Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2012, Article ID 354274, 14 pages
doi:10.1155/2012/354274

Research Article

How to Safely Integrate Multiple Applications on Embedded Many-Core Systems by Applying the "Correctness by Construction" Principle

Robert Hilbrich

Department Systems Architecture, Fraunhofer FIRST, Kekuléstraße 7, 12489 Berlin, Germany

Correspondence should be addressed to Robert Hilbrich, robert.hilbrich@first.fraunhofer.de

Received 8 December 2011; Revised 28 March 2012; Accepted 17 April 2012

Academic Editor: Michael H. Schwarz

Software-intensive embedded systems, especially cyber-physical systems, benefit from the additional performance and the small power envelope offered by many-core processors. Nevertheless, the adoption of a massively parallel processor architecture in the embedded domain is still challenging. The integration of multiple and potentially parallel functions on a chip—instead of just a single function—makes best use of the resources offered. However, this multifunction approach leads to new technical and nontechnical challenges during the integration. This is especially the case for a distributed system architecture, which is subject to specific safety considerations. In this paper, it is argued that these challenges cannot be effectively addressed with traditional engineering approaches. Instead, the application of the "correctness by construction" principle is proposed to improve the integration process.

1. Introduction

Multicore processors have put an end to the era of the "free lunch" [1] in terms of computer power being available for applications to use. The "end of endless scalability" [2] of single-core processor performance appears to have been reached. Still, the currently available Multicore processors with two, four, or eight execution units—"cores"—indicate just the beginning of a new era in which parallel computing stops being a niche for scientist and starts becoming mainstream.

Multicore processors are just the beginning. With the amount of cores increasing further, *Multicores* become *many*-cores. The distinction between these two classes of parallel processors is not precisely defined. Multicore processors typically feature up to 32 powerful cores. Their memory architecture allows the usage of traditional shared memory programming model without suffering from significant performance penalties.

Many-core processors on the other hand comprise more than 64 rather simple and less powerful cores. With a increasing number of cores, a *scalable* on-chip interconnect between cores on the chip becomes a necessity. Memory access, especially to off-chip memory, constitutes a bottleneck and becomes very expensive. Therefore, traditional shared memory architectures and corresponding programming models suffer from significant performance penalties—unless they are specifically optimized. Comparing their raw performance figures, the approach of having many, but less powerful cores, outperforms processor architectures with less, but more powerful cores [2, 3]. Of course in reality, this comparison is not as clear cut. It largely depends on the software, which has to be able to tap the full performance potential of these chips.

Many-core processors with up to 100 cores on a single chip, such as the Tilera Tile GX family (http://www.tilera.com/products/processors/TILE-Gx_Family) are currently commercially available. However, in order to benefit from these processors, an application has to exhibit sufficient parallelism to "feed" all available cores. Applications with these properties are hard to find among general-purpose software, because most applications have been developed for a single-core processor model. Parallel computing led a niche existence with the exception of its usage in high-performance computing.

In this paper, it is argued that software-intensive embedded systems will benefit from a massively parallel hardware platform. In particular, a new generation of embedded devices will require the increased performance in combination with a small power envelope: *Cyber Physical Systems* [4, 5]. These systems represent a tight integration of computation and physical processes and emphasize the link to physical quantities, such as time and energy. With the environment being composed of many parallel physical processes, why should not embedded systems controlling it, tackle this task with a parallel hardware platform?

Still there is a considerable gap to cover before this vision can be turned into reality. Therefore, this paper looks into several approaches on how to adopt many-core processors in software-intensive embedded systems. In Section 2, several adoption scenarios are described and assessed. Section 3 focuses on the integration of multiple functions and the description of domain-specific trends and developments, which aid the adoption process. Constraints and requirements which have to be addressed when multiple functions are integrated on the same platform are discussed in Section 4. Traditional and state-of-practice integration concepts are described in Section 5. Their shortcomings for many-core processors are used to argue for more formalized approaches in the context of a multifunction integration in Section 6. A case study, current research, and conclusions are contained in Sections 7 and 8.

2. Many-Core Processor Adoption Scenarios

Homogeneous and heterogeneous many-core processors will constitute the dominant processor architectures for all software-intensive embedded systems requiring flexibility and performance. Therefore, the embedded systems engineering community has to develop ways to adopt these new platforms and exploit their capabilities. The new parallel hardware platform offers a new level of processing power in a small power envelope. It enables innovative embedded applications—especially in the shape of next-generation embedded systems which are deeply integrated into the physical processes of the environment—such as *Cyber-Physical Systems* [5].

However, current engineering practices and software architectures for embedded systems are not ready to adopt a massively parallel hardware platform just yet. Current system designs heavily rely on a single-processor model with a serialized execution of tasks. The execution order of tasks is often expressed as task priorities. This approach helps to ensure proper timing by giving exclusive access for a running task to the underlying hardware. If interrupts are disabled, tasks may run to completion without interference. Dynamic scheduling algorithms at run-time focus on assigning priorities to tasks, so that the most important task is selected for execution by a dispatcher component. Albeit the simplicity of the execution model, programmers still struggle to implement complex applications. This is especially the case when quality requirements, such as real-time behavior or functional safety, have to be satisfied. Many-core processors raise the level of system complexity significantly by offering a parallel execution at run-time. With single-core processors already constituting an engineering challenge for certain applications, how should system engineers and integrators address the increased complexity of many-core processors?

A look on the state-of-practice in embedded systems engineering shows a considerable gap—that is, methodology and tools—that needs to be covered before many-core processors can be adopted in safety-critical domains. Although massively parallel systems have been extensively studied for high-performance computing, the research topic for the embedded domain remains: how to properly adopt massively parallel processors, so that the system performance is maximized and the required changes in legacy software are kept to a minimum. In this paper, this is referred to as the challenge of many-core adoption in embedded systems.

A migration to multi- and many-core processors will need to happen at some point in the future—especially for complex, software-intensive embedded systems. Simply, because there may be no more commercially available single-core processors on the market. Still, there is the chance of a sudden break-through in processor fabrication, which may allow to further increase the clock frequency beyond 5 GHz with only a moderate increase in power usage—thus avoiding the "power wall" [6]. However, at the time of writing, this appears to be unlikely [7]. Therefore, it is only sensible to look at possible multi- and many-core migration paths and analyze their effect on performance and the required changes in the software architecture.

2.1. "Single-Core" Approach. The most simple and straightforward adoption approach is to turn a many-core processor into a single-core processor by disabling all cores except one (see Figure 1). This allows to prolong the applicability of traditional engineering practices and experiences gained in previous projects. The impact on the software layer in terms of necessary changes to adapt to the new platform will be negligible, so that legacy code can be easily ported to the new architecture. For certification purposes, it may be necessary to prove that all remaining cores have been safely disabled so that no interferences at run-time are to be expected.

While this approach comes with a captivating simplicity, it is still a waste of resources, especially processing power, because parallel computing capabilities are not used. Therefore, the overall system performance is bounded by the single-thread performance offered. Given todays performance requirements, this may be sufficient. However, for the future, the single-thread performance will not increase significantly. Even worse, Borkar [7] argues that single-thread performance needs to *decrease* in the future in order to fit a potential 1000-core processor within a reasonable power envelope. Embedded applications may then get less performance in the future.

