Investigation on AUTOSAR-Compliant Solutions for Many-Core Architectures

Matthias Becker*, Dakshina Dasari†, Vincent Nélis‡, Moris Behnam*, Luís Miguel Pinho‡, Thomas Nolte*
*MRTC / Mälardalen University, Sweden
{matthias.becker, moris.behnam, thomas.nolte}@mdh.se
† Research and Technology Centre, Robert Bosch, India
dakshina.dasari@in.bosch.com
‡CISTER/INESC-TEC, ISEP, Portugal
{nelis, lmp}@isep.ipp.pt

***Abstract*—As of today, AUTOSAR is the de facto standard in the automotive industry, providing a common software architecture and development process for automotive applications. While this standard is originally written for singlecore operated Electronic Control Units (ECU), new guidelines and recommendations have been added recently to provide support for multicore architectures. This update came as a response to the steady increase of the number and complexity of the software functions embedded in modern vehicles, which call for the computing power of multicore execution environments. In this paper, we enumerate and analyze the design options and the challenges of porting AUTOSAR-based automotive applications onto multicore platforms. In particular, we investigate those options when considering the emerging many-core architectures that provide a more scalable environment than the traditional multicore systems. Such platforms are suitable to enable massive parallel execution, and their design is more suitable for partitioning and isolating the software components.**

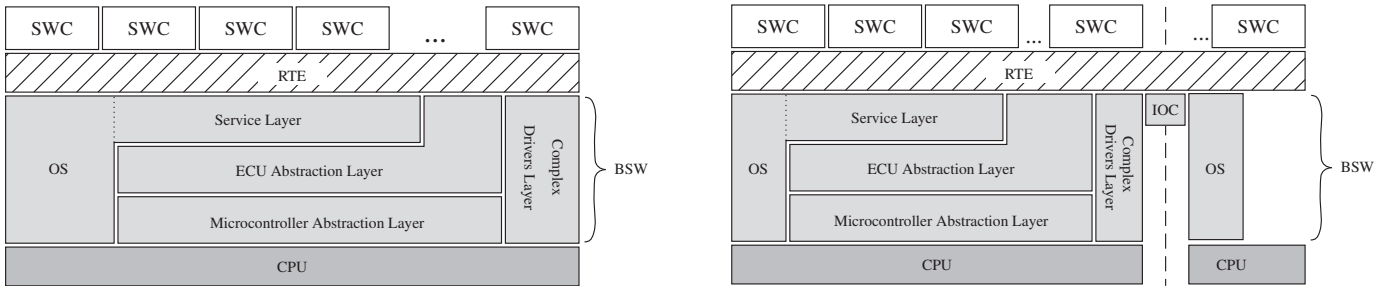
I. Introduction

Nowadays, the AUTomotive Open System ARchitecture, or AUTOSAR [1], is the de facto standard in the automotive industry. It was conceived by a consortium of automotive vendors with the objective of standardizing the Electrics/Electronics (E/E) architectures in modern cars, in order to enable an easier cross-development and integration of various software functions. This standard has seen various revisions to keep pace and to accommodate new system functionality and hardware platforms.

In the automotive industry, vehicles have transitioned from being basic (mechanical-intense) transport utilities to sophisticated (electronics-dominated) systems. The current trend is towards vehicles that are now capable of self navigation and maneuvering. This sophistication led to a steep increase of the number of Electronic Control Units (ECU) embedded in the vehicles and today's cars can have up to 100 ECUs. Car dynamics are getting more complex as well. For example, the complexity of control algorithms used in a modern Engine Management System (EMS) is steadily increasing in order to yield low fuel consumption while meeting the emission guidelines. All these trends lead to a rethinking of the E/E architecture paradigm in future cars.

Architecture-wise, there is a clear shift away from the "one ECU per function" design paradigm, which naturally resulted in numerous single-core ECUs spread all over the vehicles. Among the recent efforts to circumvent this issue of proliferation of the ECUs, one should cite the work of [2] in which the authors proposed a new centralized architecture based on Domain Control Units (DMU). In such an architecture, each functional domain of the vehicle is controlled by one DMU and a small number of ECUs to handle critical services such as airbags. Consolidating the various functionalities seems to be the way ahead and it is now technically feasible by replacing multiple ECUs driven by low-performance processors with a few more powerful processors. This advancement of the computing elements aims at providing higher computational power to applications that steadily demand higher performance and it allows to better manage the internal car electromechanics, including wiring issues, power consumption, form-factor, etc. While most current ECUs are based on single-core processors, a slow transition to multicore processors is already in progress [2]. The AUTOSAR standard has already been extended accordingly to cover these multicore platforms, and multiple vendors are bringing forth their multicore operating system implementations for AUTOSAR [3], [4], [5], [6].

At this juncture, it is important to analyze the new recommendations and guidelines of the AUTOSAR standard regarding a migration from single-core to multicore architectures, and also it is highly relevant to start investigating newer manifestations of multicore architecture designs, i.e., the many-core architectures. In many ways a many-core architecture is similar to any multicore design in that it hosts several cores on the same die, but it owns its different designation to a drastic shift in the arrangement of the cores on the die and the structure and topology of the inter-core communication technology (typically Network-on-Chip (NoC) based). Because of their distinct design, we believe that many-core architectures may succeed where traditional multicore designs have difficulties asserting themselves. In terms of performance, their flexible NoC-based architecture enables them to host a larger number of cores and offers evident enhanced computational capabilities compared to the former (traditional) bus/ring-based multicore platforms. This makes many-core platforms technically more appropriate for applications that require intense computations such as autonomous driving, object tracking, segmentation, navigation, etc. Moreover, the organization of the cores in separate and isolated structures (in tiles [9] or clusters [10] for instance) interconnected by a NoC may eventually provide a more interference-free execution environment in which



(a) Single-core architecture

(b) Multicore architecture

Fig. 1. AUTOSAR software architecture for single and multicore platforms as defined by the standard.

components may be truly isolated from each other. However adapting AUTOSAR to best harness the capabilities of the many-core platform is not straightforward and there are some challenges to be addressed.

