Multiplexing Adaptive with Classic AUTOSAR? Adaptive Software Control to Increase Resource Utilization in Mixed-Critical Systems

Angeliki Kritikakou, Thibaut Marty, Claire Pagetti, Christine Rochange, Michaël Lauer, Matthieu Roy

Multiplexing Adaptive with Classic AUTOSAR? Adaptive Software Control to Increase Resource Utilization in Mixed-Critical Systems

A. Kritikakou, T. Marty
University of Rennes 1
IRISA/INRIA

C. Pagetti
ONERA
Toulouse

C. Rochange
IRIT
University of Toulouse

M. Lauer, M. Roy
LAAS
CNRS
Université de Toulouse

***Abstract*—Automotive embedded systems need to cope with antagonist requirements: on the one hand, the users and market pressure push car manufacturers to integrate more and more services that go far beyond the control of the car itself. On the other hand, recent standardization efforts in the safety domain has led to the development of the ISO 26262 norm that defines means and requirements to ensure the safe operation of automotive embedded systems. In particular, it led to the definition of ASIL (Automotive Safety and Integrity Levels), i.e., it formally defines several criticality levels. Handling the increased complexity of new services makes new architectures, such as multi or many-cores, appealing choices for the car industry. Yet, these architectures provide a very low level of timing predictability due to shared resources, which goes in contradiction with timing guarantees required by ISO 26262.**

For highest criticality level tasks, Worst-Case Execution Time analysis (WCET) is required to guarantee that timing constraints are respected. The WCET analyzers consider the worst-case scenario: whenever a critical task accesses a shared resource in a multi/many-core platform, a WCET analyzer considers that all cores use the same resource concurrently.

To improve the system performance, we proposed in a earlier work an approach where a critical task can be run in parallel with less critical tasks, as long as the real-time constraints are met. When no further interferences can be tolerated, the proposed run-time control suspends the low critical tasks until the termination of the critical task. In an automotive context, the approach can be translated as a highly critical partition, namely a classic AUTOSAR one, that runs on one dedicated core, with several cores running less critical Adaptive AUTOSAR application(s). We briefly describe the design of our proven-correct approach. Our strategy is based on a graph grammar to formally model the critical task as a set of control flow graphs on which a safe partial WCET analysis is applied and used at run-time to control the safe execution of the critical task.

***Keywords*—Dynamic management, Real-time systems, Run-time monitoring, Multi/Many-core systems, Safety, Resources Utilization.**

I. Introduction

The chip market moved to multi/many-core systems due to increased system requirements and power dissipation issues of single-core systems [1]. As these systems offer massive computing power, a higher integration of applications is performed in the same platform. The integrated applications have diverse characteristics which create *mixed-critical systems* [2]. A *mixed-critical system* consists of applications with different levels of criticality, as described in the ISO 26262 norm, and envisioned in the future Adaptive AUTOSAR platform. The criticality level of an application depends on the potential consequences on the system in case the application fails to meet its timing constraints. The Automotive Safety Integrity Level (ASIL) model [3] defines criticality levels depending on the risk and the potential loss due to a failure of the system; the highest level, D, corresponds to the highest criticality level, and 4 other levels have been defined —C, B, A, and QM, where QM corresponds to a level where no particular safety measure should be taken.

Applications with high criticality level require strict guarantees on their correct execution. To ensure these guarantees, real-time task scheduling techniques should use a safe estimation of the Worst-Case Execution Time (WCET) [4]. Several WCET estimation techniques exist, but static analysis tools such as AIT or OTAWA, are recommended for high criticality applications. Unfortunately, many/multi-core systems have a dynamic difficult-to-predict behavior. More precisely, the concurrent accesses to the shared resources introduce timing variations, e.g. in the communication network and in the memory hierarchy with variable delays under concurrent requests. Therefore, the effects of possible task interferences have to be upper bounded to guarantee real-time response, usually by assuming full contention under concurrent requests. The result is a safe but pessimistic WCET, in the sense that the cases leading to the worst-case scenarios are unlikely to occur. This leads to over-allocating resources to high criticality applications and may even be the cause of the system unschedulability.

