, which may require extensions. These features will allow for making an efficient use of underlying parallel and heterogeneous architectures, while ensuring compliance with non-functional requirements, including those on real-time performance of the system.

***Index Terms*—parallel programming models, parallel and heterogeneous embedded processor architectures, model-driven approaches, safety-critical embedded systems**

I. Introduction

Cyber-Physical Systems (CPSs) are increasingly required to exhibit a high degree of autonomy and intelligence. This need is motivated by the societal and industrial demands appearing across multiple application domains for autonomous and safe mobility, wellbeing and health, sustainable production and smart manufacturing, etc. This trend is however challenging the development of novel CPSs, which increasingly require high-performance capabilities to coordinate a considerable number of distributed and networked computational and physical processes, while operating at small power envelopes.

These new CPSs bring an increasing need for hosting more features, which are often very complex [1], over heterogeneous computing platforms. Such features include support for different levels of criticality, complex and tightly-interacting distributed software components, and relevant energy consumption constraints. As a consequence, there is a visible trend in industry towards adopting low-power hardware architectures characterized by lower clock rates and increasingly parallel execution capabilities. These architectures are supported by a variety of technologies featuring parallel homogeneous platforms, and heterogeneous and hardware-accelerated architectures (e.g., multi-core, many-core and DSP fabrics, GPUs, and on-chip FPGAs). Examples of such platforms include the Xilinx UltraScale+ [2] and Versal [3], the MPPA Coolidge [4], and the NVIDIA Jetson AGX [5]. The introduction of parallel execution introduces two important technical challenges concerning the software architecture of CPSs:

1) The use of *model-driven engineering* (MDE) is very common in software design as it (i) allows the description of the interactions between the cyber and physical components in the system, (ii) abstracts the complexity of the software and hardware architecture, and (iii) allows for early verification and validation performed on the models. The introduction of parallel execution

* This work has been supported by the EU H2020 project AMPERE under the grant agreement no. 871669.

cannot prevent the use of MDE and should retain the benefits of simulation and verification techniques.

2) The use of parallel execution, the current dominant technique in the HPC domain to increase performance, cannot preclude the fulfilment of non-functional constraints such as energy efficiency, safety and security, resiliency and fault-tolerance, and trust and testability, which are imposed due to interactions between the cyber and physical worlds.

This paper provides a high-level overview of how these challenges are planned to be tackled in the AMPERE EU H2020 research project[1], coordinated by Barcelona Supercomputing Center and participated by Instituto Superior de Engenharia do Porto, Eidgenössische Technische Hochschule Zürich, Scuola Superiore Sant'Anna, Bosch, Evidence, Thales and Sysgo. AMPERE will address these challenges by delivering a novel software architecture for the development of CPSs to help system developers leverage low-energy, highly-parallel, and heterogeneous computations in their development process, while fulfilling the non-functional constraints inherited from the cyber-physical interaction. The novel software architecture targets safety-critical automotive and railway systems, and will be demonstrated during the project in use-cases from both domains, provided by Bosch and Thales.

The remainder of the paper is structured as follows: Section II introduces the main challenges that will be faced in the project; Section III describes the envisoned components of the projects and their expected interactions; and Section IV describes the uses-cases that will be used to test the capabilities of the project's contributed software stack.

II. Challenges of Parallel Execution for CPSs

To successfully combine parallel execution and Model Driven Engineering, two aspects are required; extended development and synthesis support, as well as runtime support to uphold the non-functional requirements at deployment. These issues are described in the following section.

A. CPSs Development: Model Driven Engineering (MDE)

The development of CPSs poses two important challenges [6]: (1) to satisfy the non-functional constraints imposed due to the interaction between the cyber and physical worlds; and (2) to ensure a smooth and safe integration of new features into existing systems. In order to address these challenges, current industrial development practices are based on MDE [7], [8]. This approach presents three main advantages, which are described next.