AUTOSAR in Many-Cores – Challenges: The main challenge in the transition arises from the need to sustain the benefits achieved in the current single core architecture (interoperability, abstraction, etc). To enumerate (i) System services in AUTOSAR were designed for single core systems. The shift to large platforms introduces possibilities but also pitfalls trough the design of the system layer. This design can influence the resulting overall performance of the platform significantly. (ii) Parallelization of applications is one of the major benefits of many-core platforms. However, programming paradigms and sufficient support from the underlying software architecture needs to be provided in order to achieve the predicted performance gains. (iii) In addition to the functional properties of the system, extra functional properties need to be considered as well — most importantly, timing requirements and the adherence of safety standards.

As a main contribution of this paper we analyze and propose different design options of the AUTOSAR software architecture for these many-core platforms. Additionally we discuss the suitability of such systems to host diverse automotive applications and specifically address the issues of parallelism, safety and analyzability. Having communication between cores as an important part of a parallel system, we present the implications of the NoC on the core-to-core communication mechanisms defined in AUTOSAR. To the best of our knowledge, this is the first work which analyzes the AUTOSAR standard in respect to many-core platforms.

The remainder of the paper is organized as follows. Section II gives background information about AUTOSAR and its software architecture. Section III briefly outlines the type of hardware architecture that we consider in this paper. Section IV discusses the different AUTOSAR-compliant options for the design of the software architecture on such many-core platforms. Section V discusses benefits of parallelization of applications and the support for it in AUTOSAR. In Section VI we discuss, system analyzability and safety standards, as important extra-functional properties. Related work is presented in Section VII, followed by the conclusion in Section VIII.

II. The AUTOSAR Architecture

The AUTOSAR standard defines a software architecture for automotive applications. The application development in AUTOSAR and the design methodology is defined in [11]. It is based on the principles of component based software engineering [12]. This section summarizes the AUTOSAR software architecture and describes each of its conceptual and logical layers. Figure 1(a) shows the basic architecture and its main components: the Software Components (SWC), the Basic Software (BSW), the Run-Time Environment (RTE), and the Operating System (OS).

A. The Software Components (SWC)

Within AUTOSAR, the application software is organized in self-contained units called software components (SWC). Located at the highest architectural level, these components are platform-agnostic, meaning that at design time there is no need to consider how they will eventually execute on a particular ECU. Like many other component models, AUTOSAR defines interfaces to connect components with each other, as well as with the lower architectural levels [13]. These interfaces define both the required and provided services and data, as well as the communication protocol to be used, namely, trigger-based, data-based, mode-based, or operation-based [14].

Conceptually a component can be composed out of a number of other components, which allows for hierarchical design and reuse of complex subsystems. Such groupings, however, are only meaningful during the design phase. During system deployment to a specific ECU, all composite components are broken down into their atomic components. Atomic components have internal states and associated software functions which are referred to as *runnables*. Those runnables are the elementary parts of execution in the AUTOSAR world. They are mapped on and executed by the *tasks* of the operating system.

B. The Basic Software (BSW)

With more than 40 defined modules, the BSW constitutes the largest part of the AUTOSAR system. The BSW provides all services needed by the SWCs in order to fulfill their functions, while their concrete implementation is hidden from the application [15]. In contrast to the component-based SWC, the BSW is structured in layers (see Fig. 1(a)) the Service Layer, ECU Abstraction Layer, Microcontroller Abstraction Layer, and Complex Drivers Layer.

Service Layer: It provides fundamental services to the SWCs, and to other BSW modules. These services include the memory

management, communication, and diagnostics. Note that the OS is part of this layer.

ECU Abstraction Layer: As the name suggests, this layer abstracts the services provided by the hardware of the ECU. For example communication services are mapped to the respective buses.

Microcontroller Abstraction Layer: This layer provides basic drivers, abstracting the concrete microcontroller architecture. All layers above are thus agnostic about the microcontroller that is used.

Complex Drivers Layer: This layer consists of special-purpose functionality and drivers which are not compliant with AUTOSAR. Legacy drivers can be used in this layer as well. Since those drivers are not necessarily in the same layered architecture than the rest of the BSW, the Complex Drivers Layer spans all layers.

Additionally, the modules of the BSW can be divided into several functional groups, namely, the *system services*, *memory services*, and *communication services*.

C. The Runtime Environment (RTE)

The RTE is generated during system synthesis and realizes the connection between the SWCs and the BSW modules [16]. During the design phase, SWCs are interconnected only through the abstract Virtual Functional Bus (VFB). At a later phase, the SWCs are eventually mapped onto ECUs and depending on that deployment, two communicating SWCs can be located on the same ECU or on different ECUs. In the former case they must use *intra-ECU* communication mechanisms as a means to communicate whereas in the latter they must use an *inter-ECU* communication protocols, which requires the communication services offered by the BSW modules. The RTE is therefore a abstraction layer that provides a unified interface to the SWCs for their communication with other SWCs or with the BSW.