A. Motivation

Let us consider $n+1$ independent synchronous partitions $\mathcal{T} = \{\tau_{\mathrm{C}}, \tau_1, \ldots, \tau_n\}$ where τ_{C} is a classic AUTOSAR partition of high criticality level (ASIL D), period T_C and deadline D_C; τ_i are partitions of lower criticality level (ASIL levels $A-C$ or QM) implemented as Adaptive AUTOSAR partitions. Each partition is executed on a different core. Hence, the classic AUTOSAR partition runs on a dedicated core, but may suffer from interference with other partitions.

Two scenarios for the WCET computation of τ_C are considered: 1) maximum load (max): all partitions run in parallel, and 2) isolation (iso): the classic AUTOSAR partition τ_C runs alone on the system. In max scenario, we assume that due to

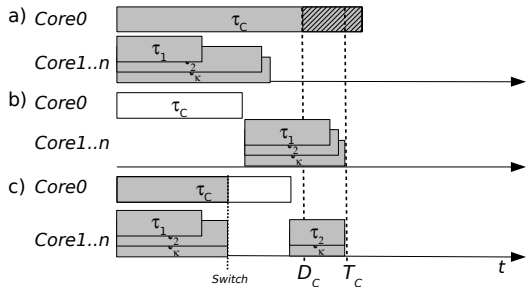

Fig. 1. Mixed-critical schedules scenarios

the resource sharing, the WCET of τ_C is above the deadline, i.e. $\mathsf{WCET}_{\mathsf{max}} > D_C$, as depicted in Fig. 1(a). In this case, the hard real-time constraints cannot be met. Existing mixed-critical scheduling approaches, such as [5], [6], [7], assume that the task set is schedulable at least in the highest criticality level, and thus are not directly applicable. Then, a safe solution is to execute τ_C in isolation. When the classic AUTOSAR partition terminates and if time slack exists, the Adaptive AUTOSAR partitions are executed. In this case, no conflict occurs with the low criticality tasks. Hence, the WCET is significantly lower and the classic AUTOSAR partition respects its deadline, i.e. $\mathsf{WCET}_{\mathsf{iso}} \leq D_C$, as shown in Fig. 1(b).

Our goal is to increase the task parallelism and to reduce the over-provisioning of resources by combining the benefits and discarding the drawbacks of the previous cases. To achieve that, all AUTOSAR partitions are started and are run in parallel, as long as it is safe. At run-time, if the interferences may lead to a deadline miss of τ_C, the lower criticality partitions are suspended until the termination of τ_C. If time slack exists, the low criticality partitions are resumed. In this way, the critical classic AUTOSAR is guaranteed to meet its deadline, whereas the low criticality partitions run in parallel improving the resources utilization, as shown in Fig. 1(c).

B. Proposed methodology and contributions

This optimistic mixed-critical schedule can be achieved using an appropriate run-time control mechanism, as proposed by our methodology. We introduce a set of *observation points* to enable the run-time (online) monitoring of the timing behavior of the critical partition and the control of partition set scenarios. At each observation point, a *safety condition* is applied to check whether it is still safe to continue the execution of τ_C in the maximum load scenario. The safety condition uses the remaining WCET of the critical partition in isolation scenario, which is run-time computed by our low-overhead algorithm. If the safety condition evaluates that a risk exists of overloading the system and, thus, the critical partition runs too slow, a backup process is applied to guarantee the real-time response of τ_C: the low criticality partitions are suspended and τ_C runs in isolation. When the critical classic AUTOSAR partition finishes its execution and if time remains until the next release of τ_C, the low criticality Adaptive AUTOSAR partitions are resumed. We proved the correctness of the proposed safety condition and the low-overhead run-time algorithm for computing the remaining WCET.

As the computation of the remaining WCET is time consuming, it cannot be performed at run-time. Hence, we proposed a run-time algorithm based on pre-computed data which reflect the program structure (static analysis). Then, the run-time computation involves only basic arithmetic and reflects the actual progress of the critical partition execution. To achieve this goal, during the design-time (offline) analysis, the functions that compose the classic AUTOSAR partition are represented by a set of Extended Control Flow Graphs (ECFGs) with observation points. We propose a graph grammar to formally describe the set of ECFGs under study and to prove the correctness of our run-time computation algorithm. Based on the obtained ECFGs, a safe WCET analysis is applied for the pre-computation of several partial remaining WCETs used by the run-time control.