First, MDE allows the construction of complex software architectures, defining components with clear interfaces, thus facilitating the integration of new features through *composability* [9]. This design principle aims at ensuring that the non-functional requirements fulfilled when developing components in isolation are maintained at system integration, avoiding unpredictable and hidden behaviors. Consequently, it reduces the complexity of the development and integration phases, which is crucial in safety-critical automotive and railway systems, as the ones considered in AMPERE. This is possible because composability allows the integration of safety-critical functions and components with different levels of criticality into the same computing platform with limited certification and testability efforts. To that end, composability (also commonly referred to as *freedom-from-interference*) is regulated by the major functional safety standards. To name a relevant example, the ISO26262 [10] standard for the automotive domain mandates strong temporal and spatial isolation capabilities.

Second, MDE enables the use of code synthesis methods and tools [7], [8] in the form of automated transformation engines and code generators. These tools transform the system model description into source code, to be later compiled and executed on the computing platform [11], [12]. The benefits of synthesis[2] methods are twofold: (1) they ensure consistency between the transformed source code and the system models that have been analyzed and guaranteed as correct with respect to functional and non-functional constraints; and (2) they facilitate formal verification of the system with respect to functional and non-functional properties and constraints. This is of paramount importance for safety-critical systems such as the ones implemented in the automotive and railway domains, for which (strong) evidence about system correctness must be provided and traditional approaches based on exhaustive testing may be computationally infeasible.

Third, MDE enables the development of domain specific model-driven languages (DSML) to further facilitate the description of the cyber/physical interaction characteristics of each domain.

Overall, the use of MDE and code synthesis methods enhances the system design by (1) generating evidence of functional and non-functional correctness, and (2) facilitating the development and integration of new advanced features with reasonable effort, at least as long as composability is ensured.

Unfortunately, current MDE synthesis methods mostly transform DSML into simple and long sequences of sequential source code that is only suitable for single-core execution. This is the case of model-based solutions using commercial tools such as SCADE [11] or the AUTOSAR methodology [13]. Other tools, such as Simulink Coder [14], provide limited support for parallelism, allowing for multi-core execution of models, but without exploiting the possible parallelism inside the model subsystems. Clearly, single-core execution cannot provide the computing power required to deal with the complex functionality of future CPSs.

There is therefore a urgent necessity to create novel development environments incorporating: (1) synthesis methods capable of generating efficient source code transformations targeting parallel heterogeneous platforms, and (2) the corresponding run-time parallel frameworks, which must be capable of preserving non-functional and composability guarantees.

[1] www.ampere-euproject.eu

[2] Synthesis is often referred to as a *correct-by-construction* paradigm [8], as opposed to the conventional *construct-by-correction* paradigm [8], that may be tedious and error prone.

B. Run-time parallel frameworks

The principle behind current advanced parallel programming models (e.g., OpenMP, OpenACC and OpenCL) is that parallel computation is not fully controlled by the programmer, but by the run-time instead. The programmer is only in charge of specifying the parallel nature of the algorithm because parallel programming models provide APIs that abstract the parallel execution by defining independent execution units and fine-grained synchronization mechanisms among data elements to guarantee the correct control-flow and data-flow execution. Then, the run-time framework has the freedom to orchestrate the parallel execution among the different computing resources within the same platform [15], [16] in the most convenient way, with the objective of maximizing performance while fulfilling non-functional requirements. Moreover, in recent years, parallel programming models also support heterogeneous computing [15]–[18] by incorporating acceleration models that enable the programmer to efficiently manage the execution between the host and multiple acceleration devices, and defining synchronization methods and transfer mechanisms to offload computation and data.

Recently, these models also incorporated support for FPGA accelerators, which enable implementing hardware (HW) system components previously implemented in software, with tremendous benefits on performance speed-up and energy efficiency. Some FPGA fabrics can be dynamic and partially reconfigured to implement new HW functionalities, while other parts of the FPGA continue their operation simultaneously. Moreover, parallel programming enables instructing the run-time to decide whether a function is executed in SW within the host or in HW within the FPGA.