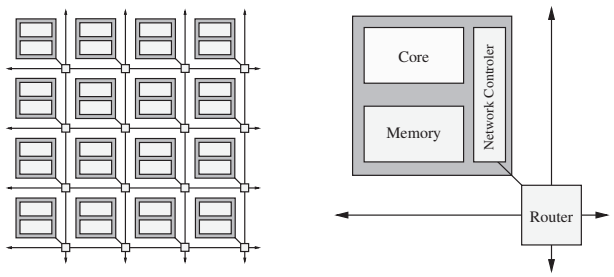
D. The Operating System (OS)

AUTOSAR uses the real-time operating system defined by the OSEK standard [17], [18]. The standard specifies a fixed priority scheduled operating system. Based on the configuration, the scheduling can be preemptive, non-preemptive or a combination of both, with a subset of tasks executing preemptively and non-preemptively. With the aim of facilitating temporal partitioning, two mechanisms are available in AUTOSAR:

Static schedule tables [17]: These tables define the time at which certain actions are performed (activation of a task, set event, start timer, etc.). Each action is given a particular offset relative to the start of the table.

Time monitoring: This mechanism is used to limit the time for which tasks can make use of different resources (CPU processor cycles, shared resources, interrupts). It ensures that tasks do not over-shoot their preset time budgets during runtime.

Spatial partitioning is another important requirement for real-time automotive systems. The responsibility of the OS is to protect the memory regions of OS-applications from other non-trusted OS-applications and also protect the address spaces of individual tasks within the OS-application [19]. This requires hardware support in form of a Memory Management Unit or a Memory Protection Unit. Detailed descriptions can be found in [17], [18].



(a) NoC architecture (b) Architecture of a tile

Fig. 2. Architecture of a many-core processor using a 2D-Mesh NoC

E. Multicore Extensions of the standard

With AUTOSAR 4.0, multicore support is also considered in the standard [20], [17]. The standard specifies that all cores operate on the same (shared) code base in order to reduce the footprint of the OS in the memory. The system shall not be viewed as a collection of virtual ECUs, rather as one distributed OS managing all cores. According to the standard, scheduling is done in a partitioned way, with each core having its own scheduler. Also, migration of tasks between cores is not allowed. The limitation of having the same code base for all cores implies that only one core executes the BSW modules at a time. The only part executed by all cores is the OS, having separate data structures for each core. Such an architecture is depicted in Fig. 1(b).

An important extension for multicore support in the standard is the addition of the Inter OS-Application Communicator (IOC) component, which is part of the OS. The RTE maps communication across cores or memory partition boundaries to the IOC module. The IOC operates using a sender-receiver protocol, where FIFO and unqueued (last is best) communications are supported. The IOC module implementation is dependent on the architecture and it also ensures data consistency. Inter-core communication between BSW modules is managed in a vendor-specific manner and is currently not standardized [20].

III. Hardware Model

The AUTOSAR extensions for multicore operations were designed with a simple hardware model in which multiple cores are connected to a shared bus in order to access shared memory and other peripherals [17]. However, as the number of cores in the same processor grows, new interconnection mediums and hardware architectures have been proposed, paving the emergence of “many-core processors”.

In typical many-core processors, cores and memories are located on tiles, connected by a Network-on-Chip [21]. In contrast to the bus/ring-based interconnection technology, the NoC provides a higher bisection bandwidth and scales up to a larger number of connected components. In this work, we consider many-core processors containing identical tiles,

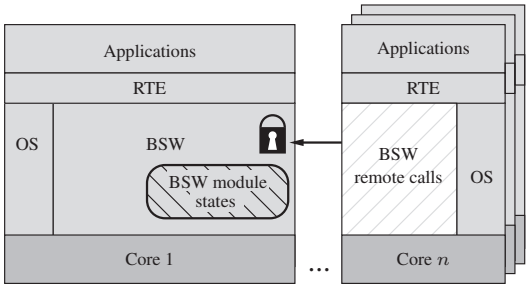


Fig. 3. Centralized approach as simplest design option, having all BSW modules on one core. Other cores use remote calls to access the BSW services.

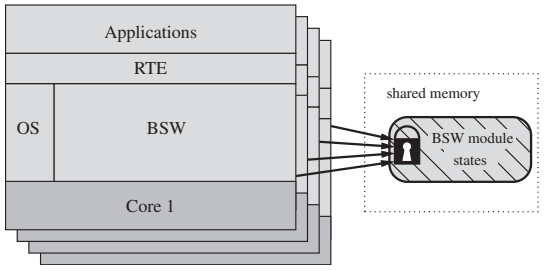


Fig. 4. Uniform distributed approach, having copies of all BSW modules in the core local memory.

interconnected by a 2-dimensional mesh-based NoC (see Figure 2(a)). Each tile contains one core, local memory, and a network controller, as illustrated in Figure 2(b). The network controller is used to connect to the router, which in turn connects to the other routers, forming the network-on-chip. Each core has a local memory with limited memory capacity (32KB onwards), depending on different many-core configurations. The access to local memory is faster in comparison to the access to the relatively larger off-chip memory. This architecture is similar to Tilera's Tile-Gx processor family [9], hosting 9, 16, 32 or 72 cores on one processor. While Tilera or Adapteva with their Epiphany processor [22] implement one core on each tile, Intel's Single Chip Cloud Computer [23] implements two cores per tile, and Kalray has 16 cores per tile comprising a compute cluster [10]. From a time-predictability point of view, the communication time between cores in a many-core processor depends on their locality, and in particular on the distance (in hops) between them, as a routing delay and possible interference needs to be considered for each hop [24].