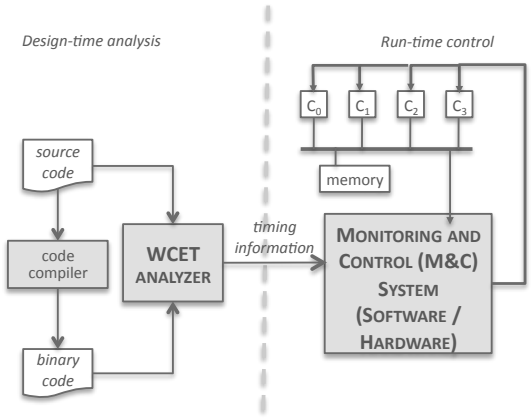

Fig. 2. Overall methodology

The proposed methodology consists of the design-time analysis to pre-compute the required data and the scenario control applied at run-time, as shown in Fig. 2. A full description of the ideas sketched in this note are detailed in [8], [9], [10].

II. Design-time analysis

A. Maximum load and isolation

This section presents the design-time analysis of our approach by describing the instrumentation of the Classic AUTOSAR partition, the static schedule and the design-time analyzer for the computation of the timing and structure information.

The proposed methodology considers two scenarios for the partitions that are executed on the platform.

Definition 1 (Execution scenarios): The execution scenarios are

1) Isolated execution (iso), where the classic AUTOSAR partition only is executed on the platform,
2) Maximum load (max), where all partitions are executed concurrently on the platform.

The proposed methodology initially executes the maximum load scenario and at each observation point of τ_C checks whether the low criticality partitions should be suspended. Hence, the following statement should be proved: "The switching from the maximum load scenario to the isolation scenario guarantees that the classic AUTOSAR partition meets the hard real-time deadline D_C."

B. Switching to isolation mode

The switching occurs when the following safety condition holds:

$$\mathsf{RWCET}_{\mathsf{iso}}(x) + \mathsf{W}_{\mathsf{max}} + t_{\mathrm{RT}} + \mathsf{ET}(x) > D_{\mathrm{C}} \quad (1)$$

where $\mathsf{RWCET}_{\mathsf{iso}}(x)$ is the remaining WCET of τ_C at an observation point x in the isolation scenario, $\mathsf{W}_{\mathsf{max}}$ is the WCET until the next observation point, t_{RT} is the total time of the proposed run-time control mechanism and $\mathsf{ET}(x)$ is the real execution time of τ_C until point x. The t_{RT} is the sum of: 1) t_{Mon} (the overhead to monitor the real execution time), 2) t_{Cnt} (the WCET of the run-time control), and 3) t_{SW} (the WCET overhead due to scenario switching).

Theorem 1: If $\mathsf{WCET}_{\mathsf{iso}} \leq D_C$, then for any execution with the proposed run-time control, τ_C always respects its deadline.

The proof of this theorem can be found in [9]. Intuitively, at each observation point, the run-time control computes the worst-case remaining time for the *next* observation point: if there is a risk that the next observation is reached too late to switch to a safe mode, then the system switches now to isolated mode, guaranteeing a correct timing behavior.

C. Extended Control Flow Graph Representation

A graph grammar is proposed to model the critical partition τ_C considered under study. τ_C is described by the general expressions of syntax in Table I, which covers a wide range of applications. From the binary code of τ_C [11], we create a set of *control flow graphs* (CFGs), where we insert observation points. The CFGs obtained from the abstract syntax of Table I and compiled without optimizations are covered by the proposed grammar.

TABLE I. APPLICATION MODEL SYNTAX

Syntax rules		
term	::=	<constant> | <variable>
expr	::=	<term> | <term> <operator> <term> | <unary-expr>
unary-expr	::=	<variable> <unary-operator> | <unary-operator> <variable>
cond-expr	::=	<expr> <conditional-operator> <expr>
assignment	::=	<unary-expr> <assignment-operator> <expr>
instruction	::=	<assignment> | <unary-expr> | <>;
stat	::=	<instruction> | <stat>;<stat> | if (<cond-expr>) then <stat1> else <stat2> | for (expr1; cond-expr; expr2) <stat> | <function-call> ;
function-call	::=	<return-type> functionName(<parameter-list>) <stat> return <expr-return>;
program	::=	<function-call>

Definition 2 (Critical partition τ_C): A *critical partition* τ_C is a set of functions $\mathcal{S} = \{F_0, F_1, ..., F_n\}$, with F_0 the main function. Each *function* is represented by an Extended CFG (ECFG).