The run-time parallel framework is therefore a fundamental software component with three main functionalities:

1) Distributing the parallel computation among the available computing resources (from the same or different processor architecture), while respecting the defined order by means of synchronization techniques.
2) Offloading (parallel) computation and its corresponding data-set to hardware acceleration devices (e.g., GPUs, many-core fabrics, and FPGA technology) for an enhanced and energy-efficient parallel execution.
3) Dynamically reconfiguring a portion of the FPGA to use a dedicated hardware accelerator for offloading parts of the computations [19], while other portions of the FPGA are simultaneously being used by other applications or even parallel entities from the same application.

The compute resource allocation and offloading operations are mainly based on run-time information with the objective of improving the performance of parallel computation. Clearly, these run-time decisions can defeat the evidence provided at analysis-time that guarantees the fulfillment of functional and non-functional constraints. For example, (i) the run-time resource allocation, either static or dynamic, should preserve the timing constraints imposed by response-time analysis; (ii) the guaranteed energy budget should never be exceeded, hence the runtime could chose at some point to offload computation to a device based on the energy consumed by the system up to that point; or (iii) the impact of fault-tolerance and resiliency techniques devised to ensure robustness should not affect other non-functional constraints[3]. Moreover, current dynamic partial reconfiguration (DPR) FPGA controllers target general-purpose architectures, in which non-functional requirements such as time-criticality are not taken into account. This results in unpredictable timing reconfiguration, which is not suitable for CPSs [20].

Finally, parallel programming is yet a tedious and error-prone process due to the unpredictable nature of the parallel execution. There are two main sources of errors when dealing with parallel code: a) the concurrent access to shared resources in a situation of race condition, and b) an error in the synchronization between parallel operations leading to a deadlock. Identifying these two error conditions is very hard, as they may be hidden in the code for a long time and only occur under certain execution conditions. As a result, the use of parallel programming models that guarantee that no data race conditions nor deadlock conditions can happen is of paramount importance [21]–[23].

There is therefore a urgent necessity to develop run-time parallel frameworks capable of guaranteeing that decisions made at run-time maintain the guarantees about system correctness achieved at analysis time. These new run-time capabilities however, cannot preclude the ability of the CPSs to dynamically adapt the execution to new working conditions or changing modes of operation to maximize the utilization and performance capabilities of parallel heterogeneous platforms.

III. The AMPERE project

The AMPERE H2020 EU project addresses the challenges presented in Section II by developing a novel software architecture for an efficient development and execution of CPSs targeting the most advanced low-energy and highly-parallel heterogeneous embedded processor architectures. This aims at fully exploiting the benefits of performance-demanding emerging technologies, such as artificial intelligence or big data analytics.

A. Bridging the gap between MDE and Parallel Programming Models

From a system model perspective, MDE is a very convenient representation for parallel abstraction, as it naturally identifies the computation and communication elements, as well as the precedence constraints existing among the reactions or computations of components. Furthermore, MDE facilitates the verification of the system by incorporating non-functional properties and constraints such as energy (e.g., operational ranges, temperature), timing (e.g., deadline, periods), safety and security (e.g., isolation properties, assurance level), fault-tolerance (e.g., components supporting redundancy), etc. Furthermore, MDE leverages multi-criteria design space exploration techniques, for a (potentially-optimal) mapping of components

[3]The AMPERE project will not consider fail-operational requirements.

to targeted processors while fulfilling the given constraints. Despite such advantages, model-driven languages and code synthesis methods used in current system developments are still not ready to support parallel source code transformations.

From the parallel programming perspective, parallel programming models are a vital element incorporated in the SDKs of current parallel heterogeneous architectures. They provide an abstraction level to program parallel applications while hiding the platform complexities. The most advanced parallel programming models (e.g., OpenMP [15], OpenCL [17], CUDA [16], and OpenACC [18]) define an interface that abstracts the parallel execution by: (1) defining independent execution units and fine-grained synchronization mechanisms among data elements to guarantee the correct control-flow and data-flow execution; (2) defining transfer mechanisms to offload computation and data to accelerator devices, and synchronization methods among them; (3) supporting DPR SoC-FPGAs; and (4) distribute the computation among the parallel processor architectures that form the computing platforms. The run-time parallel framework is then in charge of orchestrating and distributing the parallel heterogeneous execution. Despite the efforts of simplifying the programmability of parallel computation, the programmer is still responsible of determining the most appropriate parallel units, the synchronization mechanism and the sharing attributes of the data used by the parallel units: (1) among the computing processors that form the platform, (2) within the host and (3) among the host and the accelerator devices. In general, the process of manually analysis the data-flow of a program can be very difficult due to the uncertainty about the exact moment parallel parts execute, as the run-time and not the programmer is the one in charge of managing the parallel computation with the objective of maximizing performance. Overall, current parallel programming models (and their associated run-time frameworks) are not prepared to satisfy and optimize the non-functional constraints that are critical for CPSs.

