Range-Based Run-time Requirement Enforcement of Non-Functional Properties on MPSoCs

Khalil Esper, Stefan Wildermann, and Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany
khalil.esper, stefan.wildermann, juergen.teich@fau.de

***Abstract*—Embedded system applications normally come with a set of non-functional requirements defined over properties (e.g., latency), expressed as a corridor of correct values via a lower and an upper bound per requirement. These requirements should be guaranteed during each execution of an application program on a given MPSoC platform. This can be achieved using a reactive control loop, where an enforcer controls a set of properties to be enforced, e.g., by adapting the number of cores allocated to a program or by scaling the voltage/frequency mode of active processors. An enforcement strategy may react on a requirement response differently, depending on (a) satisfying a requirement, or violating (b) a lower bound or (c) an upper bound. A better strategy might be to differentiate the reaction taken according to the amount of violation of a lower or upper bound, thus to react in a finer granular way. In this paper, we propose a design space exploration (DSE) method called *Co-explore* that automatically partitions the requirement corridors into so-called *response ranges* (i.e., sub-corridors) such that formulated verification goals of simultaneously generated enforcer FSMs, e.g., the number of consecutive violations of a requirement, are optimized. The evaluation shows that the explored enforcement FSMs can achieve higher probabilities of meeting a given set of requirements compared to reacting solely based on the ternary information (a), (b), or (c).**

***Index Terms*—Runtime Requirement Enforcement, Verification, Finite State Machine, Markov Chain, Probabilistic Model Checking, PCTL, MPSoC**

I. Introduction

Embedded system applications usually come with constraints on non-functional properties. Such properties are normally exposed to uncertainty from the varying workload. *Run-time requirement enforcement (RRE)* techniques [7], [9] have been proposed to control the non-functional properties of execution of a given application program within defined bounds. Such techniques dynamically adjust control configurations like the voltage/frequency setting of resources or the number of active cores in reaction to observed changes in non-functional properties. In order to allow for a formal analysis and comparison of different control strategies, *finite state machine (FSM)* models have been proposed, e.g., [3], [4], [6]. Such approaches consider the specification of bounds on properties of discrete executions that can be described by a set of so-called *requirements* [8], i.e., expressions formulated as permissible corridors on properties of interest, e.g., latency, power consumption, and others. Intuitively, the corridor (between the user-given lower and upper bounds) splits the space of possible system responses into (a) feasible responses (inside the corridor), and responses violating either (b) the lower bound or (c) the upper bound. Enforcement strategies such as illustrated in [2], [5], change the system configuration in dependence of such responses in a reactive way. However, the above existing works are not able to distinguish the amount by which one or more requirements are violated or the closeness to a corridor boundary in taking a reaction. In this work, we show that partitioning the corridor of each requirement into so-called *response ranges* leads to better enforcer FSMs regarding their ability to satisfy a given set of verification goals.

II. Fundamentals

For each program execution on a given MPSoC, a set of non-functional requirements shall be satisfied, despite environmental changes, e.g., input variation. Such a variable input can be specified for each discrete execution k of an application by an *environment feature vector* $i(k) \in \mathcal{I}$, where $\mathcal{I}$ is called the *environment space* [3]. Assuming the number n of cores which the program utilizes can be changed as well as the voltage/frequency setting m of this set of cores, such a setting (n, m) is called a configuration c and the set of available configurations a *configuration space* C. Techniques for feedback-based RRE [3] react based on the *response* of the system-under-control by adjusting the configuration $c(k+1)$ accordingly. Let the kth execution yield H properties of interest, like latency and power consumption, depending on the input data $i(k) \in \mathcal{I}$ and the system configuration $c(k) \in C$. The response of the system-under-control can then be abstracted by a single function called *system response function* $r : \mathcal{I} \times C \to \mathbb{R}^H$ [3]. Now, requirements on these properties such as deadlines shall be satisfied for each execution k, where each property o_h, $h = 1, \ldots, H$ is assumed to be bounded by a corridor from which two propositions $\varphi_h^{LB}(o_h(k)) = (LB_h \leq o_h(k))$ and $\varphi_h^{UB}(o_h(k)) = (o_h(k) \leq UB_h)$ can be deduced, where LB_h and UB_h denote the lower and the upper bound, respectively, on the execution property o_h. The information about which proposition is fulfilled and which is violated at the kth execution can be summarized by a binary vector denoted *requirement response* $\beta := \big(\varphi^{LB}(o_1(k)), \varphi^{UB}(o_1(k)), \ldots, \varphi^{LB}(o_H(k)), \varphi^{UB}(o_H(k))\big)$ in the following.