IV. Design Options for the System Layer on Multi/Many-Core Platforms

Having massively parallel architectures, such as described in Section III, opens many opportunities for the deployment and execution of an AUTOSAR system, in particular since BSW clusters have been specified in the standard [20], [25]. The grouping of BSW modules into clusters, and assigning them to different cores, can be highly beneficial to the system in terms of performance. For example, all I/O modules can be mapped on one core and all network modules on another core, allowing for load balancing and parallel execution of those modules.

The performance can also be improved by grouping BSW modules of the same functionality into a *BSW functional clusters*. If BSW modules need to be available on several cores, the multicore extension of AUTOSAR [20] defines the master/satellite-approach. The implementation of the BSW modules is vendor-specific and thus the division of work between the master and the slaves/satellites is not defined. However, we can envisage only two approaches for that division of work: (1) a slave redirects all the calls to the master, i.e. the requests are remotely executed on the master core, or (2) a slave executes the complete request locally and communicate only the internal states to the master core. The choice of implementation of the master/satellite approach affects the performance of the system but does not affect the upper layers.

The next four subsection describe four different design options for a basic AUTOSAR system on a multicore platform and outline their respective benefits and drawbacks. It is important to note that *all presented design options are in line with the AUTOSAR specifications*.

A. Centralized Approach

The centralized approach allocates all the modules of the BSW to only one core, as shown in Figure 3. Each core runs its own OS and all the BSW calls are re-directed to the core hosting the BSW. Although this approach keeps the design simple and involves minimal changes from the existing single-core design, it generates a massive amount of cross-core communications, in particular if the platform contains many cores. This leads to the problem that tasks on the non BSW cores incur non-negligible blocking delays. Moreover, continuous interrupts from other cores can prevent the tasks running on the BSW core from having a continuous execution. It has been shown in [26] that the performance degradation for this design choice is significant, even considering two cores. In short, this approach does not seem appropriate for a many-core system considering the bottleneck at the BSW core.

B. Uniform Distributed Approach

As seen above, the centralized paradigm is inherently not in sync with the underlying distributed architecture of the many-core platform. The uniform distributed approach, which by design better aligned to the platform, may therefore be an interesting alternative. In this approach, the BSW is duplicated on each core in addition to the OS.

Having the complete BSW duplicated on each core has multiple advantages. First, BSW modules can be (in theory) accessed from different cores without causing interference on other cores. However, most of the BSW modules have critical sections (or exclusive areas) and hence cannot be accessed by different callers simultaneously. Therefore such BSW modules can still only be executed on one core at any time. In contrast to the centralized approach, when a SWC requests a BSW service, the service is executed on the local core itself and not on the remote core. In order to maintain the consistency of the internal BSW states, it is very important to save the states in a globally accessible data structure. This is to ensure mutual exclusive access and to allow every core to check whether the shared resource, i.e., the BSW module, is being accessed by another core (see Figure 4). A locking mechanism called the big BSW lock, similar to the Big Kernel Lock in Linux, was proposed by [26] to implement the above. An other

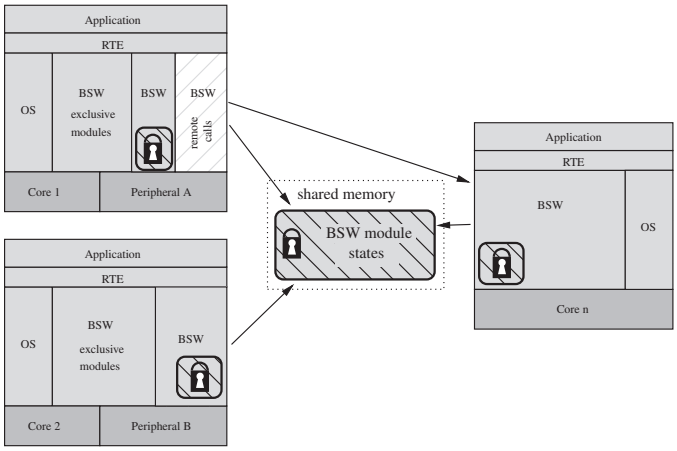


Fig. 5. Non-uniform distributed approach. BSW modules can exist on each core exclusive, as master-satelite approach or with global state.

advantage of such a design is that it provides total flexibility in the allocation of software components given the uniform capabilities of all cores.

On the other hand, one major drawback of this architectural alternative is the large footprint needed on the private memory of each core. This memory requirement obviously increases linearly with the number of cores. Another negative aspect is the fact that in such large platforms, not all the cores will have access to all hardware services, e.g. not all cores will have access to CAN, I/O, or Ethernet interfaces. Therefore, even though the execution of a BSW module is always carried out at the core that initiates the call to that module (because all the BSW modules are hosted by every core), there will be cases where some modules will need to communicate with other cores in order to access the requested hardware service. This does not defeat the uniformity of this design approach but must be considered for the lower protocol levels.

C. Non-Uniform Distributed Approach

The uniform distributed approach proposes local copies of the code, while the states of all modules are located in shared memory and thereby clearly improves upon the centralized approach. However, it still introduces a bottleneck, namely the shared states.

The non-uniform approach aims at mitigating that issue. In this approach, each core has its own local copy of the OS, stored in its private memory. From the geometry of the many-core architecture illustrated in Figure 2(a), it can be seen that hardware components such as the network modules for instance, are connected to certain peripheral cores at the edge of the NoC and to those cores only. Therefore, we can "break" the uniformity and restrict the mapping of the corresponding BSW modules (the network modules in this example) only to those cores. This way the BSW modules responsible for handling the communication over the CAN are only available on the core[s] that has access to the CAN interface.

