Efficient Parallelization of Complex Automotive Systems

Julian Kienberger
Department of Computer Science,
University of Augsburg, Germany
kienberger@ds-lab.org

Christian Saad
Department of Computer Science,
University of Augsburg, Germany
saad@ds-lab.org

Stefan Kuntz
Continental Automotive GmbH,
Regensburg, Germany
stefan.kuntz@continental-corporation.com

Bernhard Bauer
Department of Computer Science, University of Augsburg, Germany
bauer@ds-lab.org

Abstract

As the automotive industry seeks to include more and more features in its vehicles while simultaneously attempting to reduce the number of "Electronic Control Units" (ECUs) that execute the corresponding embedded software, the necessary policy shift towards multi-core technology is in full swing. In order to eventually exploit the extra processing power, there is much additional effort needed for coping with the tremendously increased complexity of such systems. This is largely due to the elaborate parallelization process (partitioning, mapping and scheduling software parts as tasks on different cores) that results in a combinatorial explosion and thus spans a vast search space. Mastering this challenge requires innovative methods and appropriate tools that are specifically designed for the creation of embedded multi-core applications or the migration of legacy software [16].

On the basis of the concept presented in [25], we use the results of its data dependency analysis performed on an "AUTOSAR"[1] model (AUTOSAR system descriptions) to determine advantageous partitions as well as initial task-to-core mappings. Afterwards, the extracted information serves as input for the simulation within an embedded multi-core timing tool suite. Here, the initial solution is evaluated with respect to the fulfillment of basic timing requirements and metrics like cross-core communication rates, average latencies or core workloads. A subsequent optimization process improves the initial solution and enables a comparative assessment. In order to demonstrate the benefit of this approach, we apply it to two models – a fictional mid-sized and a real-life complex one – and show the advantage compared to a parallelization process without preceding dependency analysis and initial partition/mapping suggestions.

Keywords Multi-core, AUTOSAR, Migration, Data Dependency Analysis, Timing Validation, Semi-Automated Parallelization, Systematic Partitioning, Deployment Simulation, Mapping Optimization

1. Motivation and Core Issues

In the automotive sector, car manufacturers constantly aspire to improve driving dynamics, include additional infotainment features, raise traveling comfort as well as enhance the safety and security properties of their vehicles.

However, adding further functionality – accompanied by car domains becoming more and more dependent on each other – increases complexity as well as required processing performance [11, 37]. Furthermore, there is a prevalent endeavor to save space and reduce weight by decreasing the number of ECUs, which can be achieved by replacing them with distinctly less (but more powerful) "domain controllers" [20, 27]. These intentions drive the pursuit of finding a possibility for boosting the available computing performance in order to stay competitive.

In [28], it is assumed that ten times as much processing power as currently available will be needed in only 10 years. Unfortunately, the rising demand has exceeded the capabilities of single-core technology whose processing power is almost completely exhausted and does not significantly increase anymore [17, 25, 42]. Embedded architectures that feature multiple cores (or in general "independent execution units" [7]) are – according to the current state of research – the only solution to satisfy upcoming requirements. It is therefore hardly surprising that they are becoming increasingly important [48].

In the area of desktop computing, the transition to multi-core platforms started about ten years ago, whereas the automotive sector was rather recently forced to start migrating its ECU software in order to pave the way for further technical advancement, because – based on experience – automotive microcontrollers follow "common" IT systems with a delay of roughly 5 to 8 years due to the typical development cycle times of cars [20].

As current automotive software (operating systems as well as applications) was usually neither designed for being executed in parallel on the "function level" (as addressed in this paper) nor on "application level", its proper migration to multi-core systems is a challenging task [15, 18, 28]. It involves a paradigm change,

[1] The "AUTomotive Open System ARchitecture" standardizes "[...] an open software architecture for automotive electronic control units (ECUs)" [5], cf. `http://www.autosar.org`.

PMAM 2016 – The Seventh International Workshop on Programming Models and Applications for Multicores and Manycores, March 12–16, 2016, Barcelona, Spain.
http://dx.doi.org/10.1145/2883404.2883421

because aspects like "expensive" cross-core communication, synchronization overheads, shared resources, significance of memory location and the complex scheduling of software parts come into play when processing is distributed and sequential data consistency has to be guaranteed [27, 36, 38].

In order to achieve the latter without producing unnecessary interference (i.e. overhead) among cores, it is crucial to appropriately determine the software's fragments in the first place ("partitioning") and to purposefully distribute them on the cores afterwards ("mapping"). Moreover, coordinating multiple cores to execute parts of a common application is tremendously augmenting the complexity of software due to dependencies between separately processed but still complected data including problems like race conditions, dead locks, non-determinism and insufficient load balancing (seeking equal workloads for each core) [32, 33]. The complexity rise correlates with the amount of software parts, because the number of possibilities to distribute them on cores grows exponentially, which makes an exhaustive design space exploration infeasible.

When looking at a typical combustion engine management system including up to 8000 "Runnable Entities" (REs, AUTOSAR's atomic executable and schedulable units), it is obvious that multi-core approaches massively increase the internal complexity of ECUs and finding a good partition within such highly interconnected software is costly [10, 15]. This calls for new concepts to overcome emerging challenges, more specifically, finding suitable leverage points and heuristics for the process of migrating to parallelized versions of existing application software as well as supporting this process with tools that, on the one hand, automatize as much work as possible and, on the other hand, illustrate detected problems to the engineer in a meaningful way [13, 15, 27, 28, 30, 39, 40].

Apart from these new requirements emerging in the course of the "multi-core era", a well-known aspect remains crucial: The heavy usage of models and their active inclusion (i.e. not only as supplement) in the whole process of software development and deployment is indispensable to keep an overview [10, 14]. This becomes clear when taking a closer look at the work process in the automotive sector, which is typically to a high degree based on the division of labor: For example, there are usually several hundred engineers working on an engine management system since its multitude of interconnections with other ECUs makes it exceedingly complex. Efficient collaborative working on such a system is hardly possible without proper organizational structures and suitable work items, like models.

In the following, we address the stated challenges by showing how our data dependency analysis approach (cf. [25]) can be carried forward by specifically covering the process of partitioning, mapping and scheduling.

2. Approach Overview

In order to mitigate the complexity of parallelized systems, we endeavor to effectively support the goal-oriented migration of legacy ECU software to a suitable multi-core architecture. We focus on parallelizing a single application addressing "function parallelism" (also known as "task parallelism"). The proposed migration process is depicted in Figure 1.

Both activities in the upper half are supported by our tool, which is implemented as plug-in based on the "Eclipse Modeling Framework" and the "Model Analysis Framework" within the "AUTOSAR Tool Platform"[2] (Artop) [12, 34, 35]. The lower three activities can be carried out within several third-party tools that provide simulation and optimization features for embedded multi-core systems.