Definition 3 (Observation point): An observation point is a check point in an ECFG where the run-time control is executed. A special observation point named *start* is defined before the starting of the execution.

The full description of the graph grammar can be found in [9], and a brief schematics of the grammar is shown in Fig. 3 and Fig. 4. A program is a set of functions (Fig. 3), and every function is described by the grammar of Fig. 4, where

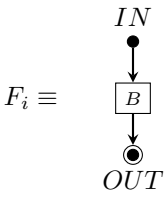

Fig. 3. Representation of function F_i.

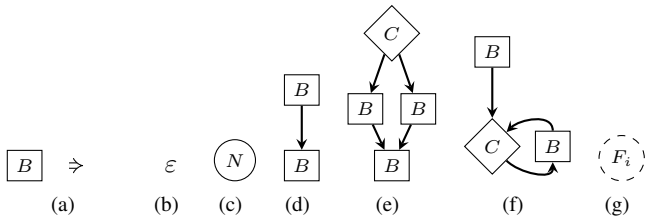

Fig. 4. Schematic representation of rewritting grammar rules

C is a conditional, N a binary instruction, F_i a function, and B a non-terminal.

D. Remaining WCET analysis

The remaining WCET of the critical partition heavily depends on the real execution. Hence, at design-time, we process the ECFGs and compute partial WCETs using safe static WCET analysis, but extended and adapted to our methodology. The WCET is computed by writing an Integer Linear Programming (ILP) formulation to express the program execution time as the combination of the individual times of the grammar components weighted by their execution counts. This expression is maximized to find the WCET, with a number of constraints that reflect flow facts, e.g. loop bounds and unfeasible paths.

Our WCET analysis is based on computing the remaining WCET from one observation point x until the end of the program, $\mathsf{RWCET}_{\mathsf{y}}(x)$, where $y \in \{\mathsf{iso},\mathsf{max}\}$. When the $\mathsf{RWCET}_{\mathsf{max}}(x)$ is computed, we consider that interferences occur from the parallel partitions, whereas when the $\mathsf{RWCET}_{\mathsf{iso}}(x)$ is computed no interferences are taken into account. When point x is the entry of the critical partition, i.e. *start* of F_0, $\mathsf{RWCET}_{\mathsf{y}}(\mathit{start})$ is the total $\mathsf{WCET}_{\tau_{i,y}}$. When point x is inside the ECFG, we compute the remaining WCET by using constraints to prohibit the execution of all the blocks which do not belong to any path from point x until the end of the ECFG.

Using the remaining WCET analysis of an observation point x, we can compute remaining WCETs between an observation point and its head point.

III. RUN-TIME CONTROL

At each observation point x, the safety condition described above decides whether switching between scenarios is required. As the $\mathsf{RWCET}_{\mathsf{iso}}(x)$ is modified at each observation point, we propose a low-overhead algorithm to run-time compute this value by efficiently reusing the $\mathsf{RWCET}_{\mathsf{iso}}$ of the head points. In [9], we describe and formally prove our algorithm when the critical partition consists of a finite set of functions,

i.e. $\mathcal{S} = \{F_0, \ldots, F_n\}$. We describe below a simplified version of the general algorithm.

The run-time computation of $\mathsf{RWCET}_{\mathsf{iso}}(x)$ is depicted in Alg. 1. An important point in this algorithm is the notion of level, that corresponds to the nesting level in a loop nest — indeed, such loop nests, as well as function calls, allow us to have a very compact representation of remaining worst-case execution times.

Pre-computed data is stored in the memory as constant arrays: *level* for the nesting level, and d and w for the partial WCETs: d is the WCET from a loop head to an observation point, and w is the WCET between two consecutive instances of a given loop. The algorithm maintains two local values o_level (for the previous observed level) and $level(x)$ for the local level. At run-time, the algorithm stores in array $last\_point$ the last observed point and in array R the computed $\mathsf{RWCET}_{\mathsf{iso}}$ per level.

ALGORITHM 1: Simplified run-time control algorithm, without function calls.