An obvious advantage of this non-uniformity is a reduction in the code footprint, since a scaled down version of the BSW is allocated to the local and private memory of each core. Given that different modules are spread across the chip, the traffic is also distributed across and such a design decreases the resulting interference between the cores.

This can be combined with BSW modules following the approaches presented in Section IV-A and IV-B. Additionally modules, where global state is not needed (for example the module for cryptographic functionalities), can be allocated on multiple cores, knowing that there will be no cross core communication necessary in order to keep the state. Such a mixed architecture is depicted in Fig. 5.

The mapping of the SWC to the cores, however, becomes much more complex, since core affinities (a SWC is said to have a core affinity with one core if it is only able to execute on that core) need to be taken into account. The decision of which BSW modules should be local to each core and which BSW modules should be distributed becomes complex and depends on both the application and the hardware characteristics. As a consequence the mapping of the SWC to the cores is less flexible as all cores are not equipped with the same functionality, i.e. the same BSW modules.

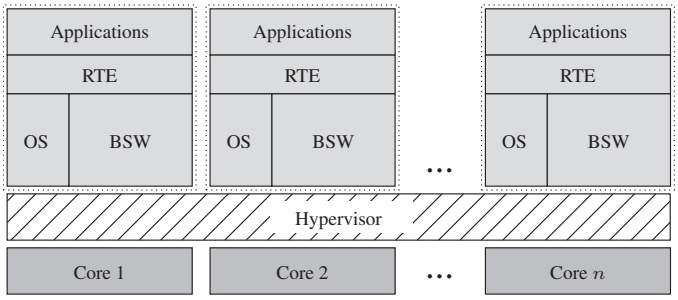


Fig. 6. Virtualization approach, a hypervisor is used to allow the execution of several AUTOSAR systems on the same or on multiple cores.

D. Virtualization Approach

As an alternative to directly deploy an AUTOSAR software on the system, virtualization techniques can be used to consolidate multiple (existing) AUTOSAR ECU configurations on a single platform [27]. In general, with virtualization, a system can host several partitions where each partition acts as an independent virtual machine (VM). Each VM can run a different instance of the OS and has its own resources (cores, memory, I/O, etc.), all managed by a hypervisor. In many-core systems, the large number of cores can be exploited and one partition can be executed on one core (see Figure 6).

Integrating AUTOSAR ECUs in this way has the advantage that the former ECU configuration can be kept, as it is executed as *virtual* ECU within one VM. This also implies that re-validation and testing efforts are not necessary/reduced and only the hypervisor needs to be additionally certified. Another benefit arises from the fault containment and safety properties that the isolated partitions provide. On the other hand, one of the major drawbacks of virtualization is the complexity of the hypervisor as it is responsible to abstract from the hardware platform. Most current hypervisors are designed for multicore platforms. The different architectures of many-core platforms add additional challenges to the design of the hypervisor. This is mainly due to the complexity of the NoC and the need to access all hardware services from all cores.

V. Improved Performance through Parallelization

In this section we first discuss the benefits of parallelizing the execution of AUTOSAR applications and later focus on the support in AUTOSAR through the IOC component, as it has direct influence on the achievable performance improvements.

A. Parallelism at the Application Level

One of the major advantages of the many-core platform is the provision for task-level parallelism. In the AUTOSAR context this means executing SWCs of one application on multiple cores in order to shorten the end-to-end computation time of the application. The current problem in the industry is to port the existing legacy sequential applications onto multicore platforms. Additionally, newer applications (for example in the computer vision domain) which naturally lend themselves to parallelism are also an integral part of the feature-suite in modern cars. To fully leverage the parallelism offered by these systems, there is a need for standards which not only define different parallel design patterns but also offer clear guidelines for mapping, scheduling, synchronization and communication. On the implementation side, the RTE will have to be augmented with means to parallelize an application into different threads and with scheduling techniques to ensure that tasks exclusively access shared resources (when needed) to minimize contention and maintain the state of the resources. To our knowledge, such guidelines are not yet formally defined in the latest AUTOSAR 4.2 standard [1].

A notable effort in this direction is taken by the EU project ParMerasa [28]. The main recommendations for AUTOSAR by these experts is to achieve the required speedup by minimizing the blocking during communication by using non-blocking buffers. A challenge in porting applications to a parallel platform is to ensure that the behaviour of the system before and after parallelizing the application is consistent — the execution and communication order must stay intact. To ensure this, experts in ParMerasa encourage time-triggered execution of entities (runnables) in a manner that the sending entity must always be scheduled before the receiving entities. They propose that the OS should support schedule tables for each core that run synchronously within few clocks. An other outcome of the project is presented in [29], where the authors present the RunPar algorithm. RunPar parallelizes tasks of a former single-core application as much as possible, while keeping the sequential execution order of the single core application, hence reducing the application execution time while avoiding the need for re-validation.

B. Communication in AUTOSAR

Together with the extensions for multicore, the Inter-OS-Application Communicator (IOC) was introduced in AUTOSAR [17]. Two intended use-cases are defined: (i) Communication between SWCs of different OS-Applications located on different cores of a multicore processor, and (ii) Communication between SWCs of different OS-Applications located on the same core, in order to guarantee memory protection between them. Since the current version of the standard does not allow OS-Applications to spread across multiple cores, the inter core communication is also always an inter OS-Application communication. As all other communication services, the IOC is abstracted by the RTE and thus transparent to the SWCs.