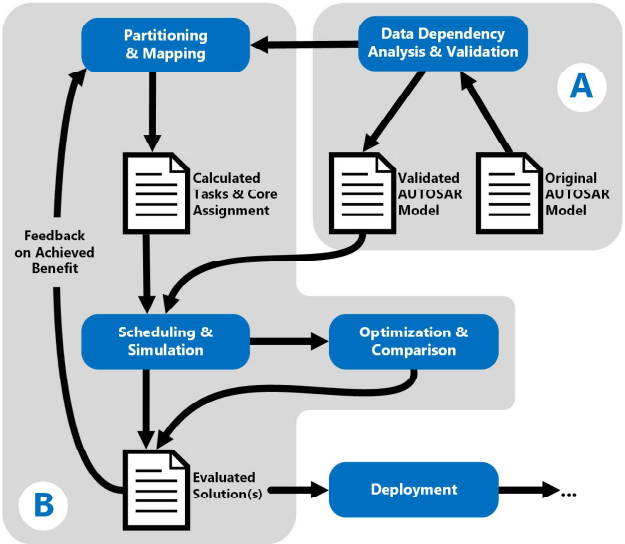

Figure 1. Overview of the migration process with previously covered research area "A" (cf. [25]) and currently addressed area "B".

Data Dependency Analysis & Validation: The basis of the approach is a data dependency analysis run on AUTOSAR models in order to detect, visualize and solve potential conflicts related to a software's distributed execution on multiple cores (corresponds to area "A" in Figure 1):

1. The analysis identifies a model's structural elements such as the aforementioned REs, their variable accesses, their recurrence (also called "triggering frequency" or "period"), the software components (SWCs) containing the REs as well as already imposed timing constraints.
2. The gathered information is assessed and potential conflicts regarding data consistency are determined.
3. Inconsistencies are addressed by an incremental (stepwise) application of timing constraints on the lowest level (i.e. on REs) for the purpose of achieving "multi-core robustness" in terms of data consistency.

This semi-automatic process provides the software with a consistent timing model to ensure the preservation of its original sequential (single-core) behavior on a multi-core platform.

Partitioning & Mapping: This step was broached in [25], where we mainly addressed the basics of the underlying partitioning algorithm, which was – at this time – not yet expedient when applied to highly complex models and was lacking a mapping feature.

The fundamental idea is to search on AUTOSAR's most-fine grained level (REs) for regions (sets of REs) with a relatively low coupling and to group them into tasks as means for creating a suitable partition as well as the subsequent "task-to-core" mapping. The immense search space can be remarkably reduced by providing a beneficial starting point for the simulation and optimization that are carried out to evaluate the initial solution and to search for further ones. This approach is based on techniques introduced in [24, 31, 47], which were further developed in [19].

[2] Artop facilitates the construction of AUTOSAR tools by serving as Eclipse infrastructure and virtually acting as "persistence layer" that enables common base functionality like easy access on AUTOSAR models that adhere to specific meta-model versions [2].

Since its initial proposition, the algorithm was extended to cope with highly complex models:

- A configurable search tolerance can be used to loosen the rather strict criteria for low-coupled regions, so that RE sets, which violate the demands to a certain tolerable extent, are not discarded.
- The relevance of the connection between two REs is calculated in order to determine concrete pair-wise dependency weights depending on the number of variable accesses between two REs as well as their respective recurrence.
- Based on experience, analyzing a whole model often unnecessarily increases the search effort while simultaneously shrinking the result set. This effect can be attributed to the high degree of interconnection in large models which exacerbates partitioning attempts. Thus, different splitting strategies are provided, e.g., dividing the model into parts with uniform recurrences before running searches on each of them.
- As these parts often remain highly complex, the search gradually ignores pair-wise dependencies below a growing threshold until a sufficient number of regions is found. This "relevance partitioning" roughly corresponds to the "multi-level partitioning" approach of first "coarsening", then clustering and afterwards restoring a graph [6].

The quality of a calculated partition and task-to-core mapping cannot be universally measured because of contradictory goals for parallelization efforts, e.g., pooling strongly-connected software parts (low synchronization overhead), distinctly separating functionalities that should not interfere (support safety) or evenly spreading computational cost (proper load balancing).

As demonstrated in the Case Study (Section 4), achieving a certain goal is possible by adjusting the preferences when arranging the tasks and mapping them to cores, e.g., the preferred granularity of the regions, a concrete load balancing strategy or enforcing certain regions to be assigned to the same core (via "pairing constraints"). In our case, we are seeking a pooling-oriented partition together with a mapping that enables satisfactory load balancing.

Of course, there are numerous other factors worth taking into consideration, e.g., if the target platform's processor is "homogeneous" concerning equally equipped cores and how many of them are available.

In the end, this step's purpose is to provide an advantageous starting point ("initial solution"), so that the following simulation and optimization are supported to find appropriate further solutions in an adequate amount of time.

Scheduling & Simulation: The subsequent step employs a third-party simulation and optimization tool in order to evaluate the usefulness of the provided solution. There are several software suits available that are designed to analyze, simulate, validate and optimize embedded multi-core systems (cf. Subsection 3.3). Regardless of what product is employed, some preparations have to be made before such a simulation can deliver expressive results:

- Importing an AUTOSAR model together with the calculated partition and mapping from the previous working step
- Adding basic timing requirements (e.g. task deadlines) to later allow a general classification into valid and invalid solutions
- Setting up the underlying hardware model, e.g., a generic processor or a specific automotive micro-controller
- Choosing an appropriate scheduling algorithm

Running a timing simulation on highly complex models can take a lot of time, but eventually yields informative findings about a solution's general validity, occurring latencies, overhead caused by necessary synchronization or a basic statements about a software's overall degree of potential parallelization (indicating possible speed-up). Generally speaking, it can be stated that a "good" preceding partitioning & mapping facilitates the scheduling step considerably which is particularly important when being confronted with cross-core communication as a substantial new resource bottleneck [27].

Optimization & Comparison: Having gained a first glimpse on the initial solution's quality enables a comparison with potential alternatives. The latter's creation is facilitated by taking the initial solution as basis to derive modified versions of it. In general, there are plenty possibilities to do so either manually or with tooling support. In many cases, the above-mentioned third-party simulation tools also provide optimization features.

A focus on "weak spots" discovered by means of interpreting the simulation results can act as beneficial leverage point for model changes, e.g., splitting up "chunky" tasks that hamper proper load balancing or enforcing two heavily communicating tasks to be mapped to the same core. Having found better solutions in terms of certain characteristics – like reduced cross-core communication or a shorter overall cycle time – allows to "close" the round-trip engineering circle by providing customized data on advantageous model-specific search parameters for the "Partitioning & Mapping" step. This iterative proceeding seems sensible in order to effectively broaden the search span by means of a fresh "solution seed".

3. Approach in Detail

In the following, we describe the introduced migration process (cf. Figure 1) in detail and start by explaining how models are prepared for distributed execution on several cores.

3.1 Data Dependency Analysis & Validation

As already broached, the key of this step is to find a consistent timing model for an embedded application, so that its original functionality is preserved. In this context, conflicts can occur when functional blocks – originating from legacy (single-core) software – are processed in parallel instead of well-matched and rigidly consecutive like before. This poses a threat to consistency, which is – in our case – defined as stability (steady signals/values over a certain period of time) being paralleled by coherency (signals/values with uniform data age).