The "single-core approach" approach requires only minor software changes, but it lacks an *active and conscious migration* to a modern parallel processor architecture. It is a viable first step for specific application domains, for instance to gain a service history and experience for a specific

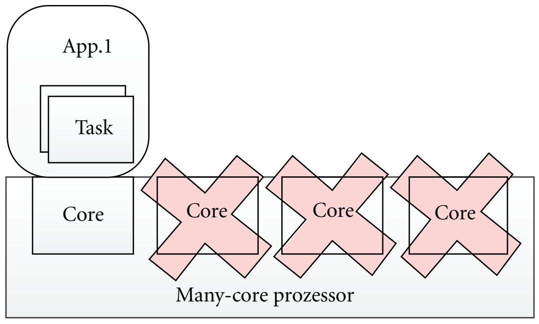


Figure 1: “Single-core” approach—disable all cores except one.

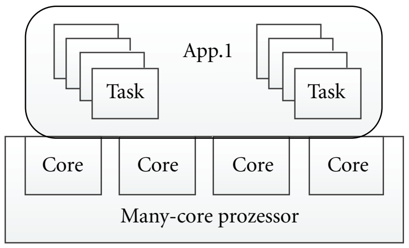


Figure 2: Use all cores run a single, parallelized application.

processor in noncritical areas. However, in the long run, the limited single-thread performance will prove to be a major obstacle in providing feature-rich applications.

2.2. Single Application on Multiple Cores. This approach tries to harvest the processing power of a many-core processor by parallelizing a single application running on a many-core processor (see Figure 2). This has a major impact on the software for the specific application. Depending on the application, data-parallel or function-parallel engineering patterns and libraries may help substantially in reducing the parallelization effort.

The impact on the entire system design, that is, the processor in the context of other processors and communication networks with regard to safety and reliability measures, is limited. From the perspective of a system design containing multiple electronic control units (ECUs), migrating to many-core processor within a single ECUs may even prove to be fully transparent—similar to changing the interior of a black box ECU. However, recertification will be necessary for the modified application. Depending on the criticality level, there is a considerable effort to be expected, as the software complexity rises significantly when a parallel execution is allowed.

Using multiple cores from a single application increases its performance and improves the overall processor utilization. Still, the speedup of parallel execution is governed by Amdahl’s law [8]. It distinguishes between code to be executed in parallel and code to be executed sequentially. In Amdahl’s model, the total execution time is the sum of the execution time of the parallel part and of the the sequential part. By adding more processors, the execution time of the parallel part can be reduced, because the work can be split among several processors. This reduces the total execution time and increases the performance. According to Amdahl’s law, an application containing a parallel code portion of 95% and a sequential code portion of 5%, the potential speedup on a processor with 8000 cores in comparison to a single-core processor with the same clock speed is at most 20. It becomes apparent that the performance of the sequential part dominates the speedup. Still, it is the single-thread performance that matters.

With regard to typical embedded applications, a parallel code ratio of 95% seems to be rather unlikely. Most applications are essentially based on processing state machines or they are used in feedback and control loops. In both cases, the potential for parallel execution is rather limited. In this environment, a famous quote by Brooks comes to mind: “The bearing of a child takes nine months, no matter how many women are assigned” [9]. In certain embedded applications, such as digital signal processing, *data parallel* execution instead of *function parallel* execution may be used to increase the parallel portion and to better exploit the parallel performance.

The migration to a single-application spreading over several cores does not have a significant impact on the safety assessment for the electronic control unit containing the Multicore processor. In the context of an entire system architecture, the effects are negligible as the requirements for the implemented function do not change. However, the effort for the development and quality assurance of an application consisting of several task executing in parallel may increase significantly.

2.3. Multiple Applications on Multiple Cores. The discussion of the previous approach showed that the parallelism offered

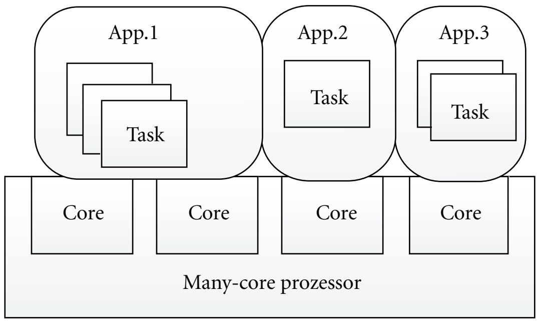


Figure 3: Integrate multiple, possibly parallelized applications.

within a single application will not be enough to exploit the potential of many-core processors. Consequently, higher levels of abstraction have to be analyzed for a potential parallel execution as well. The integration of multiple applications on a single many-core processor with each application potentially using multiple cores is a promising approach to harvest the parallel processing power of more than 100 cores on a single chip (see Figure 3).

While this approach offers the most benefits in terms of power and space savings, it is also very challenging throughout the engineering process. All applications need to be carefully analyzed with regard to their resource requirements and deployed to the new parallel hardware platform. Additionally, the entire system design—often consisting of multiple processors and communication networks—needs to be reevaluated as well. Going from two applications on separate single-core processors to the same applications running on a single processor increases the likelihood of interferences and potential fault propagation due to the use of shared resources. This effectively reduces the overall system reliability.

The consolidation of applications on fewer processors *naturally* has to be payed with an increased use of shared resources. In other words, it is the *intrinsic sharing of resources* that enable the much acclaimed benefits of many-core processors in the embedded domain in the first place.

Therefore, the impact on the certification effort is significant. As hardware-based isolation mechanisms between applications, such as dissimilar and incompatible network components, can no longer be used on a single processor, software-based isolation mechanisms have to be applied.

Fuchsen [10] describes necessary approaches to address the certification of multiple functions on Multicore processors in the avionics domain. In short, all "interference channels" have to be analyzed for the effects on the integrity of the isolation between applications. This is especially the case when applications with mixed criticality levels are to be integrated on the same processor.

With multiple applications executing in parallel and competing for the use of shared resources the visibility of the migration challenge being posed by this approach raises. Multiple development teams—at least one for each application—have to be coordinated in order to properly integrate all applications on the same processor *and* satisfy quality requirements, especially real-time behavior and safety considerations. When teams are even located in separate organizational entities, that is, supplier and original equipment manufacturer (OEM), the integration of multiple application creates another challenge that has to be addressed explicitly.

As it was described in this section, there are various approaches to adopt many-core processors in embedded systems. They differ in migration costs and performance benefits. Each approach has its right to exist, so in the end the choice which approach to take depends on the need for more performance. Fortunately, all approaches are going towards exploiting a parallel execution on some level, so that a stepwise adoption appears to be a sensible choice.

3. Domain-Specific Trends for Embedded Software

Adopting many-core processors by consolidating multiple functions on a single processor is a significant engineering challenge. However, recent developments in the field of domain-specific software architectures clearly demonstrate a trend to address this challenge and ease the migration process. Still it must be noted, that these developments are predominantly a result of the need to reduce development costs for software-intensive embedded systems by requiring interoperability and increasing competition among suppliers. In the following, the avionics domain and the automotive domain will be used as an example to show that the measures taken to support interoperability also help to simplify many-core migration.

3.1. Avionics: Integrated Modular Avionics (IMA) and ARINC 653. Among the dominating trends in the aviation electronics domain (avionics) are increasing functional requirements, the demand for more computer-based systems and the need for a shorter time to market. In the past, avionics systems were based on heterogeneous *federated architectures*. A distinctive feature of these architectures is that each function has its own independent computer and network system—"*one function-one computer*". A major advantage of this approach is that there is fault-containment *by design*—due to the heterogeneity of the components used.

Federated architectures, however, exhibit significant drawbacks. First, there are many dedicated resources required, which raises costs for procurement, cooling, maintenance and also increases the space, weight and power (SWaP) requirements. And second, aircraft functionality is artificially separated into independent components with no global control or synchronization, thus rendering the implementation of complex functionality challenging and costly.