Pre-computed data: level, w, d
Input: x
Data: $o\_level = 0$, last_point[0]=*start*, $RWCET_{iso}[0]$=$WCET_{iso}$
Output: $RWCET_{iso}(x) = R[level[x]]$
if $o\_level < level[x]$ **then** /* *condition 1* */
  $R[level[x]] = R[level[x] - 1] - d[x]$
else
  if *(last_point[level[x]] == x)* **then** /* *condition 2* */
    $R[level[x]] = R[level[x]] - w[x]$
  else
    $R[level[x]] = R[level[x] - 1] - d[x]$
  end
end
last_point[$level[x]$]= x
o_level=$level[x]$

The general algorithm [9] shares many ideas with the above presented one, with the added complexity of function calls, that can be seen, in some sense, as a dynamic version of (static) loop nests.

IV. Conclusion

In this work, we presented a methodology to run several partitions of different criticality levels on a single multi or many-core chip, while guaranteeing the real-time response of a critical classic AUTOSAR partition. At design-time analysis, the critical partition is described by a set of ECFGs and partial WCET analysis is applied to compute the required data for the run-time part. At run-time, a low-overhead controller computes the remaining WCET of the critical partition and decides the switching between maximum load scenario and isolation scenario.

We advocate that such a scheme permits a better utilization of system resources, but further work has to carried out to refine this approach. For now, only two possible scheduling scenarios are supported: total isolation for the critical classic AUTOSAR partition, or full parallelism. One cannot assume that all non-classic AUTOSAR applications can be stopped at any time without any consequence on the system. Thus, supported several (statically defined) scheduling strategies, and switching between different strategies depending on run-time evaluation of remaining resources seems a necessary extension to this work in order to be included into real automotive architectures.

More precisely, our method is applicable to several high criticality partitions running on one core. In case the criticality partitions are executed on several cores, the partial RWCETs in the isolation scenario will be computed by considering that more cores are concurrently running etc. This will increase the partial RWCET, but a trade-off exists to explore between the number of active cores and the pessimism of the remaining WCETs. A similar approach could be followed in case in isolation scenario we allow some low criticality partitions to run on other cores during isolation. When the number of criticality levels is increased, we could also explore several cases regarding the parallel execution of partitions during isolation scenario. In addition, we believe that the combination of our approach with time and partitioning methods will lead to further increase in the task parallelization during isolation scenario. In addition, we are developing a methodology to decide the position of the observation points over the ECFGs in order to further reduce the monitoring overhead.

References

[1] A. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: Survey of current and emerging trends," in *DAC*, pp. 1–10, 2013.

[2] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, pp. 239–243, 2007.

[3] "ISO26262 Road Vehicles - Functional Safety." http://www.iso.org/iso/home/news%20index/news%20archive/news.htm?refid=Ref1499.

[4] M. Gatti, "Development and certification of avionics platforms on multi-core processors," in *Tutorial Mixed-Criticality Systems: Design and Certification Challenges, ESWeek*, 2013.

[5] J. H. Anderson, S. K. Baruah, and B. B. Brandenburg, "Multicore operating-system support for mixed criticality," in *WMC*, April 2009.

[6] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems.," in *CIT*, pp. 1864–1871, 2010.

[7] H. Li and S. Baruah, "Global mixed-criticality scheduling on multiprocessors," in *ECRTS*, pp. 166–175, 2012.

[8] A. Kritikakou, O. Baldellon, C. Pagetti, C. Rochange, M. Roy, and F. Vargas, "Monitoring on-line timing information to support mixed-critical workloads," in *WiP PRTSS*, 2013.

[9] A. Kritikakou, O. Baldellon, C. Pagetti, C. Rochange, and M. Roy, "Run-time control to increase task parallelism in mixed-critical systems," in *ECRTS*, 2014.

[10] A. Kritikakou, C. Rochange, M. Faugère, C. Pagetti, M. Roy, S. Girbal, and D. G. Pérez, "Distributed run-time wcet controller for concurrent critical tasks in mixed-critical systems," in *RTNS*, pp. 139:139–139:148, 2014.

[11] K. D. Cooper, T. J. Harvey, and T. Waterman, "Building a control-flow graph from scheduled assembly code," Tech. Rep. TR02-399, Rice University, 2002.