Possible conflicts are, e.g., data not being available in time or data being read inconsistently. Of course, consistency conflicts like this can occur in single-core platforms too, but multi-core systems are more prone to "evoke" them because here it is – due to concurrency – significantly harder to maintain consistency.

Therefore, it is inevitable to take care of any potentially unintentional behavior within a system. The challenge consists of the already addressed complexity rise caused by the exponentially growing number of possibilities to distribute tasks on cores, which leads – together with scheduling – to a tremendous count of possible execution sequences including many adverse, i.e. conflict-ridden, ones. In order to ensure that a system will work properly irrespective of a certain task-to-core mapping, it is required to distinctly exclude all unwanted behavior which we achieve by enriching the model with timing constraints (or modifying existing ones).

As implied in Section 2, our approach as a whole is based on the following principles:

- Bottom-up: Analysis and validation take place on the most "fine-grained" level (i.e. REs in AUTOSAR), because a top-down approach is hardly applicable here as ECU software is rather constantly refined and extended than newly created (due to competitive pressure). In addition, abstracted views on the

system can still be created by interpreting gathered low-level information.

- Stepwise: Validation is done stepwise and not all at once for the purpose of preventing – on the one hand – the generation of "new" conflicts by overlapping constraint scopes and – on the other hand – indiscriminately calculating all possibilities at the beginning which would take more time and would require disproportionate computing power.
- Minimal: Only a minimum set of constraints is imposed in order to not unnecessarily restrict the degree of freedom for succeeding steps (and therefore not to unintentionally exclude promising solutions).

Sticking to this concept, we perform a static data dependency analysis directly on AUTOSAR models (cf. [25] for more details) The basic steps are:

1. Parse the model and analyze the data dependencies ("connections") between the Executable Entities (i.e. instances of REs) contained in the SWCs and possible execution sequences of them based on already imposed EOCs.
2. Classify the detected dependencies and filter out possibly conflicting variable accesses among them.
3. Impose timing constraints to restrict all possible execution sequences to a set that provides every RE instance with its required input data in time.

First of all, parsing the AUTOSAR model provides the information basis needed for further steps: AUTOSAR's SWCs are the main structural elements as their "Internal Behavior" comprises the contained REs, the communication taking place within the component (between several REs) and between different SWCs of one ECU. Each RE may be instantiated multiple times (e.g. four wheel speed sensors of a car), so every Runnable Entity instance (REI[3]) has its own data dependencies, which each arise from the interaction between at least two REIs. AUTOSAR differentiates seven kinds of variable accesses used for "local" (intra-SWC) access as well as for communication that crosses SWC borders, which are all taken into account.

Furthermore, timing constraints are identified: AUTOSAR provides seven such constraints and – as explained in the following two subsections – two of them are currently detected and deployed: "ExecutionOrderConstraints" (EOCs) and "AgeConstraints" (ACs) [4, 14]. The former are "[...] used to specify the order of execution of ExecutableEntities" (i.e. specify a fixed order for multiple REs) and the latter "[...] to specify a minimum and maximum age that is tolerated when a variable data prototype is received" (i.e. determine the tolerated data age of a read variable) [4].

EOCs can dictate a rigid execution order between two or more REs, which is suitable if they are logically (and semantically) connected as it is the case in a classical "sensor-controller-actor system", where sensors transfer measurement data to a controller that computes the appropriate action carried out by succeeding actors, e.g., brakes in an anti-lock braking system (ABS). As opposed to that, ACs solve potential conflicts via marking a potentially conflicting dependency as unproblematic by allowing that certain accessed data originates from a previous "computing cycle"[4]. This is viable if the reading RE does not require current data to work correctly, e.g., a speedometer that cannot (and is not intended to) react within milliseconds due to the speedometer needle's inertia.

Taking all this information into consideration allows to map them on a directed graph illustrating the data-flow by means of nodes that represent the REs and edges standing for the variable accesses semantically connecting them.

The second step harnesses the gathered information, so that the graph can be used to derive sets of node neighborhood, i.e., successor and predecessor relations between REIs, for the access on a specific variable. These sets are useful to make statements about possible execution sequences and to accordingly classify the dependencies in order to find possible inconsistencies, which are represented by every contingency of unintentionally consuming data before producing it in the scope of one computing cycle.

Furthermore, it is checked whether existing timing constraints are correctly set (valid). EOCs should only be set for REs that have the same recurrence, because when dealing with ECU software, static scheduling is prevalent. Therefore, a once found execution order stays the same. In contrast, EOCs are rather inappropriate for REs with diverging recurrences, because the execution order of them within a computing cycle can change, e.g., when a "consuming RE" is triggered more often than a "producing one". In such a case, an EOC demanding the producing RE to be executed first (within a computing cycle) can cause extra latency if the consuming RE is triggered earlier than the producing one. In such cases, ACs are the more adequate means.

Besides simply missing constraints, typical fault cases are, e.g., EOCs imposed on pairs of REs with diverging periods, EOCs directly contradicting each other (e.g. "(A before B) && (B before A)") or "insufficient" ACs that allow a smaller data age than actually occurring.

In order to reach the goal of "multi-core robustness", the validation process is intended to resolve the model's potential conflicts to prevent every unintentional consuming before producing. This is achieved by imposing a minimal amount of timing constraints to restrict all possible execution sequences to a set that provides every REI with its required input data in time.

It is important to know that EOCs do heavily restrict the "degree of freedom" for mapping the REs (grouped as tasks) to cores. Thus, the potential for parallel execution is greatly reduced when a model is highly order-constrained. Parallelism can even be entirely prevented if the EOCs' combination lead to a single-chain execution order. Therefore, as little EOCs as feasible are imposed, whereby they are preferably set in a local scope (e.g., being valid only within a SW-C). In the case of imposing constraints on dependencies that cross SW-C borders, ACs are an appropriate choice, because their (global) effect is less limiting and they do not per se reduce the number of possible execution orders (which is advantageous for the multi-core use case). However, ACs and EOCs are not mutually exclusive, but rather can be very expedient in combination (depending on the specific situation). This is in particular the case if a variable is used for different purposes, e.g., the value of a current wheel speed is frequently read by an ABS but only seldom by the speedometer.

After eradicating all potential conflicts including those that influence parallelization behavior, it is possible to ensure a system's validity with regard to data age and the corresponding model is ready to be split up safely into functional blocks that can be mapped on different cores.

3.2 Partitioning & Mapping

After having found consistency threats and solving them with the aid of constraints, the next logical step is to determine how the application can be split up and distributed in a feasible way: "Partitioning" means breaking up a model into sets of REs according to

[3] This is not an official but an implicit AUTOSAR element.

[4] We define a computing cycle as the time elapsed between two events that involve periodically activated tasks being guided by the slowest (least triggered) task occurring.

a given objective, while the succeeding "mapping" is intended to determine tasks within the obtained partitions and to assign them to specific cores, so that their later execution can be scheduled.