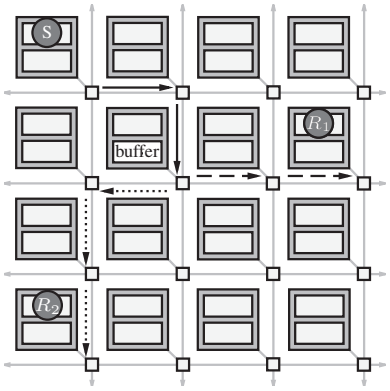


Fig. 7. Example of the effect of the shared buffer placement for 1:N communication via an underlying NoC.

Communication is always done in a sender-receiver fashion. The sending side writes data into a buffer, located in memory accessible by the respective communicating partners. One data item can be transfered via each IOC invocation, where one to one (1:1), many to one (N:1) and many to many (N:M) communication is possible. Communication itself can be queued or unqueued, except for N:M communication, where only unqueued communication is allowed. Different examples for the intended use of each relationship are given in [17]. In all cases data consistency is guaranteed by the IOC. Data transmission can be notification based, i.e. the sender notifies the receiver about the new data available. In that case, after the transmission of the data, the receiver is notified by a callback method which it needs to supply and which is triggered by the sender. The concrete mechanisms used to trigger the callback on the receiver side depends on the microcontroller platform that is used.

Several challenges arise when transitioning to a larger massively parallel platform. In contrast to the shared memory of todays multicore platforms, such many-core platforms provide a distributed memory architecture (see Section III). Thus, the memory used for the communication buffer, associated with a concrete message, can practically be located on all tiles (see Fig. 2(a)). Having the NoC in the access path to the communication buffer implies that the time taken for the communication depends on the location of the sender, the receiver, and the buffer itself. While this might be straight forward for 1:1 communications, where the buffer can easily be located within the receiver tile, it gets more complex if a N:1 or even N:M communication is used. In Figure 7, an example of 1:2 communication is shown. A sender S sends data which is read by receiver R_1 and R_2. In the example, the communication buffer as well as the sender and the two receivers are located on different tiles. While the communication path from buffer to R_1 is two hops long, the path from the buffer to R_2 takes three hops.

To reduce these delays, and thus their impact on the application performance, it seems beneficial to take the mapping of communication buffers into account when SWCs are mapped to the cores. Another approach could be to rethink

the underlying paradigms of the IOC and adapt to the new hardware platforms. Either way, a low overhead solution is needed in order to effectively leverage the advantages brought by the parallel hardware.

VI. Extra-Functional Properties

Automotive software design is complex due to the different metrics for assessment. The core-functional behavior must not only be in-line with the requirements, but also certain extra functional requirements must be addressed. In this section, we assess the suitability of many-core systems considering two important parameters in this regard: safety standards and timing analyzability of real-time systems.

A. Safety Considerations

An integral part of the automotive design is the adherence to the safety norms stipulated by the ISO 26262 standards [7]. Within this, each functionality in an ECU is assigned an ASIL (Automotive Safety Integrity Level) on the basis of the likelihood of a failure (after a risk analysis), the impact of that failure, and the likelihood of detecting that failure. The standard also gives explicit guidance concerning interference to other parts of the system, drawing particular attention to risks associated with interference between software components of different ASILs. The major reason for breaching the safety norms and causing incorrect behavior is due to interference in both the spatial and temporal domain. This may involve different shared resources – software resources like mutexes/locks and hardware resources like the shared communication channel. Interference can be caused in the spatial domain by memory corruption and by external data from insecure sources.

The standard proposes isolation as a basic mechanism for freedom from interference. By design, a many-core platform provides multiple natural physical partitions which may provide exclusive execution spaces for applications with different criticalites. While platforms like Kalray provide many clusters so that applications of different criticalites can be separately executed, Tilera provides "hardwalls" allowing groups of cores to be isolated and controlled w.r.t. communication with the external world. Another basic design merit arises from the fact that each cluster/tile has access to some limited local memory, which together with a memory management unit in software/hardware can provide the required spatial isolation. Each cluster or tile-group could host applications of a different criticality and the source software can then be developed as per the ASIL certification requirements only for that cluster. In other words, it could not be necessary for the software (OS + BSW) on each core to be certified to the needs of the highest ASIL levels, while allowing a mix of applications to reside on the same chip.

There are some advantages from the perspective of failover and redundancy as well in these systems. A given pair of cores can act as an active/backup core in order to have fail-safe execution, allowing the back-up core to resume execution when one of the core fails. A pair of cores can also be tuned to work in a lock-stepped manner to cross verify results when required in critical applications.

B. System Analysability

Many of the applications in a modern car have real-time software components, meaning they need to adhere some timing requirements. The temporal properties of applications like their Worst-Case Execution Time (WCET) must be determined at design time by timing analysis, which not only depends on the application code but also the execution eco-space (the platform: hardware and software). Given this, AUTOSAR describes the support for timing analysis in [30]. From the many-core system architecture perspective, it is important to consider the analyzability of these systems.

Today's multicore processors are heavily tuned and complex, designed for performance and not predictability, making timing analysis difficult [31]. Many-core processors on the other hand are generally equipped with simple cores. The computing cores on Kalrays MPPA 256 for example are fully timing compositional [10]. This means they do not exhibit timing anomalies which allows for tight WCET calculations. A second benefit of many-core processors is their memory layout. Platforms like Kalray provide different modes of memory bank accesses: interleaved bank accesses for highly parallel applications, and blocked memory mode, specifically targeted for embedded applications. With the blocked mode, application data is laid out sequentially along the address in a given bank, providing a kind of dedicated space for that application (subject to the bank memory capacity).