There exists therefore a gap between the DSMLs used to develop CPSs and the parallel programming models and run-time frameworks supported by current energy-efficient parallel heterogeneous platforms, as it is shown in Figure 1. AMPERE will bridge this gap by developing a novel *Software Architecture* with the following main capabilities: (1) synthesis tools capable of generating parallel source code optimized for low-energy parallel heterogeneous platforms, as well as fulfilling the functional and non-functional constraints of the system, and (2) run-time frameworks capable of respecting the guarantees devised at analysis time, while dynamically optimizing the parallel execution to changing execution conditions.

B. The AMPERE Software Architecture

The AMPERE software architecture will incorporate advanced techniques capable of capturing the component definition and communication described in the system model, together with the non-functional properties, to key parallel constructs (e.g., allocation of data and scheduling of data movements, data sharing attributes, tasking, synchronization

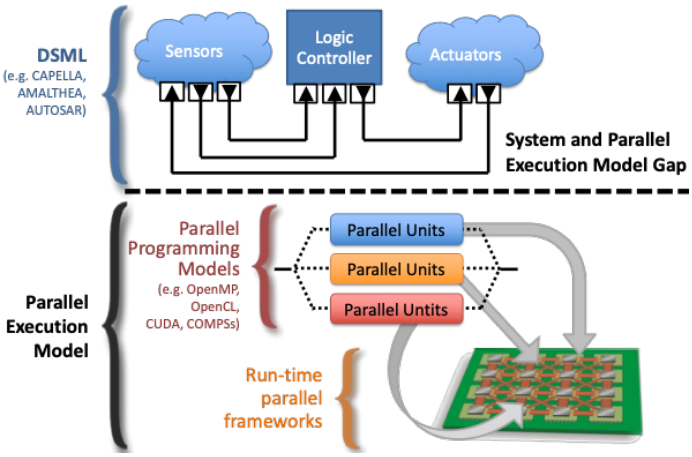

Figure 1. Execution gap existing between the MDE and the parallel programming model.

or accelerator kernels) present in current parallel programming models. This code transformation will perform a multi-criteria optimization addressing not only performance, but also energy efficiency, safety, cyber-security, real-time response, resiliency, fault-tolerance and testability non-functional requirements. Moreover, the AMPERE's run-time mechanisms will be aware of such optimization, ensuring that guarantees on functional and non-functional correctness devised at system design time will be maintained at deployment time, while fully exploiting the performance capabilities of the underlying parallel heterogeneous platforms.

AMPERE's vision is that state-of-the-art parallel programming execution models are mature enough (or can be efficiently extended) to support the fulfillment of functional and non-functional constraints captured by the DSML used to develop CPSs. Interestingly, despite current parallel programming models are agnostic of non-functional constraints, recent studies demonstrated that OpenMP allows providing trustworthy timing guarantees [24]–[28], and so fitting the timing requirements of critical systems. Moreover, it has been demonstrated that OpenMP is not a threat regarding functional safety [29], and that both compilers and runtimes can be used to check correctness and recover from failures [21]. Finally, recent releases of OpenMP and OpenCL introduce the concept of prioritization in parallel execution units, clearly opening the door to support multiple criticality levels as already supported in most of DSML, such as AUTOSAR.