3.2.1 Overview

Unfortunately, there is no general-purpose approach to find a partition or mapping and it is difficult to assess whether a specific solution will satisfy a certain property. Thus, it is crucial to thoroughly consider the desired aspects of the target system and the objectives in advance, which is usually done according to specific goals like ...

- having only little coupling between the tasks and therefore rather few communication and/or necessary synchronization ("pooling"),
- assuring certain safety requirements like distinctly separating highly critical tasks or
- preserving the processing of logically related software parts on one core, e.g., REs contributing to one common function.

As there are numerous possibilities to fractionalize a model, finding an "optimal" partition according to specific goals is ranked as NP-hard problem [8]. In addition, searching for an advantageous "task-to-core" mapping entails traversing an overwhelmingly huge solution space, because the number of mapping possibilities grows exponentially according to the amount of given tasks. Together, both activities represent one of the biggest challenges when trying to build an optimized multi-core system.

An easy sample calculation shows how the search space quickly escalates even for small examples: The "Brake-by-Wire" application from "TIMMO"[5] and "TIMMO-2-USE"[6] consists of clearly organized 18 REs [45, 46].

Assuming that each RE is supposed to be mapped separately on one of three available cores, there are about 388 million (387420489, i.e. 3^{18}) different ways to do so. After choosing one of these distribution solutions, there are again many possible execution sequences: there are over six quadrillion ("18 factorial", i.e. $6 * 10^{15}$) sequences for executing all REs successively on one core. And there are a lot (exponentially) more options in a multi-core setting, because most REs can theoretically be processed in parallel (fully or partially overlapping). Generally speaking, every random set of REs can be simultaneously executed as long as it is valid regarding the absence of two REs being interconnected by an EOC. The exact count of possibilities depends on the number of available cores (defining the maximum set size), the number of tasks encapsulating the REs and possibly given minimum requirements for load balancing (together with execution times).

Therefore, we aim to reduce the number of possibilities to consider by first providing a beneficial initial partition and secondly – based on this starting point – an advantageous initial mapping, which increases the efficiency of the following scheduling, simulation and optimization. As the partition is created with respect to imposed constraints and existing dependencies, the subsequent computational effort is limited to a "corridor" of preferably promising solutions. Since it cannot be guaranteed that proper paths are not discarded, this process should be repeated in order to ensure a balance between searching deeply and broadly.

[5] The project "TIMing MOdel" developed "[...] a common, standardized infrastructure for the handling of timing information during the design of embedded real-time systems in the automotive industry".

[6] The project "TIMing MOdel - TOols, algorithms, languages, methodology, and USE cases" provides "[...] tools, algorithms, languages, methodology, and use cases for dealing with timing requirements and properties for timing analyses during the development of distributed embedded automotive systems".

Without a given partition and if no further knowledge of the system is available, a simulation tool would be forced to draw on simple strategies to obtain initial tasks, e.g., preferably encapsulating REs with equal recurrences and therefore creating homogeneous and easily relocatable tasks. Such regions are particularly suitable for being executed on a common core, so that the duration of one "computational iteration" on this core is not needlessly delayed due to REs' recurrences that are cumbersome to reconcile. However, such a partition can be very adverse too, especially when load balancing is hampered by strongly differing task sizes or when – as it is almost always the case – cross-core communication is an issue and heavily connected REs are not assigned to the same core. According to our experience, this holds particularly true for highly complex models like the one used in the Case Study (cf. Section 4).

3.2.2 Partitioning

As stated in Section 2, "low coupling" (corresponds to "pooling") acts as standard partitioning objective. It is determined by counting the dependencies that cross region borders within a certain partition, i.e., data accesses that are "broken" by assigning the involved REs to different regions. Restoring these dependencies (preserving their function) requires additional synchronization effort, because at scheduling, the execution of the respective REs has to be coordinated according to their specific cross-linking. Furthermore, we start from the premise that a target system's processor has "homogeneous", i.e. equally equipped, cores.

This concept is realized by the "Single Entry Region Analysis" algorithm that searches for virtually isolated RE sets within the model. They are characterized by a common starting point (the "entry node") and by not having any dependencies to outside nodes before a common end point (a merger node) "closes" the region. Details on the algorithm, an exact definition, its origin and implementation are available in [25].

As broached in Section 2, the algorithm used to be not productive enough for highly complex models as its strict rules were not defined for heavily interconnected graphs. In order to make it applicable to all models, we purposefully extended it to meet the emerging requirements:

Search tolerance: Being configurable according to the specific model's complexity, the algorithm accepts a certain number of "isolation violations" without immediately discarding the identified RE set. This is useful to perform a search that takes the average node degree (i.e. the number of dependencies per node) into consideration, making it possible to find "hot spots" even in dense graphs. Based on experience, it is – in most cases – relatively easy to detect a sensible upper limit for this tolerance, because found groups beyond this "turning point" are often too bulky and not significant anymore.

Dependency weights: Treating the connection between all node pairs equally is obviously not expedient when having to decide which one to "break" while trying to form RE sets. Therefore, we calculate weights for the connection degree of every connected node/RE pair using the information usually available in AUTOSAR models: the REs' period and the number of dependencies (variable accesses) connecting them. The weight value rises according to periods getting smaller (meaning a higher triggering frequency) and a rising number of dependencies. The formula is:
$weight = (1/period1 + 1/period2) * dependencies.$
It is easily adaptable if further information (like the amount of transfered data of a specific variable access) is given and serves as basis for the later introduced "relevance partitioning".

Splitting strategy: As previously mentioned, it is basically advantageous to identify groups whose REs have a uniform recurrence. This can be achieved by different strategies:

- "Split, then analyze": In our experience, building subgraphs that consist of uniformly triggered REs and then running partitioning searches on each of them, has produced the most valuable results for highly complex models. In addition, the overall search effort is remarkably reduced.
- "Analyze, then split according to periods": This strategy takes the graph as is and assumes that the search finds sufficient groups, which can afterwards be split according to the number of diverging RE periods occurring. This is rather suitable for small heterogeneous or for huge but loosely connected models.
- "Do not split, discard mixed regions": Here, identified regions are discarded if they do contain REs with diverging periods. This can be useful for models with a small amount of different periods that are nevertheless relatively complex.
- "Do not split, keep mixed regions": As pretty simple strategy, this approach is rather used as starting point to gain an insight into the possible partitioning degree of a model in general.

Relevance partitioning: It is in most cases rather fruitless to pursue simple partitioning approaches like, e.g., "Sparsest Cut" which repeatedly cuts a graph into two (roughly) equal-sized pieces [9]. This is due to strongly differing model structures which are usually not suitable for being strictly divided into 2^x parts.

Thus, we use a more sophisticated approach vaguely resting on "Multi-Level Partitioning" (MLP), which better adapts to specific model structures [6]. MLP reduces a graph via "edge contraction" ("coarsening") in order to cluster and afterwards restore it. However, we do not "erase" nodes/edges but gradually increase the relevance threshold for dependencies taken into consideration by the search until the graph is "manageable" enough to find appropriate RE sets.

Due to the dynamic adaption, the algorithm can cope with models of any size and complexity. However, this does not mean that every application can be efficiently parallelized but it is almost always possible to identify a proper partition according to circumstances.