Driven by the economic considerations described above, the avionics domain is slowly transitioning from federated avionics architectures to an Integrated Modular Avionics architecture (IMA) [11]. IMA describes the concept of computing modules with standardized components and interfaces to hardware and software. Thus IMA constitutes a logically centralized and shared computing platform, which is physically distributed on the aircraft to meet redundancy requirements. This transition is motivated by the expected benefits of hosting a variety of avionics functions on a single platform—"multifunction integration." More precisely, IMA was developed to reduce space, weight, and power requirements and furthermore to reduce maintenance and certification costs by allowing *incremental* certification of applications with mixed criticality levels [12].

Despite these advantages, the use of common components in IMA architectures significantly increases the probability of common cause errors affecting a variety of aircraft functions. Therefore, federated architectures are still considered to represent the benchmark for fault containment [13].

To consolidate multiple avionics functions on a *single-core processor* and address the challenges introduced by intransparent fault propagation on shared resources, IMA requires the use of *software partitioning*. This is a concept to achieve fault containment in software, independent of the underlying hardware platform.

On a shared computing platform, such as IMA, faults may propagate from one application to another through shared resources, such as a processor, memory, communication channels, or I/O devices. Partitioning isolates faults by means of access control and usage quota enforcement for resources in software.

In avionics safety standards, this is referred to as *partitioning in time and space* [14, page 9]. *Spatial* partitioning ensures that an application in one partition is unable to change private data of another. It also ensures that private devices of a partition, for example, actuators, cannot be used by an application from another partition. *Temporal* partitioning on the other hand guarantees that the timing characteristics and the quality of service of an application, such as a worst case execution time, are not affected by the execution of an application in another partition.

The standardized software API that allows partitions to communicate with the operating system and other partitions is specified in the ARINC 653 standard [15]. It is referred to as APplication EXecutive (APEX) and its underlying terminology is illustrated in Figure 4, which also depicts the relationship between *tasks*, *partitions*, and *applications*, which implement avionic functions.

3.2. Automotive: AUTOSAR. In modern automobiles, the electronic and electric architectures (E/E architectures) are constantly growing to address steadily increasing functional requirements. It is not uncommon for these architecture to encompass over 80 electronic control units (ECU) in a heavily distributed system. These E/E architectures constitute the "nervous system" of modern vehicles.

Traditionally, the design of these architectures focused on ECUs and their role in the entire system. With more and more ECUs being integrated into a single E/E architecture, keeping track of the development became increasingly challenging and costly. At the same time, reusability and exchangeability become important features of embedded software to reduce development costs and vendor lock-in effects.

These challenges lead to the development of AUTOSAR (AUTomotive Open System ARchitecture), (http://autosar.org/), which is an open and standardized automotive software architecture. It is jointly developed by automobile manufacturers, suppliers, and tool developers and tries to pave the way for a paradigm shift. Automotive software should be developed in a function-oriented way—in sharp contrast to the traditional an ineffective ECU-orientation.

AUTOSAR is based on the basic concept of having a standardized *virtual function-bus* (VFB), which facilitates the integration of software components in an existing architecture. This approach significantly streamlines the development of distributed software and improves the manageability of the entire architecture. The realization of an VFB requires an abstraction from the underlying hardware and its communication networks.

The AUTOSAR architecture (Source: http://autosar.org/gfx/AUTOSAR_ECU_softw_arch_b.jpg) is depicted in Figure 5. The architectural part of AUTOSAR specifies a complete *basic software* layer as an *integration platform* for hardware-independent AUTOSAR software applications. Furthermore, AUTOSAR contains a description of software application programming interfaces (APIs) for typical automotive applications. It also defines a complete methodology for the configuration of the basic software and the integration of AUTOSAR software on an ECU.

In Revision 4.0, the use of *software partitioning* was added to AUTOSAR [16, pages 101–107]. According to the standard, partitions are used as "error containment regions" [16, page 103]. Their consistency with regard to spatial aspects (e.g., memory access violation) or temporal aspects (e.g., time budget violation) has to be ensured by the platform.

3.3. How Do These Trends Affect Many-Core Adoption? What can be learned from the trends and developments described above? First of all, it becomes obvious that the embedded industry is actively working towards the design of a system (e.g., an ECU or an IMA board) with *multiple* vendors supplying *different* functions running on the *same* platform with some form of fault containment. This is referred to as a *multitenant processor platform* based on software-partitions and time-sharing of resources. Furthermore, the entire system design often comprises of several distributed processors

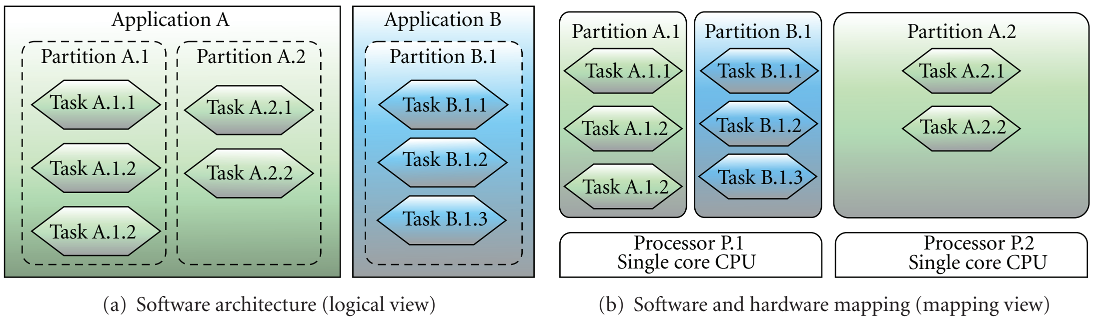

Application A
Partition A.1
Partition A.2
Task A.1.1
Task A.1.2
Task A.1.2
Task A.2.1
Task A.2.2
Application B
Partition B.1
Task B.1.1
Task B.1.2
Task B.1.3
Processor P.1 Single core CPU
Processor P.2 Single core CPU
(a) Software architecture (logical view)
(b) Software and hardware mapping (mapping view)

Figure 4: ARINC 653 terminology: an *application* is composed of one or more *partitions*. Each partition contains one or more *tasks*, which may execute concurrently. A partition is statically mapped onto a *processor*.