Figure 2 shows the layers of the AMPERE software architecture and their relationships. Both are described next:

- *DSMLs*. It will include enhanced model-driven languages capable of better expressing and verifying non-functional constraints such as performance, energy, safety and security, time predictability, and fault- tolerance in the context of the parallel heterogeneous computing.
- *Parallel programming models*. It will prioritize well-known programming models capable of distributing the computation across different platforms and expressing structured and unstructured parallelism with fine-grain

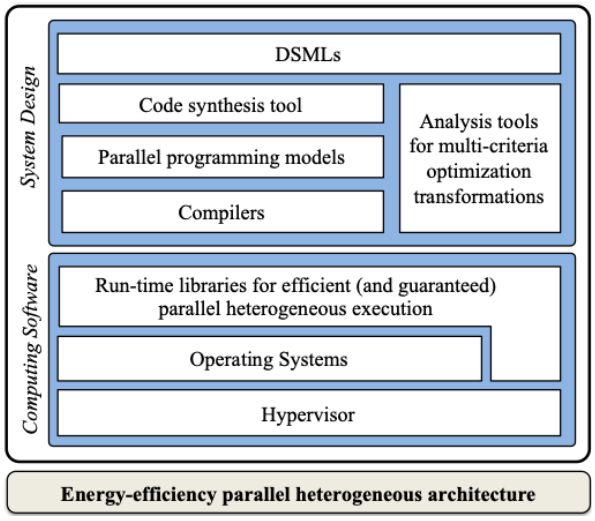

Figure 2. Software components that will form the AMPERE software architecture.

support for synchronization, while featuring acceleration devices including many-cores, GPUs, or FPGAs.

- *Code synthesis tools*. It will include novel code synthesis tools capable of transforming the DSML into an optimized parallel source code supported by the underlying parallel heterogeneous platform.
- *Analysis and testing tools*. It will include a set of powerful analysis tools, capable of guaranteeing efficient multi-criteria optimizations at development and execution phases, guiding the model-driven to programming model transformation and ensuring that functional and non-functional constraints are fulfilled.
- *Compilers*. It will include compilers capable of extracting the control-flow and data-flow information needed by the analysis tools to maximize multi-criteria optimization and correctness.
- *Run-time libraries*. It will include three different libraries in charge of: (1) orchestrating the parallel execution as defined by the parallel programming model; (2) managing heterogeneous computing, including an efficient offloading of code and data to accelerator devices; and (3) supporting an efficient computation on accelerators, including DPR SoC-FPGAs, while preserving the non-functional requirements devised at design time.
- *Operating systems and hypervisors*. It will incorporate multiple operating systems supporting a variety of architectures and accelerators, as well as a safe and secure real-time hypervisor capable of running on the target architectures and managing accesses to hardware resources. Part of the project will focus on the support for parallel real-time applications running on the Linux operating system, exploiting advanced scheduling capabilities like SCHED_DEADLINE [30] or its variants [31] in combination with on-the-fly reprogramming of FPGA slots that will be made accessible in time-shared manner synchronously with the scheduling decisions.

In order to minimize the impact on current design and development frameworks, the project will not only prioritize well-known parallel programming models, but also extend existing tools rather than developing new ones.

IV. The AMPERE use-cases

The AMPERE Software Architecture will be applied in relevant application environments by developing and executing two real-world use cases that are very close to production.

A. Automotive use-case: Intelligent Predictive Cruise Control

The AMPERE partner Bosch will provide an intelligent Predictive Cruise Control (PCC) use case, as an example for the increasingly autonomous decision-making capabilities of advanced automotive systems. This use case will consist of four components: Adaptive Cruise Control (ACC), the powertrain control subsystem, the advanced PCC, and Traffic Sign Recognition (TSR) subsystems. The first and the second components already exist, while the third and the fourth components are new.

The PCC extends the scope of the ACC functionality by calculating the vehicle's future velocity curve using the data from the electronic horizon (topographical data like curvature, inclines, or speed limits), which provides information on the route ahead, extending well beyond the next bend. In combination with the PCC, the ACC functionality now not only automatically reduces the vehicle's speed when it detects obstacles or slow-moving vehicles ahead, but also when approaching bends and reduced-speed zones. The deep learning TSR functionality (based on deep neural networks) detects road signs and provides speed limits to PCC which reflect the current state of the road (e.g. considering variable-speed zones or construction sites) that are not featured in the electronic horizon. Moreover, in cooperation with the powertrain control, the PCC improves fuel efficiency by configuring the driving strategy based on predictive data analytics and AI methods.