3.2.3 Mapping

The partitioning algorithm determines preferably large RE sets, which can – hierarchically structured – contain smaller ones. This is done deliberately in order to maintain RE sets of every size and therefore to retain all granularities for a later mapping of tasks to cores.

As the count for both, the mapping of tasks to cores and the possible execution sequences, strongly depends on the initial number of tasks, seeking to prevent a too fine-grained partition (many small tasks) is a reasonable trade-off because fine granularities provide more flexibility but they involve much more effort to distribute and are harder to synchronize.

As opposed to this, a coarse-grained partition "[...] can more easily result in an improvement" and thus seems appropriate as a first step [29]. However, having only very few large tasks can make it difficult to distribute them properly on different cores without again causing overhead for additional synchronization, e.g., if being forced to map two intensively connected partitions on different cores or when trying to achieve even workloads for cores (load balancing).

Of course, the latter – as well as the whole mapping process – is only possible when the target hardware is known as the number of available cores and their dependencies among each other (e.g. differing features) are crucial.

Our principles remain pooling and load balancing, for which we need to sensibly choose a suitable size for each available RE set in order to find the most convenient mapping. This is due to the fact that a partition does usually not contain groups with uniform size and therefore following a rather coarse-grained approach should not lead to a brute force method like "streamlining" the partition by reducing large groups.

Eventually, we take the following aspects into consideration when creating a mapping:

- Number of cores (or generally speaking "independent execution units") being available on the target platform
- Task clustering strategy: preferred relative regions size (if they are nested) and handling of remaining REs (e.g., a new task for each or create clusters according to periods)
- Distribution: assignment of tasks to cores according to algorithms that take heed of the expected workload caused by the tasks and that draw on well-known patterns like "bin packing" or "round robin"

For our Case Study (Section 4), we use an exporter tool that creates a CSV file comprising these aspects. Additionally, it produces basic timing requirements, i.e., task deadlines according to given periods, to support the succeeding simulation.

3.3 Scheduling & Simulation

In order to determine the actual benefit when harnessing the information acquired by the previous two steps, it is necessary to create a final schedule and to simulate the software's execution on a multi-core platform. This step (and the following optimization) can be carried out by use of various third-party tools that are designed for simulating (and optimizing) embedded real-time software, such as:

- "chronSIM" by Inchron GmbH is a "[...] tool for design, visualization, quality testing and analysis of embedded systems" [21].
- "SymTA/S & TraceAnalyzer" by Symtavision GmbH are "[...] tools for scheduling analysis, architecture optimization and timing verification for: ECUs and software integration, Embedded networks and communication, Distributed embedded systems (E/E)" [43].
- "TA Tool Suite" by Timing-Architects Embedded Systems GmbH is intended to be integrated in "[...] whole development process of multi-core systems" and is used for "[...] designing, developing, and verifying embedded multi- and many-core-systems" [44].

Like already mentioned in Section 2, there are some basic steps that are necessary before being able to perform a simulation:

Import the AUTOSAR model: As we conduct our data dependency analysis and the consistency validation directly on AUTOSAR system descriptions, it is crucial that the employed tool can handle such models by processing its structural elements (REs, SWCs, variable accesses) as well as included timing information like the REs' recurrence and timing constraints (ACs, EOCs). In general, every piece of information that can be retrieved further refines the simulation, e.g., data on the REs' execution times remarkably improves a simulation's accuracy.

Include data on partitioning and mapping: This information can either be stored directly in the AUTOSAR models (like described in [25]) or it can be imported from an external file. We currently use the latter approach to import necessary data from an CSV file: the assignment of REs to tasks, the mapping of tasks to specific cores, the tasks' recurrence and task priorities.

Setting basic requirements: In addition to the aforementioned data on tasks and their core assignment, it is essential to add basic timing requirements for them, i.e., upper limits (deadlines) for the

tasks' response time according to their respective periods. They enable the basic assessment of a simulated solution regarding its general validity.

Configuring the hardware model: Providing the simulation with concrete details on the target hardware helps to enhance the significance of the results. This includes – among other things – the number of cores, their clock rate (frequency), their cache and available shared memory. In addition, existing signals (i.e. the model's variables) have to be initially mapped to certain memory modules. In the Case Study (Section 4), we use the model of a real-world automotive microcontroller featuring three cores and assign the signals to core-local memories according to the made task-to-core mapping.

Selection of scheduling algorithm: Usually, a simulation tool provides different algorithms for the scheduling of the tasks. Popular examples are "Rate-Monotonic scheduling", "Deadline Monotonic Scheduling", "Earliest Deadline First" or "Proportionate Fairness" [3, 26, 41]. There is no universally valid heuristic helping to select the optimal one according to a certain model. Therefore, running tests with a couple of them seems to be expedient.

The more concrete parameters are set, the more realistic and expressive the obtained results are. In particular, we selected the following key figures to represent the solution's quality:

- General validity: A solution is valid if the simulation proves that all basic timing requirements of the tasks are fulfilled, i.e., all tasks are fully processed before they are triggered again.
- Average latency: If compared to the best known other solution, the maximum response time for a whole model (derived from the latencies of its tasks) is a meaningful value revealing needlessly caused overhead. In addition, it gives a hint on the model's overall potential degree of parallelization.
- Communication overhead: As different cores execute tasks that depend on each other, a certain rate of "cross-core communication" is virtually always inevitable. Like the latency, this rate can be assessed relative to other solutions through comparison.
- Average core load: This indicates how uniform the division of processing work is (proper load balancing).

This measured data serves as basis of comparison for solutions calculated by the succeeding optimization.

3.4 Optimization & Comparison

After the simulation has returned key figures for the initial solution, further ones can be generated by using it as alterable seed. According to the strategy of the applied optimization software, the initial solution is modified to a certain extent and then re-assessed in order to learn if the changes are beneficial. In the Case Study (Section 4), the employed third-party optimizer uses a genetic algorithm to automatically create new "solution generations". Doing this manually by inspecting the simulation results and purposefully varying (now simpler to identify) critical parts is also expedient yet most likely more time-consuming.

The outcome of the optimization is heavily influenced by the set of parameters which can be changed by the algorithm. Our approach does not exclude any part of a solution from being altered. Thus, no limits are set for modifying a model, which allows a variety of different optimizer settings. Some common "leverage points" for variation are, e.g., changing the core assignment of tasks and – accordingly – the signals' mapping to core-local memory, altering task priorities or splitting given tasks at several positions.

In the end, the crucial insight is if the optimization can – in regard to the mentioned key figures – deliver distinctly better solutions and how much additional effort (most notably time) is necessary. If the latter remains within acceptable limits, the optimization results can give valuable feedback for the "Partitioning & Mapping" step in the form of, e.g., narrowing down the "corridor" of expedient amounts of groups and their sizes or indicating which groups are recommended to be assigned to one common core from the very start ("pairing").