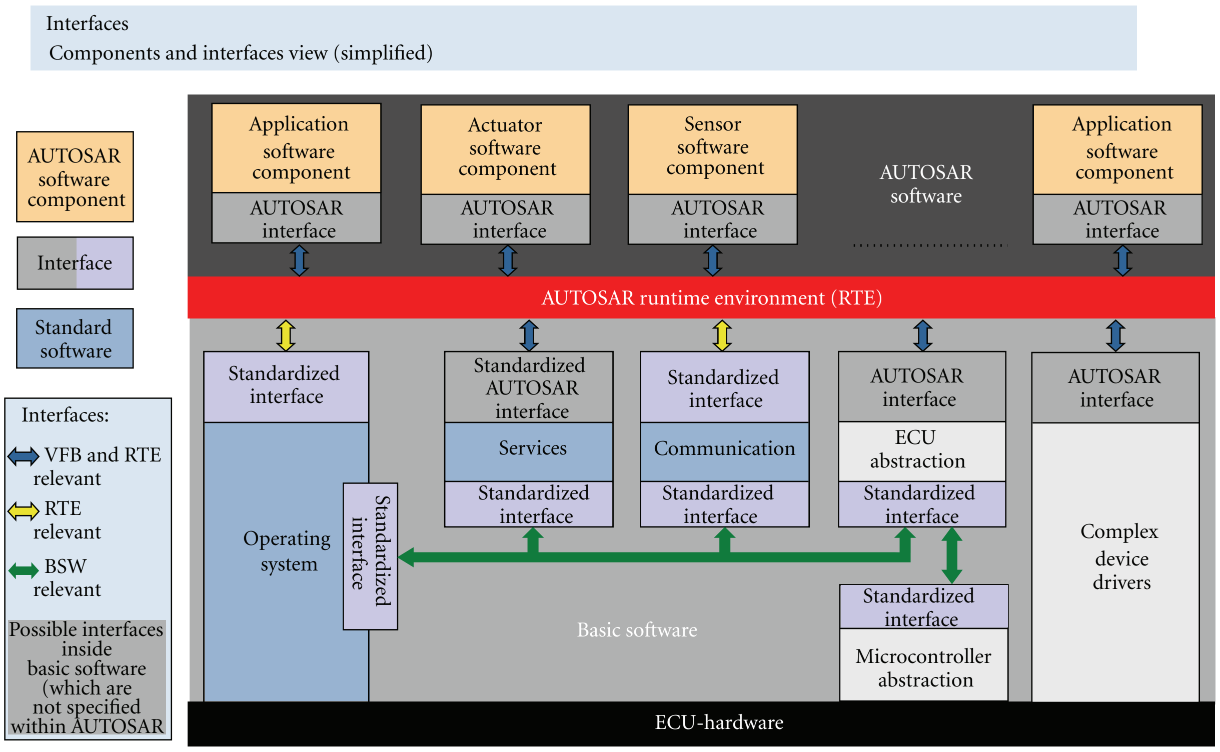

Interfaces
Components and interfaces view (simplified)
AUTOSAR software component
Interface
Standard software
Interfaces:
VFB and RTE relevant
RTE relevant
BSW relevant
Possible interfaces inside basic software (which are not specified within AUTOSAR
Application software component
Actuator software component
Sensor software component
AUTOSAR software
AUTOSAR interface
AUTOSAR runtime environment (RTE)
Standardized interface
Standardized AUTOSAR interface
Services
Communication
ECU abstraction
Operating system
Complex device drivers
Microcontroller abstraction
Basic software
ECU-hardware

Figure 5: The AUTOSAR ECU software architecture.

being connected to each via communication buses. Therefore, it is crucial to design a single multitenant processor platform with respect to other platforms in the same system.

Of course, a multitenant processor platform requires some form of standardization with regard to the hardware and/or operating system layer. Therefore, specific software application programming interfaces (APIs) are mandatory to acquire access to hardware resources. These APIs can also be used to restrict and control the way a function may access the hardware—thus enforcing isolation between separate applications with different criticality levels.

These changes in the engineering process have been initially motivated by the need for increased flexibility and for reduced vendor lock-in effects in order to cut development and maintenance costs. At the same time, multi-tenant system designs significantly facilitate a migration to many-core processors. By using only a standardized set of API calls to access system resources—instead of processor and operating system specific instruction sets—the software layer seems to be well prepared for a many-core processor environment—at least with regard to application-level parallelism.

This leads to a fundamental question for many-core adoption in software-intensive embedded systems: from the perspective of an application executing within a partition, where is the difference between having multiple partitions on a single-core platform with time-sharing and multiple partitions on a many-core platform allowing for true parallelism?

4. Software Requirements of Embedded Applications

To answer the previous question, it is necessary to take a look at the initial system requirements and compare these to the expected system behavior for each of the aforementioned architectures.

Generally speaking, there are *functional requirements* and *quality requirements*. Functional requirements specify the functionalities that the system has to provide to its users [17, page 15]. Quality requirements on the other hand, define desired attributes of the system, such as performance or reliability [17, pages 15-16]. Software-intensive embedded systems are often used to control and monitor real-world processes so their effectiveness is not only determined by the computational correctness. Instead, quality requirements are often the *primary drivers* of the entire system architecture [18, pages 155-156].

Functional requirements are usually satisfied by the application code within a partition, so in order to asses the equivalence of partitions with time-sharing and partitions executing in parallel, the focus needs to be shifted to quality requirements. For safety-critical embedded systems, quality requirements usually refer to *timing* or *safety* attributes. In some cases, *security* attributes play an important role as well. Timing attributes often comprise of *deadlines* for specific tasks, which have to be met under *all* circumstances. This leads to the requirement of a fully *deterministic* system behavior. The system engineer must to be able to know what the system is doing at all times during the entire lifetime of the system.

Safety-related attributes on the other hand are concerned with the assurance that no human lives are endangered when the system is running. These attributes can be addressed with *run-time monitoring* in addition to *independence* and *redundancy* of resources. These patterns provide protection, so that no single error leads to a system failure with human lives at stake. Safety measures also encompass designated safe states to which the system may transition to in case of a fault at run-time. Finding a safe state is not always trivial. While in some cases shutting down the power supply may be a transition to a safe state ("all offline"), it may be entirely different for a critical avionics component. Therefore, the analysis of safe states and possible transitions to these states is often a mandatory part of a safety assessment for a safety-critical embedded system.

How does a software partitioning approach of applications help to address timing and safety attributes of a system? In general terms, partitioning is a resource assignment to an application. In that sense, it is a vital component for the application and the entire system as its applications cannot execute properly if none or insufficient resources are assigned to it. If an application does not get enough processor time by the partition scheduler in the operation system, it may not be able to produce the results in time so that deadlines may be missed and timing attributes are in jeopardy.

A correct resource assignment is also very important to satisfy safety requirements. If an incorrect assignment creates an unintended interference channel between two separate partitions [10], the isolation between partitions in order to prevent fault propagation is flawed and the integrity of the entire system is endangered.

These quality requirements constitute the many-core migration challenge and lead to the difference between having multiple partitions on a single-core platform with time-sharing and multiple partitions on a many-core platform allowing for true parallelism. With time-sharing, each partition has an exclusive access to all hardware resources during its time slice. The only interruption is to be expected from the operating scheduler when the end of a time slice has been reached. Therefore, the entire system becomes very deterministic as there are no unforeseen waiting times to be expected due to resource contention at run-time. Every application executes as if it were the only application executing on that platform.

The most significant difference to the execution on the many-core platform becomes apparent when a parallel hardware platform effectively invalidates the "exclusive access" assumption. Two applications running in parallel on separate partitions are suddenly competing for resources, which affects the quality attribute of the entire systems. The performance may degrade significantly due to cache thrashing. Concurrent access to resources, such as network interfaces, may lead to nondeterministic waiting times—thus risking the deterministic system behavior. Generally speaking, every processor resource (caches, on-chip interconnect, memory controller, ...), which is now shared between partitions as a result of a parallel execution, constitutes a potential channel for interferences.

How does this affect the software engineering approach? Without parallel execution, applications executing within a partition could be developed and integrated on the platform with only very little knowledge about other partitions. A partition schedule simply defined when to start which partition. With parallel execution, that is, several partitions potentially running at the same time, their integration becomes more challenging. Every partition needs to define, which resources are needed for execution. This encompasses not only processor time, but also on-chip interconnect links, caches, memory controllers, interrupt controllers, CAN bus controllers, and so on—essentially every resource which may be shared and concurrently accessed at runtime. In order to integrate several partitions on a many-core processor platform, these resources requirements need to be aligned, so that no resource is overbooked and resources contention is eliminated.

Also safety analysis for a multi-tenant system becomes more challenging. A single fault within the resources used by one partition may then affect the execution of another partition as well. Shutting down the entire control unit may help to reach a safe state, but on the other hand, this will affect all functions deployed on this processes. In some situations, a fine-grained approach on the partition level may be applicable. If a fault manifests itself in a commonly used resource, all partitions using this resource have to transition to their designated safe state.