The computational power and communication bandwidth required for complex systems-of-systems such as the PCC exceeds the capabilities of current compute nodes (mainly micro-controller SoCs) and is leading to the paradigm of so-called centralized E/E architectures that are based on a new class of computing nodes featuring powerful micro-processors and accelerators such as FPGAs and GPUs.

B. Railway use case: Obstacle Detection and Avoidance System (ODAS)

Tramway systems are facing the challenge of autonomous urban transportation systems in a similar way as automotive industry is facing the autonomous car challenge. Interestingly, these two challenges converge in cities, as most of the time tram vehicles operate in a mixed traffic environment with cars. Autonomy applied to urban transport imposes severe real-time, reliability, and cyber-security requirements to guarantee a safety operation of the system. Moreover, it requires to make the trams smarter through digitalization, sensors data-fusion and artificial intelligence algorithms. As a result, the

use of new complex and high-demanding parallel computing platforms are requested to be installed in tram vehicles.

The AMPERE partner Thales Italy will provide a use case applied to a real demonstrator on the Florence Tramway Network. Specifically, the use case will implement an Obstacle Detection and Avoidance System (ODAS) supporting the tram driver, improving the level of safety of the Florence LRT transportation system. The use-case incorporates two main subsystems: the Sensor Data Fusion (SDF) and the AI Analytics (AI) components. The SDF component will be in charge of collecting a large mass of raw data from the multiple advanced sensors installed in tram vehicle, i.e., optical and thermal cameras, radars and LiDARs (light detection and ranging). Cameras are a very good tool for classifying objects (rails, signs vehicle, people, etc.) through deep learning technologies; LiDAR and radar are good at estimating the position of objects around the vehicle. The AI component will incorporate machine learning (e.g., SVM) and deep learning (e.g., CNN, RNN) algorithms to identify and track objects along the tramway infrastructure and extract knowledge that will be displayed to the tram driver.

The two components will be distributed and executed in a COTS parallel and heterogeneous platform installed onboard tram vehicles, featuring multi-core SoC with FPGAs, GPUs and dedicated AI accelerators such as TPUs, capable of accelerating large matrix operations and perform mixed-precision matrix multiply and accumulate calculations in a single operation.

V. Conclusions

The AMPERE H2020 project, started in January 2020, and with a duration of 36 months, is developing a new generation of software development environments for low-energy and highly parallel and heterogeneous computing architectures, capable of implementing correct-by-construction advanced CPSs. The key innovation of the AMPERE software architecture will be its capability of transforming the system model description of the CPSs based on specific model-driven languages to the parallel programming models supported by the underlying parallel architecture, and so providing the level of performance required to implement the most advanced functionalities. Moreover, the AMPERE software architecture will fulfill the non-functional requirements (i.e., real-time, safety, energy-efficiency, security, reliability) imposed due to the cyber-physical interactions and captured in the CPSs system description. The project proposals will be demonstrated through the development and execution of two real-world use cases, an intelligent Predictive Cruise Control application for advanced automotive systems and an Obstacle Detection and Avoidance System for tramway systems.

References

[1] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein, and M. Wolf, "Future automotive systems design: Research challenges and opportunities: Special session," in *CODES*. IEEE Press, 2018.

[2] Xillinx, "UltraScale+," 2020. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[3] ——, "Versal," 2020. [Online]. Available: https://www.xilinx.com/products/silicon-devices/acap/versal-prime.html

[4] Kalray, "MPPA," 2020. [Online]. Available: https://www.kalrayinc.com/portfolio/processors

[5] NVIDIA, "Jetson AGX Xavier," 2020. [Online]. Available: https://developer.nvidia.com/embedded-computing

[6] ECSEL, "Multi Annual Strategic Research and Innovation Agenda (MASRIA), 2015."