4. Case Study

In order to illustrate the applicability and benefit of our approach, we apply it to two AUTOSAR models: the mid-sized sample "DemoCar" originating from the "AMALTHEA Project"[7] and a part of a huge real-world engine management system from Continental ("Continental EMS"). The former consists of one SWC containing 43 REs with 3 different recurrences, 71 variables/signals and 59 variable accesses (dependencies). The latter comprises 178 SWCs including 552 REs with 20 different recurrences, 11460 variables/signals and 45399 variable accesses.

First, our "Data Dependency Analysis Tool" (cf. Section 2) analyzes the structure, variable accesses and timing properties of the models. The subsequent partitioning step employs the "split with respect to periods, then analyze" splitting strategy and conducts a coarse-grained low-coupling search (with increasing fault tolerance) on parts of the models. For the DemoCar, it is relatively easy to quickly find a near-to-optimal result as it is structured in an straightforward way and does not include many data dependencies. The Continental EMS is significantly more complex, thus only a combination of rising tolerance values and onward coarsening allows to find justifiable partitions. Our tool needs less than a minute for the analysis and partitioning search when being run on conventional laptops or desktop computers.

We create two partitions for both models: one with rather small RE sets, one with large RE sets. In order to enable a comparison to searching without a preceding data dependency analysis and partitioning/mapping, we also include a simple partition for both models that assigns all REs with uniform period to one task/group (simulating a situation where no further knowledge about a system is available).

The mapping is geared towards an embedded platform featuring three cores (details below). The initial distribution follows a "bin packing" approach that estimates the specific RE set's core utilization and assigns it accordingly as task on the least busy core aiming at a proper load balancing.

In addition, process requirements are created that define fundamental deadlines for each task considering their recurrence. That means that the included REs' uniform period determines the whole task's maximum response time. They are mandatory for the purpose of classifying later found solutions into "valid" (viable) or "invalid" (unsuitable).

We use the above-mentioned "TA Tool Suite" (TATS) to simulate and optimize the models as it provides the required import functions for both, AUTOSAR system descriptions and task assignments provided via CSV files.

In order to achieve results that are as realistic as possible, we use the model of an "Infineon AURIX TC27x"[8] microcontroller as hardware platform, which is "[...] designed for ultimate reliability in harsh automotive environments" [23]. It features three processing cores that can be regarded as homogeneous [18, 22].

In TATS, a separate scheduler is assigned to each core. A selection of algorithms is available for the scheduling of the tasks on one

[7] "AMALTHEA is an open source tool platform for engineering embedded multi- and many-core software systems." [1].

[8] The "AUtomotive Realtime Integrated NeXt Generation Architecture" is a microcontroller family for the automotive sector featuring three independent 32 bit "TriCore" CPUs [23].

core. For this case study, we chose either “EDF” or “AUTOSAR” as strategy for all cores within one test run. The last necessary adjustment is the mapping of signals/variables to the memory. We initially assign the signals to the local memory of the core reading them. If multiple cores are involved, we distribute the signals equally across them.

Now, the simulation can be carried out. We simulate an execution of the system that lasts one second and delivers extensive feedback as well as our selected key figures. The crucial information is whether the basic requirements are met, i.e., if the hardware is – considering the given tasks, mapping and schedule – capable of executing the software fast enough to keep up with the recurring tasks.

Based on this, the succeeding optimization is conducted to compare the initial partition/mapping with further possible solutions calculated by a genetic algorithm employed in TATS. We conscientiously selected appropriate basic settings for the optimization, namely a simulation time of one second, “maximum Normalized Response Time” (mNRT) and “Inter-Core Communication rate” (ICC) as optimization goals, an exploration size setting with initial population size “32”, variation count “16” and selection count “16” as well as “stagnation for 5 iterations” and similar ranges of “generations” as stop criteria.

The most important setting is the granted “degree of freedom” when altering the initial solution. It is represented by selecting which modifications are allowed during optimization. We use the following options offered by TATS [44]:

- “Runnable Sequencing” (RS), to “[...] change the order of runnables inside call sequences.”
- “Task Splitting with Enforced Migration” (TS-EM) to “[...] enforce migration of tasks to other schedulers/cores.”
- “Task Splitting with Inter Process Activation” (TS-IPA) to “[...] split tasks into several subtasks.”
- “Process Allocation” (PA) to “[...] change the scheduler where processes are allocated to.”
- “Periodic Stimulus Offset Assignment” (PSOA) to “[...] change the Offset of Periodic Stimuli.”
- “Automatic Task Parallelization” (ATP) to “[...] partition tasks into several subtasks which run in parallel on different cores.”

Among these, TS-EM and TS-IPA, PO and ATP as well as TS-IPA and ATP are each considered mutually exclusive because they interfere with each other. With respect to that and in order to cover a broad search span, we apply different “strategy sets” to find preferably advantageous solutions. The steady basic setting is to allow RS and PA as well as to choose maximum allowed splitting/migrating values for either TS-EM or TS-IPA. The concrete sets arising out of this are:

1. allow RS, PA, TS-EM and ATP with optimization goal mNRT
2. allow RS, PA, TS-EM and ATP with optimization goal ICC
3. allow RS, PA, TS-IPA and PSOA with optimization goal mNRT
4. allow RS, PA, TS-IPA and PSOA with optimization goal ICC

The optimization of complex models like the Continental EMS can – due to the vast search space – take up to a whole day even with low exploration sizes and performed on a rather powerful laptop like the one we used (Dell M6700, Intel Core i7-3729QM, 8GB RAM, SSD hard drive, Win10 64 Bit). The results include a variety of data and statistics of these we concentrate on the ones representing the key figures described in Subsection 3.3:

- The general validity is represented by entirely fulfilled “Process Requirements” (deadlines).
- The average latency is expressed through the “maximum Normalized Response Time”.
- The communication overhead is shown as “Inter-Core Communication rate”.
- The core load balance is indicated by the difference of the cores’ individual “CPU Load (Utilization) average”.

The results of our test run series are shown in Figure 2 (“DemoCar”) and Figure 3 (“Continental EMS”).

Test Runs for the AMALTHEA DemoCar

Partitioning & Mapping	Scheduling & Simulation	Optimization & Comparison				
groups (distribution)	*algorithm <met deadlines>*	*strategy set: RS, PA and ...*	*optimiz. goal*	*init rank*	*#solutions valid*	*#solutions all*
simple (on one core)	AUTOSAR <0/3>	TS-EM, ATP	mNRT	n/a	165	241
		TS-IPA, PSOA	mNRT	n/a	186	225
	EDF <0/3>	TS-EM, ATP	mNRT	n/a	208	305
		TS-IPA, PSOA	mNRT	n/a	144	177
"SERA 24" (distributed)	AUTOSAR <22/24>	TS-EM, ATP	mNRT	56.	173	177
		TS-IPA, PSOA	mNRT	251.	253	257
		TS-EM, ATP	ICC	1.	327	337
		TS-IPA, PSOA	ICC	180.	319	337
	EDF <24/24>	TS-EM, ATP	mNRT	59.	176	177
		TS-IPA, PSOA	mNRT	174.	174	177
		TS-EM, ATP	ICC	4.	238	241
		TS-IPA, PSOA	ICC	62.	157	177
"SERA 31" (distributed)	AUTOSAR <29/31>	TS-EM, ATP	mNRT	72.	177	177
		TS-IPA, PSOA	mNRT	193.	193	193
		TS-EM, ATP	ICC	1.	177	177
		TS-IPA, PSOA	ICC	124.	333	337
	EDF <31/31>	TS-EM, ATP	mNRT	51.	193	193
		TS-IPA, PSOA	mNRT	289.	289	289
		TS-EM, ATP	ICC	1.	177	177
		TS-IPA, PSOA	ICC	124.	333	337