Generally speaking, on a parallel hardware platform the resource requirements of each partition need to be compared to the requirements of *every* other partition on the same platform in order to satisfy quality requirements. This *integration effort* increases with an increasing number of partitions and an increased use of shared resources. Therefore, it is safe to assume, that the more the embedded domain embraces many-core architectures, the more challenging the integration will become and the more it will pay off to look at efficient ways to address this integration challenge.

5. Traditional Integration Approaches

The last section described the major challenges of a multifunction integration in order to tap the full performance potential of a many-core hardware platform. Essentially, there are three distinct obstacles to be address during the integration:

(i) identification of *all* shared resources,

(ii) providing sufficient isolation between partitions to prevent fault propagation via shared resources,

(iii) creating and managing an assignment from resources to partitions, so that quality requirements—especially timing—can be satisfied.

The first obstacle can be addressed by a thorough analysis of all applications and the hardware platform. Therefore, it is often necessary to acquire detailed documentation about the processor architecture. With all shared resources being successfully identified, several well-established approaches can be used to ensure isolation. Rushby [13] gives a detailed overview about static and dynamic isolation techniques in the avionics and aerospace domain. In addition to the approaches mentioned by Rushby, highly reliable *hypervisors* can also be used to isolate separate partitions from each other (cf. [19–21]).

The third obstacle turns out to be the most challenging one—especially on many-core processors. This is the case as traditional approaches for an integration of software applications has reached its limits when quality requirements, such as timing properties have to be met. The current practice in the embedded industry can often be described with "timing by accident."

The deployment of partitions onto processors and their operating system schedule—that is, the assignment of resources—is often initially derived from previous project in a similar context. Changes to the resource assignment are realized based on engineering experience and estimations about resource consumption. The initial resource assignment is subsequently analyzed for potentials flaws (see the tool suites from INCHRON (http://www.inchron.com/), TA SIMULATOR (http://www.timing-architects.com/products.html), or SYMTA VISION (http://www.symtavision.com/)). These real-time simulators help to determine whether the priority assignment to tasks and the chosen scheduling algorithm is sufficient to satisfy all real-time requirements. However, this analysis is of NP-complexity and for a larger task set it may take up a significant amount of time in the engineering process.

For low or noncritical applications, it may be sufficient to focus on the *average* case. This approach is often combined with an overprovisioning of resources based on the assumption, that if the worst case scenario occurs, these spare resources will help to mitigate the effects.

The "timing by accident" methodology clearly reaches its limit when a fully deterministic system behavior for highly critical and complex systems is required. In this case, every access to system resources, for example, processor or network interfaces, has to be coordinated based on knowledge at *design time*. Providing spare resources is no longer a viable option, as no deviations from the static operating system schedule are tolerated. The entire system behavior based on the execution patterns of all partitions on a processor has to be defined at design time.

This information is often not gathered during the engineering process and therefore seldom available during the integration. Nevertheless, the process of designing a schedule satisfying all timing requirements of all partitions is very cumbersome, but at the same time a very sensitive matter. Creating a static schedule thus often takes several person months for reasonably complex systems. The complexity of this specific scheduling problem arises not only from the challenge to select tasks so that all deadline are met. Instead the schedule has to mimic the entire system behavior with regard to timing, for example, context switches take a certain amount of time or writing to buffers after a computation finished may take some additional time. It also has to incorporate intra- and interapplication relations, which may affect the order of execution. Furthermore, the execution of some applications may be split into several slices (offline determined preemption), which affects the amount of context switches and thus the timing. And last but not least, it is often necessary to optimize the schedule according to a certain criteria. This is especially beneficial when hypervisor-based systems with a two-level scheduling approaches (for partitions and applications) need be addressed. For these systems, it is a common goal to optimize the schedule for a minimal amount of partition switches to minimize the overhead introduced by the hypervisor. A "timing by accident" approach in which timing is addressed after all functional requirements have been satisfied is not sufficient anymore. Timing is a cross-cutting concern for the entire system design. Therefore, it needs to be addressed and incorporated throughout the entire engineering process [22].

And timing requirements are only a part of the quality requirements, which need to be addressed explicitly for a multi-function integration. The same is true for quality requirements regarding safety. Partitions cannot be freely deployed on the entire system architecture spanning over several processors. Often pairs of partitions need to be mapped on fully *dissimilar* hardware components to avoid a failure due to undetected design errors in the hardware. In some cases with lower criticality requirements, it is sufficient to map partitions on *redundant* hardware nodes. This task can still be accomplished manually up to a problem size of about 20 partitions on 10–15 processors. However, in complex avionics systems, there will be about 100 processors and about 1000 individual functions to be managed in the foreseeable future. A similar trend to increasing number of electronic control units and software components is evident in automotive systems, where more than 80 ECUs in current high-end automobiles are not uncommon.

With many-core processors, the engineers face a significant increase in complexity, which cannot be handled manually in an efficient way. The complexity of the hardware increases as there are significantly more shared resources. At the same time, there will be more and more partitions to be migrated to a single processor. With more partitions being integrated on the same platform, more development teams—which are most likely spatially spread over several companies—need to be synchronized. This poses an interesting nontechnical management challenge all by itself. At the same time, the development processes will have to become more agile in order to address the need for a shorter time to market and shorter product evolution cycles.

Generally speaking, the ability to efficiently realize a *temporal* and *spatial* assignment of resources to partitions determines the degree to which important quality attributes can be satisfied. In the end, it also determines the level to which software components can be integrated on the same hardware processor platform. A multi-function integration on a many-core system cannot be done efficiently with the aforementioned traditional approaches. So the research question remains: how to tackle this integration challenge which affects the degree to which the performance of many-core processors can be exploited in software-intensive embedded systems?

A few years ago, the development of complex safety critical software—not entire systems—was confronted with a similar challenge. The application engineers were confronted with a very high system complexity. They had to satisfy challenging functional requirements and of course, there was zero tolerance for defects. This challenge was addressed with a new approach: software development based on *formal methods*. The underlying idea was based on the realization that the correctness of reasonably complex software could no longer be assured by analyzing all possible execution paths. The correctness of a system could not be proven by simply observing its behavior. There were simply too many execution paths, so that each path could not be analyzed and tested without significantly exceeding the project budget.

Instead of analyzing the exhibited behavior of the software after it has been built, this new approach focuses on earlier stages in the development: requirements engineering, design, and implementation. During these stages, the software is *constructed* based on a specific rule-set, which derives the components from formalized requirements—hence its name "*correctness by construction.*" This approach has been successfully applied in several case studies and lead to fewer bugs and lower development costs [23–25]. The design space of the entire system is restricted by a construction rule-set, so that a significant amount of implementation errors are circumvented.

The challenge of a *multi-function with mixed-criticality integration on many-core systems* could be properly addressed by a new engineering approach based on similar tactics. The complexity and volatility of the design space simply exceeds the capabilities of traditional integration approaches. Of course, existing "construction rules" for the development of software cannot be used here, because the challenge affects the integration and not the development. So the question is, how to adapt this approach and use it for an *efficient construction* of a resource assignment in a multi-function integration scenario?

6. Integration Based on "Correctness by Construction"

This engineering principle has been pioneered by Chapman and Hall [23, 24] for the development of "high integrity software" [24]. Although it applies to the entire life cycle of a software components, it focuses on the programming aspect. Its goal can be summarized as "*argue the correctness of the software in terms of the manner in which it has been produced ("by construction") rather than just by observing operational behavior*" [24, page 1].

How does this principle help for the multi-function integration? The benefits are desirable, but the adoption of a similar approach for the integration is still the subject of applied research. Up to this point, the following promising strategies have been adapted to the integration challenge based on similar strategies for the prevention and removal of defects in software [24].