[7] e. Khaitan, "Design Techniques and Applications of Cyber-physical Systems: A Survey," in *IEEE Systems Journal*, 2015.

[8] D. C. Schmidt, "Model-Driven Engineering," in *IEEE Computer, 39(2)*, Feb 2006.

[9] G. Fernandez, J. Abella, E. Quiñones, L. Fossati, M. Zulianello, T. Vardanega, and F. J. Cazorla, "Seeking time-composable partitions of tasks for cots multicore processors," in *18th IEEE ISORC*, 2016.

[10] I. 26262, "1:2011, Road vehicles – Functional safety," 2011.

[11] E. Technologies, "SCADE suite." [Online]. Available: http://www.esterel-technologies.com/products/scade-suite/

[12] Bosch, "AMALTHEA - Model based open source development environment for automotive multi-core systems." [Online]. Available: http://amalthea-project.org

[13] A. Consortium, "AUTomotive Open System ARchitecture." [Online]. Available: http://www.autosar.org

[14] M. Works. [Online]. Available: http://www.mathworks.com/

[15] OpenMP ARB, "OpenMP Application Program Interface, version 5.0," 2018. [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

[16] NVIDIA, "Programming Guide :: CUDA Toolkit Documentation," 2019. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[17] Khronos, "OpenCL Open Computing Language): A Parallel Programming Standard for Heterogeneous Computing Systems," 2020. [Online]. Available: http://www.khronos.org/opencl

[18] OpenACC, "Application Programming Interface Version 3.0," 2020. [Online]. Available: https://www.openacc.org/specification

[19] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs," in *IEEE RTSS*, 2016.

[20] M. S. L. Pezzarossa and J. Sparso, "Reconfiguration in FPGA- based multi-core platforms for hard real-time applications," in *ReCoSoC*, 2016.

[21] Royuela, Sara and Martorell, Xavier and Quiñones, Eduardo and Pinho, Luis Miguel, "OpenMP tasking model for Ada: safety and correctness," in *Ada-Europe*, 2017.

[22] S. Royuela, L. M. Pinho, and E. Quiñones, "Converging Safety and High-performance Domains: Integrating OpenMP into Ada," in *DATE*, 2018.

[23] D. Casini, A. Biondi, and G. Buttazzo, "Analyzing Parallel Real-Time Tasks Implemented with Thread Pools," in *DAC*, 2019.

[24] Vargas, Roberto E and Royuela, Sara and Serrano, Maria A and Martorell, Xavi and Quiñones, Eduardo, "A lightweight OpenMP4 Runtime for Embedded Systems," in *ASP-DAC*, 2016.

[25] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quiñones, "Timing Characterization of OpenMP4 Tasking Model," in *CASES*, 2015, pp. 157–166.

[26] Maria A. Serrano, Alessandra Melani, Sebastian Kehr, Marko Bertogna, Eduardo Quiñones, "An Analysis of Lazy and Eager Limited Preemption Approaches under DAG-based Global Fixed Priority Scheduling," in *ISORC*, 2017.

[27] M. A. Serrano, S. Royuela, and E. Quiñones, "Towards an OpenMP Specification for Critical Real-time Systems," in *IWOMP*, 2018, pp. 143–159.

[28] Maria A. Serrano, Eduardo Quiñones, "Response-time analysis of DAG tasks supporting heterogeneous computing," in *DAC*, 2018.

[29] S. Royuela, A. Duran, M. A. Serrano, E. Quiñones, and X. Martorell, "A Functional Safety OpenMP* for Critical Real-Time Embedded Systems," in *IWOMP*. Springer, 2017, pp. 231–245.

[30] J. Lelli, G. Lipari, D. Faggioli, and T. Cucinotta, "An efficient and scalable implementation of global EDF in Linux," in *OSPERT*, Porto, Portugal, 2011.

[31] D. Casini, L. Abeni, A. Biondi, T. Cucinotta, and G. Buttazzo, "Constant Bandwidth Servers with Constrained Deadlines," in *RTNS*. New York, USA: Association for Computing Machinery, 2017, p. 68–77.