Figure 2. Results for the test runs of our approach performed on the “AMALTHEA DemoCar”

The included tables show how different combinations of specific partitions, mappings, scheduling algorithms for the simulation and optimization strategies are used to search for advantageous solutions. The rows can be interpreted as follows:

- Partitioning & Mapping:
  - “Simple” indicates a grouping without further knowledge as basis of comparison whereas “SERA X” denotes partitions and mappings found by our approach including a number of “X” tasks (RE sets).
  - “On one core” acts as initial mapping for the “simple” partitioning whereas “distributed” follows our approach of low coupling and load balancing.
- Scheduling & Simulation: “AUTOSAR” and “EDF” are the employed scheduling algorithms and the ratio below represents the validity of the initial solution.
- Optimization & Comparison: Here, the settings and strategies of the specific optimization run are stated together with the rank of the initially calculated solution among the total number of determined valid as well as all solutions.

The additional “ICC” optimization runs were only conducted if the initial solution was not assessed as “predominantly invalid”, i.e., violating more than 50% of the deadlines. Thus, they were left out for both models when the “simple” partitioning/mapping solution was evaluated. This also applies for the rank of the initial solution

Test Runs for the Conti EMS						
Partitioning & Mapping	Scheduling & Simulation	Optimization & Comparison				
groups (distribution)	*algorithm <met deadlines>*	*strategy set: RS, PA and ...*	*optimiz. goal*	*init rank*	*#solutions valid*	*#solutions all*
simple (on one core)	AUTOSAR <1/20>	TS-EM, ATP	mNRT	n/a	4	113
		TS-IPA, PSOA	mNRT	n/a	29	113
	EDF <1/20>	TS-EM, ATP	mNRT	n/a	106	129
		TS-IPA, PSOA	mNRT	n/a	141	144
"SERA 47" (distributed)	AUTOSAR <37/47>	TS-EM, ATP	mNRT	n/a	0	113
		TS-IPA, PSOA	mNRT	n/a	7	129
		TS-EM, ATP	ICC	n/a	0	177
		TS-IPA, PSOA	ICC	n/a	96	305
	EDF <47/47>	TS-EM, ATP	mNRT	149.	241	241
		TS-IPA, PSOA	mNRT	253.	289	289
		TS-EM, ATP	ICC	4.	305	305
		TS-IPA, PSOA	ICC	117.	321	321
"SERA 248" (distributed)	AUTOSAR <222/248>	TS-EM, ATP	mNRT	n/a	0	153
		TS-IPA, PSOA	mNRT	n/a	0	29
		TS-EM, ATP	ICC	n/a	0	177
		TS-IPA, PSOA	ICC	n/a	0	97
	EDF <248/248>	TS-EM, ATP	mNRT	90.	113	113
		TS-IPA, PSOA	mNRT	15.	29	29
		TS-EM, ATP	ICC	1.	129	129
		TS-IPA, PSOA	ICC	8.	85	85

Figure 3. Results for the test runs of our approach performed on the "Continental EMS"

column "init rank" which is not available if it cannot be compared to valid ones. Within the latter, a rank is determined by TATS via comparing the solutions' specific "fitness" – a value reflecting the calculated goal achievement.

In order to sum up the outcome of the test runs, we can state the following: Our approach – especially its focal point described here – contributes to ...

- ... avoiding most adverse starting points where many basic deadlines are violated. This is illustrated by comparing the simulation results of "simple" partitions/mappings to those calculated by our approach ("SERA X").
- ... quickly finding promising starting points for optimization when primarily aiming at low response times.
- ... quickly finding remarkably advantageous starting points for ICC-optimized solutions, where our EDF-scheduled initial solutions are all valid right from the start in this case study.
- ... generally challenge the existing software structure (such as the assignment of REs to SWCs) by analyzing, partitioning and mapping directly on RE level and independent from their given assignment to SW-Cs.

The time saved by providing a viable initial solution comes even more into effect when the exploration size is increased and optimization durations are – as a consequence of exponential growth – multiplied, e.g., when several days of optimization do not result in a – according to a certain goal – considerably better solution than the initial one that was created within minutes.

In this case study, our specific settings were chosen in order to compare preferably many and diverging partitioning/mapping solutions. Distinctly bigger exploration sizes for the optimization are possible with an significantly increased time exposure or by utilization of high performance computing resources.

5. Conclusion

Because of the inherent complexity of migrating single-core legacy ECU software for a proper execution on multi-core platforms, fresh methods and approaches are urgently needed.

In order to support the efficient parallelization of AUTOSAR application software on function level, we introduced a tool-supported systematic approach which helps software engineers in analyzing, validating, partitioning and mapping AUTOSAR model data. This process facilitates the subsequent scheduling, simulation and optimization tasks which are carried out to find proper solutions concerning preferably low overall latency as well as minimal cross-core communication rates. Our approach is designed to detect and solve potential consistency conflicts right from the start, to support the parallelization of AUTOSAR models by automatically providing concrete partitions and mappings and therefore by considerably reducing necessary subsequent search effort.

To verify the potential benefit of our approach, we applied it to a fictional mid-sized and a real-life complex model to obtain a variety of different partitioning/mapping solutions which we afterwards compared to the previously calculated one. Experiments have shown that a preceding data dependency analysis and a partitioning/mapping (building on its outcome) can significantly reduce time and effort for finding a suitable solution.

In closing, it can be stated that the automotive sector's demands are rapidly rising. It is already evident, that even many-core technology becomes increasingly wide-spread (e.g. growing number of cores with distributed memories or rather heterogeneous connectivity) [27]. Thus, the approach detailed in this paper can serve as starting point for coping with the challenges that arise from this development.

Acknowledgments

We would like to cordially thank the Timing-Architects Embedded Systems GmbH for giving us the opportunity to use their software within TA research partner program.

References

[1] AMALTHEA Project. An Open Platform Project for Embedded Multicore Systems. `http://www.amalthea-project.org/`, 2015. (accessed on November 13th, 2015).

[2] Artop Group. AUTOSAR Tool Platform. `https://www.artop.org/`, 2012. (accessed on July 20th, 2013).

[3] N. C. Audsley, A. Burns, M. Richardson, and A. Wellings. *Deadline Monotonic Scheduling – Theory and Application*. Citeseer, 1990.

[4] AUTOSAR. *Specification of Timing Extensions*, 2014.

[5] AUTOSAR. AUTOSAR Basic Information - Short Version. `http://www.autosar.org/fileadmin/files/basic_information/AUTOSARBasicInformationShortVersion_EN.pdf`, 2014. (accessed on October 28th, 2014).