III. Methodology *Co-Explore*

Given a sequence of inputs $\mathcal{I}_{\text{input}}$, a configuration space C, a maximum $\Xi_h^{\max}$ and a minimum $\Xi_h^{\min}$ observed system responses of a property h over all inputs and configurations, the *response space* Ξ_h of a property h is $\Xi_h = [\Xi_h^{\min}, \Xi_h^{\max}]$. Given a requirement φ_h with a lower LB_h and an upper bound UB_h, Ξ_h can be obviously divided into three intervals:

979-8-3503-4859-0/DATE24/© 2024 EDAA

$\Xi_h^L = [\Xi_h^{\min}, LB_h[$, $\Xi_h^M = [LB_h, UB_h]$, $\Xi_h^R =]UB_h, \Xi_h^{\max}]$. Based on that, Ξ_h can be partitioned into G_h (disjoint) *response ranges* $\omega_{h,g}^W$, $g \in \{1, \ldots, G_h\}$ within requirement corridors $\Xi_h^W, W \in \{L, M, R\}$, corresponding to below the lower bound, inside the user-given corridor, above the upper bound, respectively. Let G_h^L, G_h^M, and G_h^R be the number of ranges in each interval Ξ_h^L, Ξ_h^M, and Ξ_h^R, respectively, $G_h = G_h^L + G_h^M + G_h^R$. A range-based requirement response β is formulated as follows:

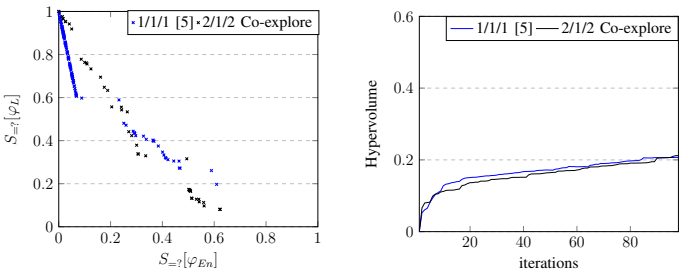
$$\beta := \left(\varphi_{1,1}^W\left(o_1(k), \omega_{1,1}^W\right), \ldots, \varphi_{H,G_H^W}^W\left(o_H(k), \omega_{H,G_H^W}^W\right)\right) \quad (1)$$

where $\varphi_{h,g}^W(o_h(k), \omega_{h,g}^W)$ is the proposition of a property o_h to be in a range $\omega_{h,g}^W$ for the kth execution.

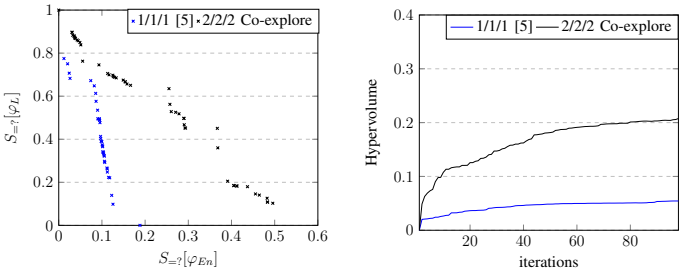
Given a system with execution properties o_h, $h = 1, \ldots, H$, a configuration space C, and numbers of response ranges G_h^W in each corridor $\Xi_h^W, W \in \{L, M, R\}$, we now propose to co-explore different response ranges $\omega_{h,g}^W$ while simultaneously optimizing the transition relation of corresponding enforcement FSMs for a given set of *verification goals* during a DSE. Each generated enforcement FSM F is able to react based on requirement responses B that are derived from the generated ranges $w_{h,g}^W$, see Equation (1).