Write Right. When quality requirements need to be addressed during the integration, they need to be explicitly expressed and formalized, so that there is no ambiguity about their meaning. Specifications should be distinguished regarding the resources supplied from the hardware and the resource demand from the applications. Typically, these specifications need to contain:

(i) *spatial* parameters (e.g., memory usage, network interface usage, . . .),

(ii) *temporal* parameters (e.g., access patterns, tolerated jitter, . . .),

(iii) *safety-relevant* parameters (e.g., vendor information, architectural types, . . .).

In addition, the hardware architecture of the system with its processing nodes and communication channels have to be modeled as well as the software architecture with all of its applications and their communication intensity.

Step, Do Not Leap. When constructing a resource assignment, typically the questions of "where" and "when" have to be addressed. *Where* does an application get executed—on which resources—and when does it get executed on these resources—its scheduling pattern. In an incremental process, with the question of where ("*mapping*") should be addressed before the question of when ("*scheduling*"). Each constructed mapping should be validated independently whether all spatial and safety-relevant parameters are correctly addressed. In a second step, a schedule gets constructed based on the mapping from the previous step. If a schedule cannot be constructed, feedback should be provided so that the initial mapping can be efficiently adapted accordingly.

Check Here before Going There. Every resource assignment should be checked whether its demand exceeds the resource supply. The distinction between *additive* and *exclusive* resources and the determination of a *resource capacity* is a crucial prerequisite for this validation. While exclusive resources can be acquired by only one application, additive resources can be used by more than one application until their capacity has been reached. However, the differentiation is not as simple as it appears and depends on other parameters as well. For instance, a network interface may typically be used by several applications so it may be categorized as being *additive*. On the other hand, are all applications allowed to access the interface *at the same time*? Do all applications need to have been developed according to same criticality level to avoid an unintended interference channel?

Screws: Use a Screwdriver, Not a Hammer. For the construction of mappings and schedules, a variety of algorithmic approaches in combination with special heuristics can be applied to tackle the challenge of NP-completeness. There is no omnipotent tool being applicable for all integration challenges. Every approach has its advantages in the construction process, but only the combination of all tools significantly boosts the efficiency.

As stated at the beginning of this section, the application of correctness by construction for a multi-function integration on many-core processors is still subject to active research. The next section will give an overview about case studies in which this principle was evaluated in practice.

7. Case Studies and Current Research

A model-based approach for the construction of static operating system schedules was used as an initial study of feasibility [26]. It is based on the underlying assumption that predictable and entirely deterministic real-time behavior for a system—which relies only on periodical tasks with known periods—can be achieved on multi- or many-core processors with static schedules if the following information (or at least an estimation) is available at design time (or configuration time):

(1) *timing characteristics*, for example, worst case execution time (WCET), of all applications and their processes on the underlying hardware platform

(2) *scheduling dependencies* between applications and processes

(3) *off-chip resource* usage patterns of all applications, such as busses or other significant external devices.

All conflicts, which may appear at run-time and lead to unpredictable timing, are resolved statically at design time. Our approach aims to optimize the schedule beforehand rather than troubleshooting afterwards.

A scheduling tool—called *PRECISION PRO*—was developed as a prototype. It automatically generates a valid static schedule for high-level timing characteristics of a given set of applications and a given hardware architecture. In addition to generating solutions, that is, static schedules, for the underlying NP-hard problem, the user is also given the opportunity to adjust the generated schedules to specific needs and purposes. PRECISION PRO ensures that no hard constraints are violated during the adjustment process. For the sake of simplicity, the input is currently modeled in a textual notation similar to Prolog clauses.

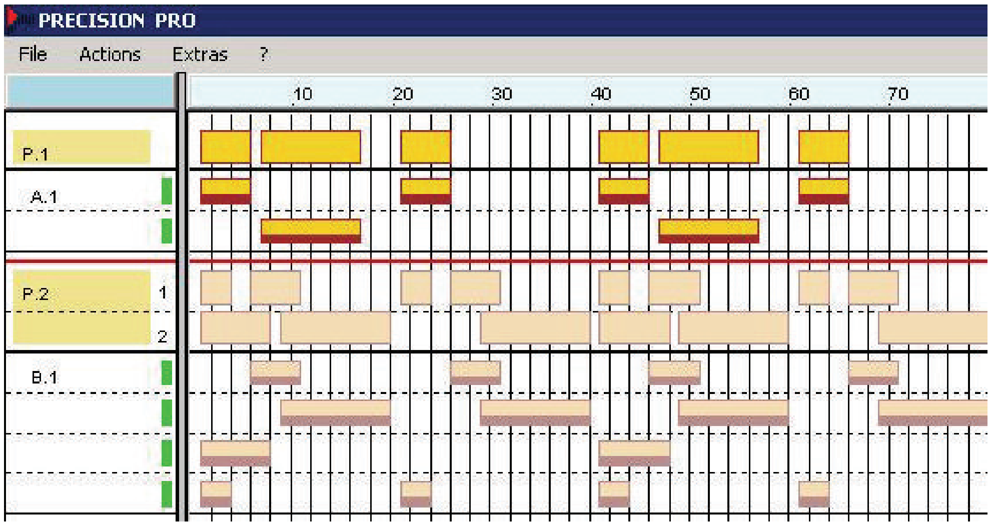
With the help of heuristic approaches [26], PRECISION PRO tries to construct a schedule for the constraints defined in the input model. The input model and the problem specifications have been developed to suit the needs of a safety critical domain: avionics. Therefore, all execution times for software tasks in PRECISION PRO are based on a worst-case analysis. All tasks are executed at run-time with the help of a time-triggered dispatcher. This approach is often overly pessimistic, but still a hard requirement for systems in accordance to avionic safety levels A, B, C, and D.

However, there are other scheduling approaches aimed at optimizing the resource utilization despite having a fixed schedule. On approach is called “slack scheduling.” It assigns processor cycles, which are unused to special applications. Another approach to improve resource utilization is based on specifying lower and upper bounds on the execution time. With the help of precedence constraints, a robust schedule can also be constructed, which avoids possible scheduling anomalies [27].

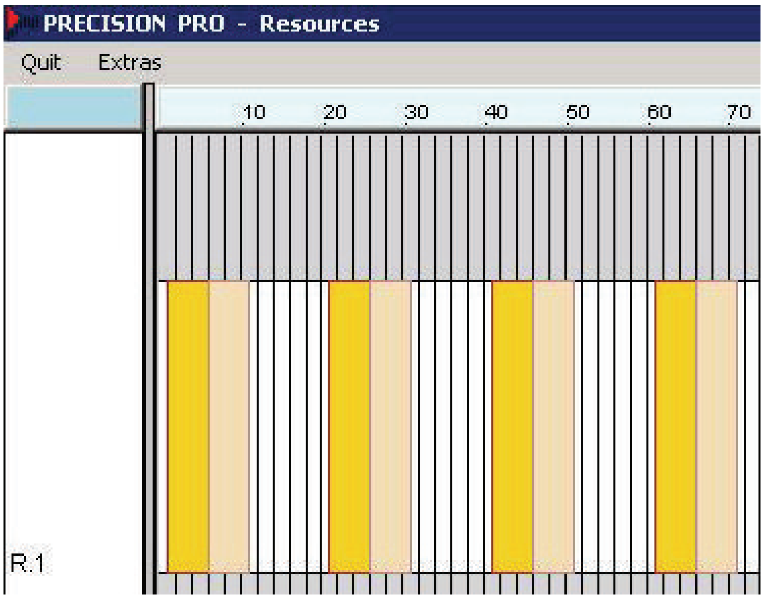
The complexity of the scheduling problem addressed with PRECISION PRO does not only come from determining a fixed schedule for nonpreemptive tasks so that deadlines are met. Instead it tries to capture the timing behavior of the entire systems and let the system engineer construct the desired behavior. Therefore, tasks may be preempted by other tasks if necessary. Tolerated jitter can be specified for each application. Relations between tasks executing on different processors can be incorporated into the schedule as well. Furthermore, the modeled system can also contain other resources despite processing time. These resource may be used exclusively, for example, network bandwidth, or cumulatively, for example, an actuator, by an application. During the construction of a schedule, it is guaranteed that the capacity of these external resources is not exceeded.