One prominent issue is the complex NoC fabric present in these systems, compared to the single shared bus which brings forth the issue of bounding the communication delay across the cores. Some vendors like Kalray claim that bounds on the delays can be computed using data flow models and sigma-rho calculus [32], but this issue must be taken care of. Different traffic groups, like I/O traffic, core-to-core traffic, or core-to-memory traffic, need to be handled by the NoC fabric. Different requirements for the different groups lead chip designers to implement separate networks for each group. This also allows for simpler analysability. Since for example, the traffic on the network used for core-to-core communication is not affected by the requests to off-chip memory, because they are handled by a different network.

Having the NoC as interconnection medium also introduces locality as one important factor. As discussed in Section V-B, placement of communication buffers affects the communication delays. Thus, the placement of software components on cores as well as the placement of the IOC buffers must take the locality aspects of those platforms into account. Additionally to the placement of SWC, the design of the system architecture (RTE and below) should consider the hardware characteristics. Having the different design options of Section IV, we can see that they introduce different degrees of possible blocking and thus introduce different degrees of pessimism in the analysis.

VII. Related Work

Several implementations of the AUTOSAR system for multicore exist from the industry as well as from the academia. In [4], Morgan and Borg outline the challenges encountered at ETAS, when moving to a multicore platform. They additionally present a prototype implementation of a multicore AUTOSAR system, including the needed multicore OS, IOC and RTE

TABLE I. COMPARISON OF THE PROPOSED APPROACHES

	Scalability	Memory Footprint	Modification on BSW modules	Time-Predictability	Mapping
Centralized Approach	**LOW** Blocking introduced by the remote BSW calls increases as the number of cores grows	**LOW** BSW only on one core and all other cores carry implementations of the remote calls	BSW modules must provide remote call functionality	Timing analysis becomes increasingly pessimistic and complex, since the BSW as a shared resource is frequently accessed by all cores, increasing the network traffic	Flexible: All SWCs can be mapped to all cores
Uniform Distributed Approach	**MEDIUM** Low blocking but footprint grows linearly with the number of cores	**HIGH** BSW is present on all cores plus states have to be kept in global memory, need for synchronization	BSW modules can be used with minor modifications, only states need to be accessed from global memory	Timing analysis needs to account for delays due to frequently accessed shared resources by all the cores; but message passing is not necessary	SWCs can be mapped to all cores
Non-Uniform Distributed Approach	**HIGH** Blocking and footprint size can be controlled by the selection of modules	**MEDIUM** Footprint lies between the centralized and uniform distributed approach, depending on the modules used	BSW modules should be provided in three implementation variants: Local exclusive, remote call, shared states	Pessimism in the timing analysis becomes reduced, since placement of BSW modules as well as their different variants reduce uncertainties	Mapping of SWCs must take the core dependencies into account
Virtualization	**MEDIUM** Having multiple guests will result in increased traffic on the NoC.	**HIGH** Guests use different implementations (different versions or even OS) independent of each other	No modification required	Timing analysis becomes complex due to the extra layer of abstraction (the hypervisor[s])	Mapping is already done, since existing ECU configurations are integrated by virtualization

support with cross task-activations. As the initial version of the standard dictates, they adopt the centralized approach in which the BSW is located on one core as described earlier in Section IV. With this prototype they were able to validate that the behavior on the multicore system did not vary from the single core setup but their experimental results clearly highlight the problem of workload distribution on the multicore.

An academic work was carried out by Böhm et al. [26]. They also evaluate an AUTOSAR multicore system with a centralized architecture. Having the need to lock the BSW while accessing it showed already performance degradation on their dual core experimental setup. When transitioning to a many-core platform, as considered in this work, multiple bottlenecks arise. With increased core count, the core hosting the BSW modules turns out to be the major bottleneck.

Bradatsch [33] compared two different implementation strategies for OS service calls, message and lock based, in a shared memory multicore implementation of AUTOSAR. Evaluation on a 2 and 4 cores processor showed clear advantages of the lock-based approach. However, this approach requires shared memory which is not implemented in the large parallel platforms considered in this paper.

More recently, two trends are popular. On the one hand, focus of most industrial implementations of the AUTOSAR multicore system lies on functional safety, as described by the ISO-26262. On the other hand, virtualization techniques are exploited to consolidate multiple systems (of same or different OS) on the same hardware platform.

An example for the former is EB tresos safety OS multicore from Elektrobit [5]; also the multicore operating system described in [3]. Both implementations introduce additional or refined functionality in order to cope with the requirements for functional safety. [3] for example allows program flow monitoring to supervise the execution of safety critical parts and their right execution order and time. Brewerton [19] discusses experience of an implementation of the steering column lock functionality on a multicore ECU. Having highest safety requirements (ASIL D), several hardware and software based solutions are discussed.

Virtualization support for automotive systems is discussed in [2], [27]. OpenSynergy, as an industrial example, provides COQOS [6]. With an avionics grade hypervisor, based on PikeOS [34], COQOS allows the execution of guests with ISO-26262 requirements. In [34], a PikeOS based prototype of a multicore based ECU, hosting multiple critical and non critical OS is discussed. The coexistence of such a variety of different safety and security levels is possible due to the SIL 4 certification of PikeOS.

VIII. CONCLUSION

All the works presented in the previous section target current multicore architectures, i.e., a low number of cores and shared memory, accessible by all cores. While the BSW implementations are optimized for multicore operation, the complete BSW is located on one core. The four design choices discussed earlier in this paper target much larger platforms without global memory. In Table I we compare those approaches with respect to scalability, required memory footprint, the need for modifications of an existing BSW implementation, predictability, and their influence on application mapping. Depending on the target platform and system requirements, different architecture choices are preferable. We believe it is possible to use many-core processors in the AUTOSAR context, but it is a challenging task. As discussed in this paper, different features found on such processors can be used, making the whole system more scalable and predictable. Many-core architectures can therefore be seen as a promising candidate for the future complex automotive systems. Future work will focus on the problem of mapping SWCs and communication buffers to the many-core architecture.