IV. Experimental Results

We evaluated our approach using two applications called Object Detection and SHA and measured their execution properties on a tiled many-core system via simulation. We then applied the proposed Co-explore DSE that partitions the response space Ξ_h into ranges $\omega_{h,g}^W$ and generates enforcement FSMs that react on these ranges, given a set of verification goals VG for each application. For comparison, we performed another DSE without partitioning Ξ_h into response ranges according to [5]. We used the NSGA-II [1] algorithm for performing the DSEs. Each run features 100 iterations with a population size of 20 enforcement FSMs with a crossover rate of 0.9 and a mutation rate of 0.01. Each experiment was repeated three times. We refer to the number of ranges in each of Ξ_h^L, Ξ_h^M, Ξ_h^R for a property o_h by $G_h^L/G_h^M/G_h^R$. The results are shown in Figure 1. When comparing the found efficient sets (left) after 100 generations, we can see that partitioning the requirement corridors into response ranges enables the generation of enforcement FSMs with increased probabilities of meeting the shown two verification goals $S_{=?}[\varphi_{En}]$ and $S_{=?}[\varphi_L]$ which denote the steady-state probabilities of staying within a given energy, respectively latency corridor. We can also notice (right) that our Co-explore method may take more iterations to achieve a Hypervolume higher than the approach in [5] (in case of the application in (a)). This may be explained from the larger search space resulting from partitioning into response ranges.



(a) Efficient sets (left) and Hypervolume (right) of application Object Detection



(b) Efficient sets (left) and Hypervolume (right) of application SHA

Fig. 1: DSE results showing the explored efficient sets and the hypervolume over DSE iterations for the proposed technique Co-explore compared to a DSE for the generation of enforcement FSMs not supporting response ranges [5] for the verification goals $S_{=?}[\varphi_{En}]$ and $S_{=?}[\varphi_L]$ to be maximized.

Acknowledgment

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

References

[1] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.*, 6(2):182–197, 2002.

[2] Khalil Esper, Jan Spieck, Pierre-Louis Sixdenier, Stefan Wildermann, and Jürgen Teich. RAVEN: reinforcement learning for generating verifiable run-time requirement enforcers for MPSoCs. In *NG-RES 2023, Toulouse, France*, volume 108 of *OASIcs*, pages 7:1–7:16, 2023.

[3] Khalil Esper, Stefan Wildermann, and Jürgen Teich. Enforcement FSMs: specification and verification of non-functional properties of program executions on MPSoCs. In *MEMOCODE '21: 19th ACM-IEEE International Conference on Formal Methods and Models for System Design, Virtual Event, China, November 20 - 22, 2021*, pages 21–31. ACM.

[4] Khalil Esper, Stefan Wildermann, and Jürgen Teich. A Comparative Evaluation of Latency-Aware Energy Optimization Approaches in Many-Core Systems. In *NG-RES 2021, Budapest, Hungary*, volume 87 of *OASIcs*, pages 1:1–1:12, 2021.

[5] Khalil Esper, Stefan Wildermann, and Jürgen Teich. Automatic synthesis of FSMs for enforcing non-functional requirements on MPSoCs using multi-objective evolutionary algorithms. *ACM Trans. Design Autom. Electr. Syst.*, 28(6):98:1–98:20, 2023.

[6] Jan Spieck, Pierre-Louis Sixdenier, Khalil Esper, Stefan Wildermann, and Jürgen Teich. Hybrid genetic reinforcement learning for generating run-time requirement enforcers. In *21st MEMOCODE Hamburg, Germany, September 21-22, 2023*, pages 23–35. ACM / IEEE, 2023.

[7] Jürgen Teich et al. Run-Time Enforcement of Non-functional Program Properties on MPSoCs. In *A Journey of Embedded and Cyber-Physical Systems*, pages 125–149. Springer, 2021.

[8] 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 *MCSOC 2016, Lyon, Frances*, pages 313–320. IEEE Computer Society, 2016.

[9] 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 *ASP-DAC 2020, Beijing, China, January 13-16, 2020*, pages 629–636. IEEE, 2020.