Due to the complexity of possible constraint specifications and the use of heuristics, PRECISION PRO focuses on “safe acceptance” for static schedules, that is, a static schedule for a feasible task system may not be found within time, but if a schedule was found, it is guaranteed to satisfy all requirements. Although the worst-case run-time for a complete search is exponential, the heuristics allow for affordable run times. For a real world example consisting of three Multicore processors, about 40 applications, a scheduling hyperperiod of 2000 ms and a 1 ms timeslot, it requires about 800 ms for generating the model from the input data and about 250 ms for searching a valid solution, that is, a feasible static schedule (executed on a virtual machine running Microsoft WindowsXP on a 2.4 GHz dual-core laptop with 4 GB RAM).

The generated schedule and the mapping onto processors and cores is presented in a graphical user interface (see Figure 6(a)). For each processing element, all processes that are executed at a given time within the scheduling period



(a) Generated schedule for an example scenario with two processors (P.1 and P.2)



(b) Resource usage

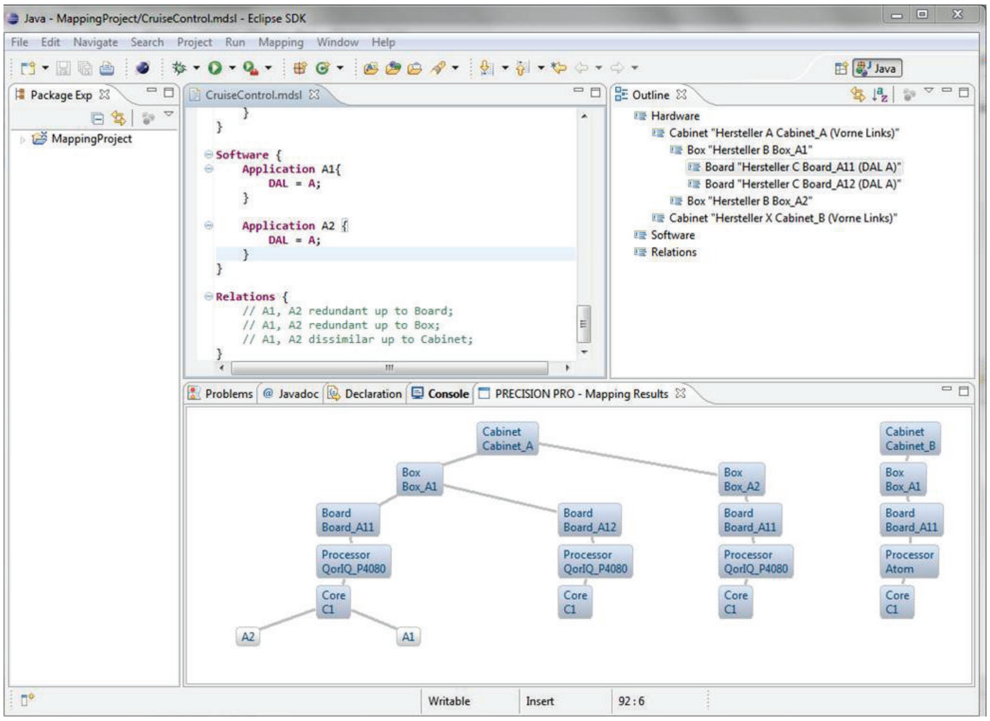
Figure 6: PRECISION PRO Output—Figure 6(a) contains the schedules as it was generated for all processes and their mapping onto processors and cores. Figure 6(b) depicts the resource usage of a fictional off-chip resource.

are graphically represented by colored bars in a single row. Relations between processes are indicated by solid lines. Additionally, the usage of external resources is presented in a special *Resource Window* (see Figure 6(b)).

All relevant scheduling information can be exported into an operating system independent ASCII file. This file is usually transformed to satisfy formatting requirements of specific operating systems. After being approved by the certification authority, it is used in the final configuration of the scheduling component in the operating system.

Dynamic scheduling approaches based on fixed or variable priorities represent another common approach for real-time scheduling to allow for exclusive access to the hardware. Although this approach improves resource utilization, it also hides relations between tasks and scheduling constraints by assigning priorities. Priorities express a relationship to other tasks, but they do not capture the intent of the system engineer. Furthermore, with multi- and many-core processors, tasks with different priorities may still be executed at the same time as there may be plenty of cores available. This may not be a problem for systems with less stringent predictability requirements. However, it quickly becomes an issue for safety-critical systems in which predictability and determinism matters. "Correctness-by-Construction" in combination with static scheduling is about making the design decisions explicit, so that they can be reused for other hardware platforms without compromising their integrity.

Generating static schedules for safety critical Multicore systems is only the first step in improving the development



```
        }
    }

    Software {
        Application A1{
            DAL = A;
        }

        Application A2 {
            DAL = A;
        }
    }

    Relations {
        // A1, A2 redundant up to Board;
        // A1, A2 redundant up to Box;
        // A1, A2 dissimilar up to Cabinet;
    }
```

```
Hardware
    Cabinet "Hersteller A Cabinet_A (Vorne Links)"
        Box "Hersteller B Box_A1"
            Board "Hersteller C Board_A11 (DAL A)"
            Board "Hersteller C Board_A12 (DAL A)"
        Box "Hersteller B Box_A2"
    Cabinet "Hersteller X Cabinet_B (Vorne Links)"
Software
Relations
```

Figure 7: Construction of mappings from applications onto processors and cores.

processes and facilitating certification. In addition to PRECISION PRO, two supplementary components are currently in development: a *validator* and a *mapper*.

Validator. Due to the importance of an operating system schedule for a safety critical system, special attention has to be given to its verification during certification and official approval—especially, since it was automatically constructed by a software tool. Although PRECISION PRO will—by design—only construct schedules satisfying all constraints, an additional verification step is still necessary, because there may be undetected software bugs in the underlying libraries used by PRECISION PRO, which may eventually lead to the construction of erroneous schedules.

Usually, there are two options for a system engineer who wishes to use tools to automate development processes: either the software tool has to be *qualified*, which is a very intricate and costly task, or the result has to be verified with another, *dissimilar* verification tool (see [14], pages 59–62). (The verification tool does not need to be qualified, as its output is not directly used in the final product.) Here the second option was chosen, so that a special *validator* for static Multicore schedules is currently in development. It uses the output of PRECISION PRO and creates a formal model, which is a deterministic transition system based on time discretization. This model is used for applying model-checking techniques with the linear temporal logic (LTL) model checker UPPAAL (http://www.uppaal.org/).

Model properties, which are checked by the validator—*never claims*, are essentially LTL formulae, which are derived from the original input model of PRECISION PRO. If all properties were successfully checked, that is, no never claim turned out to be true, the model of the static schedule satisfies all formalized software requirements and the construction was correctly done.

Mapper. Previously, the challenge of mapping from software components, that is, partitions and processes, onto processors and their cores was introduced. Currently, this is often done manually, which may be sufficient for systems with only a few processors. However, it is certainly not a viable approach for complex systems comprising of up to 100 processors, especially, since mapping for safety critical systems does not solely depend on achieving scheduleability, but also on other safety criteria, such as *redundancy*, *dissimilarity* and *independence*.