[6] C. Aykanat, B. B. Cambazoglu, and B. Uçar. Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. *Journal of Parallel and Distributed Computing*, 68(5):609–625, 2008.

[7] M. Bohn, J. Schneider, C. Eltges, and R. Rößger. Migration von AUTOSAR-basierten Echtzeitanwendungen auf Multicore-Systeme. In *Workshop: Entwicklung zuverlässiger Software-Systeme (Stuttgart, Germany)*, 2011.

[8] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42(3):153–159, 1992.

[9] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the Hardness of Approximating Multicut and Sparsest-Cut. *computational complexity*, 15(2):94–114, 2006.

[10] M. Deubzer. Multi-Core Software Architecture - Moving Towards Software Engineering. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[11] M. Deubzer, M. Hobelsberger, J. Mottok, F. Schiller, R. Dumke, M. Siegle, U. Margull, M. Niemetz, and G. Wirrer. Modeling and Simulation of Embedded Real-Time Multicore Systems. In *Proceedings of the 3rd Embedded Software Engineering Congress*, pages 228–241, 2010.

[12] Eclipse Foundation. Eclipse Modeling Framework Project. `http://eclipse.org/modeling/emf/`, 2009. (accessed on July 15th, 2013).

[13] T. Eißenlöffel. *Embedded-Software entwickeln*. dpunkt, 2012.

[14] T. Flämig. Software Architecture Methods for Multicore - Distributed Development and Validation of Architecture in Collaboratively Engineered Multicore Systems. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[15] S. Fürst. An OEM's Point of View On Multi-Core. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[16] M. Gehrke, P. Nawratil, O. Niggemann, W. Schäfer, and M. Hirsch. Scenario-Based Verification of Automotive Software Systems. In *MBEES*, pages 35–42, 2006.

[17] U. Gleim and T. Schüle. *Multicore-Software*. dpunkt, 2012.

[18] GLIWA embedded systems. An Introduction to Automotive Multi-Core Embedded Software Timing. `https://www.gliwa.com/downloads/Multi-core%20Poster.pdf`, 2015. (accessed on November 13th, 2015).

[19] M. Götz, S. Roser, F. Lautenbacher, and B. Bauer. Token Analysis of Graph-Oriented Process Models. In *13th Enterprise Distributed Object Computing Conference (EDOC)*, pages 15–24, 2009. .

[20] R. Grave. Software Integration Challenge Multi-Core Experience from Real World Projects. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[21] INCHRON GmbH. chronSIM. `http://www.inchron.com/tool-suite/chronsim.html`, 2015. (accessed on November 11th, 2015).

[22] Infineon Technologies AG. AURIX TC27xT data sheet. `http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/aurix-tm-family-%E2%80%93-tc27xt/channel.html?channel=db3a30433cfb5caa013d01df64d92edc`, 2015. (accessed on November 16th, 2015).

[23] Infineon Technologies AG. 32-bit TriCore Microcontroller. `http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/channel.html?channel=ff80808112ab681d0112ab6b64b50805`, 2015. (accessed on November 16th, 2015).

[24] R. Johnson, D. Pearson, and K. Pingali. The program structure tree: Computing control regions in linear time. In *ACM SigPlan Notices*, volume 29, pages 171–185. ACM, 1994.

[25] J. Kienberger, P. Minnerup, S. Kuntz, and B. Bauer. Analysis and Validation of AUTOSAR Models. In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, 2014.

[26] D. Liu and Y.-H. Lee. Pfair Scheduling Of Periodic Tasks With Allocation Constraints On Multiple Processors. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 119. IEEE, 2004.

[27] H. Mackamul. AMALTHEA - An Open Source Development Platform for Embedded Multi- and Many-Core Systems. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[28] R. Mader. Timing and Design Tool Support in Continental Powertrain Multi-Core Platform. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[29] B. Moyer. *Real World Multicore Embedded Systems*. Newnes, 2013. ISBN 9780124160187. URL `http://booksite.elsevier.com/9780124160187/`.

[30] A. Multicore. Relevanz eines Multicore-Ökosystems für künftige Embedded Systems: Positionspapier zur Bedeutung, Bestandsaufnahme und Potentialermittlung der Multicore-Technologie für den Industrie- und Forschungsstandort Deutschland, 2011.

[31] K. J. Ottenstein and L. M. Ottenstein. The Program Dependence Graph in a Software Development Environment. In *ACM Sigplan Notices*, volume 19, pages 177–184, 1984.

[32] F. Padberg and O. Denninger. Multicore-Softwarefehler im Visier: Automatische Fehlererkennung in Entwürfen paralleler Programme. *OBJEKTspektrum, Ausgabe 01/2013*, 20(1):72–76, 2013.

[33] D. Patterson. The Trouble with Multi-core. *Spectrum, IEEE*, 47(7): 28–32, 2010.

[34] C. Saad. Model Analysis Framework. `http://www.informatik.uni-augsburg.de/en/chairs/swt/ds/projects/mde/maf/`, 2009. (accessed on July 20th, 2013).

[35] C. Saad and B. Bauer. Data-Flow Based Model Analysis and Its Applications. In *Model-Driven Engineering Languages and Systems*, pages 707–723. Springer, 2013.

[36] B. Schatz. Challenge Multi-Core TCU: An Applied Example of Multi-Core Migration. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[37] J. Schäuffele and T. Zurawka. *Automotive Software Engineering*. Springer DE, 2010.

[38] R. M. Schneider. The ARAMiS Automotive LSSI Demonstrators and the Lessons Learned. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[39] C. Shih, C.-T. Wu, C.-Y. Lin, P.-A. Hsiung, N.-L. Hsueh, C.-H. Chang, C.-S. Koong, and W. Chu. A Model-Driven Multicore Software Development Environment for Embedded System. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 261–268, 2009. .

[40] A. C. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, and B. Esbaugh. Parallelism via multithreaded and multicore CPUs. *Computer*, 43(3):24–32, 2010.

[41] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms*, volume 460. Springer Science & Business Media, 2012.

[42] H. Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3):202–210, 2005.

[43] Symtavision GmbH. SymTA/S and TraceAnalyzer. `https://www.symtavision.com/products/symtas-traceanalyzer/`, 2015. (accessed on November 11th, 2015).

[44] Timing-Architects Embedded Systems GmbH. Timing Architects Tool Suite. `http://www.timing-architects.com/ta-tool-suite/simulator/`, 2015. (accessed on November 11th, 2015).

[45] TIMMO. Timing Model. `https://itea3.org/project/timmo.html`, 2007. (accessed on November 16th, 2015).

[46] TIMMO-2-USE. Timing Model - TOols, algorithms, languages, methodology, USE cases. `https://itea3.org/project/timmo-2-use.html`, 2010. (accessed on November 16th, 2015).

[47] F. Tip. A Survey of Program Slicing Techniques. *Journal of programming languages*, 3(3):121–189, 1995.

[48] L. Wirbel. Embedded Multicore Goes Mainstream. `http://www.designnews.com/author.asp?section_id=1386&doc_id=231676`, 2011. (accessed on July 15th, 2013).