ACKNOWLEDGMENT

The work presented in this paper was partially supported by the Swedish Knowledge Foundation via the research project PREMISE and by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and when applicable, co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within project UID/CEC/04234/2013 (CISTER Research Centre), and also by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0001/2013 - JU grant nr. 621429 (EMC2).

REFERENCES

[1] *AUTOSAR*, last access April 2015, available at www.autosar.org.

[2] D. Reinhardt and M. Kucera, "Domain controlled architecture - a new approach for large scale software integrated automotove systems," in *International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2013)*, 2013, pp. 221–226.

[3] H. Brock and J. Kalmbach, "Autosar goes multi-core - the safe way," Vector Informatik GmbH, Tech. Rep., 2014.

[4] G. Morgan and A. Borg, "Multi-core automotive ECUs: Software and hardware implications," ETAS Group, Tech. Rep., 2009.

[5] *EB tresos AutoCore*, Elektrobit, last access April 2015, available at https://automotive.elektrobit.com/products/ecu/eb-tresos/autocore/.

[6] *COQOS*, Open Synergy, last access April 2015, available at http://www.opensynergy.com/en/Products/COQOS/.

[7] *ISO/DIS 26262-1 - Road Vehicles – Functional safety*, International Organization for Standardization / Technical Committee Std., 2009.

[8] Certification Authorities Software Team (CAST), "Position paper CAST-32 multi-core processors," FAA, Tech. Rep., 2014.

[9] *Tile-Gx Multicore Products*, last access April 2015, available at http://www.tilera.com/products/?ezchip=585&spage=614.

[10] B. D. de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager, "Time-critical computing on a single-chip massively parallel processor," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14, 2014, pp. 97:1–97:6.

[11] *AUTOSAR - Methodology*, AUTOSAR Std. 4.2.1, 2014.

[12] I. Crnkovic and M. Larsson, *Building reliable component-based software systems*. Artech House Publishers, 2002.

[13] *AUTOSAR - Autosar Model Constraints*, AUTOSAR Std. 4.2.1, 2014.

[14] *AUTOSAR - Software Comp. Template*, AUTOSAR Std. 4.2.1, 2014.

[15] *AUTOSAR - Layered Software Arch.*, AUTOSAR Std. 4.2.1, 2014.

[16] *AUTOSAR - Specification of RTE*, AUTOSAR Std. 4.2.1, 2014.

[17] *AUTOSAR - Specification of Operating System*, AUTOSAR Std. 4.2.1, 2014.

[18] "OSEK/VDX operating system," OSEK group, Tech. Rep., 2005.

[19] S. P. Brewerton, "Demonstration of automotive steering column lock using multiocre AUTOSAR operating stystem," *Society of Automotive Engineering (SAE)*, 2012.

[20] *AUTOSAR - Guide to Multi-Core Systems*, AUTOSAR Std. V1.1.0, Rev. R4.1 Rev3, 2014.

[21] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[22] *Epiphany Architecture Reference*, Adapteva Inc., Adapteva Inc. 1666 Massachusetts Ave, Suite 14 Lexington, MA 02420 USA, 2012.

[23] *Intel Single Chip Cloud Computer*, last access April 2015, available at https://communities.intel.com/docs/DOC-6005.

[24] D. Dasari, B. Nikolić, V. Nélis, and S. M. Petters, "NoC contention analysis using a branch-and-prune algorithm," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, pp. 113:1–113:26, 2014.

[25] *AUTOSAR - Guide to BSW Distribution*, AUTOSAR Std. 4.2.1, 2014.

[26] N. Böhm, D. Lohmann, and W. Schröder-Preikschat, "A comparison of pragmatic multi-core adaptions of the AUTOSAR system," in *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-Time applications*, 2011.

[27] D. Reinhardt and G. Morgan, "An embedded hypervisor for safety-relevant automotive e/e-systems," in *9th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2014, pp. 189–198.

[28] T. Ungerer et al., "parMERASA – multi-core execution of parallelised hard real-time applications supporting analysability," in *Euromicro Conference on Digital System Design (DSD)*, Sept 2013, pp. 363–370.

[29] M. Panić, S. Kehr, E. Quiñones, B. Boddecker, J. Abella, and F. J. Cazorla, "Runpar: An allocation algorithm for automotive applications exploiting runnable parallelism in multicores," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES '14, 2014, pp. 29:1–29:10.

[30] *AUTOSAR - Timing Analysis*, AUTOSAR Std. 4.2.1, 2014.

[31] G. Fernandez, J. Abella, E. Quiñones, C. Rochage, T. Vardanega, and F. J. Cazola, "Contention in multicore hardware shared resources: Understanding of the state of the art." in *14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, 2014.

[32] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, "Guaranteed services of the NoC of a manycore processor," in *Proceedings of the International Workshop on Network on Chip Architectures (NoCArc'14)*, 2014, pp. 11–16.

[33] C. Bradatsch, F. Kluge, and T. Ungerer, "Comparison of service call implementations in an AUTOSAR multi-core os," in *9th IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2014, pp. 199–205.

[34] S. Fisher, "PikeOS – a new standard for a safe and secure automotive platform." Sysgo AG, Tech. Rep., 2014.