For instance, the mapping of an application comprising of two *redundant* partitions should not allow these identical partitions to be mapped on the same core or on the same processor. This would clearly lead to a violation of safety requirements, because the underlying hardware usually does not offer *enough* redundancy for the designated assurance level.

It is also common for safety critical applications to comprise of partitions which implement the same functionality, but in a dissimilar fashion. Depending on the criticality level, these partitions may need to be mapped onto *dissimilar* processors and configured to use *dissimilar* communication channels to prevent design errors.

As a result of research conducted in the avionics domain, it became clear that safety related constraints, for example, *"partition A and partition B have to be mapped onto dissimilar/redundant hardware,"* require additional *qualification*. Redundancy and dissimilarity have to be expressed in relation to certain aspects of the underlying hardware. For instance, partition *A* and partition *B* have to be mapped on processors with redundant power supplies and dissimilar communication channels.

By acknowledging the fact that the design of safety-critical embedded systems has to follow complex safety restrictions and regulations, but at the same time it also needs to minimize costs and reduce space, weight and power requirements, a *mapper* is currently in development, which *constructs* feasible mappings that satisfy safety constraints (see Figure 7). These mapping are then used in PRECISION PRO to generate a valid schedule. If such a schedule cannot be found, the *mapper* is requested to modify the initial mapping. PRECISION PRO will help guiding the mapping process by providing elaborate information regarding the degree of capacity utilization for each processor. The mapper may then choose to remove partitions from overutilized processors, so that a schedule can be constructed and all mapping-related safety requirements are met.

8. Conclusions and Discussion

Many-core processors can be used to significantly improve the value of software-intensive systems. A small power envelope in conjunction with unprecedented performance levels—at least in the embedded domain—pave the way for cyber physical systems. Having many, but relatively simple cores requires the exploitation of parallelism on all levels, especially on the application layer and the thread layer. This approach constitutes a significant prerequisite in tapping the full potential of a massively parallel processor. At the same time, this leads to a multi-tenant situation, when multiple functions from different vendors are to be integrated on the same chip. Trends and standardization efforts in key domains, that is, avionics and automotive, already address these challenges arising from the functional requirements. The integrated modular avionics in conjunction with ARINC 653 in avionics and AUTOSAR in the automotive domain provide a standardized abstraction layer, so that software components can be developed relatively independent from the specifics of the underlying processor hardware.

While this helps address the functional correctness of integrating several applications on a many-core processor, quality requirements—especially timing and safety—are not sufficiently addressed. These become especially apparent during the integration. Suddenly applications are competing for resources, which may get congested leading to unpredictable waiting times and jeopardizing the predictable system behavior.

Software partitioning is a well-established concept in operating systems to prevent fault propagation by restricting the access to resources. Partitioning is essentially an assignment from resources to applications. With many-core processors, this assignment can no longer be created manually as there are simply too many resources and too many applications to balance.

"Correctness by construction" is an engineering principle that has been successfully applied in the implementation of highly reliable software. Its fundamentals can also be applied in the context of multi-function integration on many-core. Formalized requirements and a resource assignment as a result of a *construction process* provide an efficient engineering approach to address the many-core challenge in software-intensive embedded systems.

References

[1] H. Sutter, "The free lunch is over: a fundamental turn toward concurrency in software," *Dr. Dobb's Journal*, vol. 30, no. 3, pp. 202–210, 2005.

[2] A. A. Vajda, *Programming Many-Core Chips*, Springer, 2011.

[3] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[4] E. A. Lee, Cyber-Physical Systems—Are Computing Foundations Adequate?, 2006.

[5] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*, Lee & Seshia, 2011, http://www.lulu.com/.

[6] K. Asanovic, R. Bodik, B. C. Catanzaro et al., The Landscape of Parallel Computing Research: A view from Berkeley, 2006.

[7] S. Borkar, "Thousand core chips—a technology perspective," in *Proceedings of the 44th ACM/IEEE Design Automation Conference (DAC '07)*, pp. 746–749, June 2007.

[8] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the Spring Joint Computer Conference*, pp. 483–485, ACM, April 1967.

[9] F. P. Brooks Jr., *The Mythical Man-Month*, Addison-Wesley Longman Publishing, Boston, Mass, USA, 1995.

[10] R. Fuchsen, "How to address certification for multi-core based IMA platforms: current status and potential solutions," in *Proceedings of the 29th IEEE/AIAA Digital Avionics Systems Conference: Improving Our Environment through Green Avionics and ATM Solutions (DASC '10)*, pp. 5.E.31–5.E.311, October 2010.

[11] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," in *Proceedings of the 26th IEEE/AIAA Digital Avionics Systems Conference—4-Dimensional Trajectory-Based Operaions: Impact on Future Avionics and Systems (DASC '07)*, pp. 2.A.1-1–2.A.1-10, October 2007.

[12] RTCA, Integrated Modular Architecture—Development Guidance and Certification Considerations, 2005.

[13] J. Rushby, "Partitioning for avionics architectures: requirements, mechanisms, and assurance," NASA Contractor Report CR-1999-209347, NASA Langley Research Center, 1999, Also to be issued by the FAA.

[14] RTCA, Software Considerations in Airborne Systems and Equipment Certification, 1994.

[15] ARINC, ARINC Specification 653P1-2: Avionics Application Software Standard Interface Part 1—Required Services, 2005.

[16] AUTOSAR, "Layered Software Architecture," 2010, http://autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

[17] K. Pohl, *Requirements Engineering: Grundlagen, Prinzipien,Techniken*, Dpunkt.Verlag GmbH, 2nd edition, 2008.

[18] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley Longman Publishing, Boston, Mass, USA, 1998.

[19] R. Rose, "Survey of system virtualization techniques," Tech. Rep., Oregon State University (OSU), 2004.

[20] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[21] R. Hilbrich and M. Gerlach, "Virtualisierung bei Eingebetteten Multicore Systemen: Integration und Isolation sicherheitskritischer Funktionen," in *INFORMATIK 2011—Informatik Schafft Communities*, H. U. Heiß, P. Pepper, H. Schlingloff, and J. Schneider, Eds., vol. 192 of *Lecture Notes in Informatics*, Springer, 2011.

[22] E. A. Lee, "Computing needs time," *Communications of the ACM*, vol. 52, no. 5, pp. 70–79, 2009.

[23] A. Hall and R. Chapman, "Correctness by construction: developing a commercial secure system," *IEEE Software*, vol. 19, no. 1, pp. 18–25, 2002.

[24] R. Chapman, "Correctness by construction: a manifesto for high integrity software," in *Proceedings of the 10th Australian workshop on Safety critical systems and software (SCS '05)*, vol. 55, pp. 43–46, Australian Computer Society, Darlinghurst, Australia, 2006.

[25] S. Resmerita, K. Butts, P. Derler, A. Naderlinger, and W. Pree, "Migration of legacy software towards correct-by-construction timing behavior," *Proceedings of the 16th Monterey Conference on Foundations of Computer Software: Modeling, Development, and Verification of Adaptive Systems (FOCS '10)*, Springer, Berlin, Germany, pp. 55–76, 2011.

[26] R. Hilbrich and H. J. Goltz, "Model-based generation of static schedules for safety critical multi-core systems in the avionics domain," in *Proceedings of the 4th ACM International Workshop on Multicore Software Engineering (IWMSE '11)*, pp. 9–16, New York, NY, USA, May 2011.

[27] M. Lombardi, M. Milano, and L. Benini, "Robust non-preemptive hard real-time scheduling for clustered multicore platforms," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '09)*, pp. 803–808, April 2009.