Multi/Many-Core Programming: Where are we Standing?

Jeronimo Castrillon
TU Dresden, jeronimo.castrillon@tu-dresden.de

Lothar Thiele, Lars Schorr
ETH Zurich, thiele@tik.ee.ethz.ch

Weihua Sheng
Silexica Software Solutions GmbH, sheng@silexica.com

Ben Juurlink, Mauricio Alvarez-Mesa, Angela Pohl
TU Berlin, b.juurlink@tu-berlin.de

Ralph Jessenberger
BeOne Frankfurt GmbH, ralph.jessenberger@beone-group.com

Victor Reyes
Synopsys Inc., victor.reyes@synopsys.com

Rainer Leupers
RWTH Aachen University, leupers@ice.rwth-aachen.de

***Abstract*—This paper presents different views exposed in a special session on the current standing of programming and design tools for multi and manycores in the embedded domain. After approximately ten years of the advent of multicore architectures, we take a look at state-of-the-art and trends in model-based programming methodologies from an academic point of view. This view is contrasted with early experiences in transferring multicore compiler research to industry, and complemented with a critical view on the performance gap introduced by compilers for complex architectures. Today, multicores permeate new applications domains, creating new requirements and forcing researchers to rethink some underlying assumptions. This paper exposes the requirements of one such new domain, namely automotive. Applications in this domain require not only programming tools that comply to standards (e.g., ISO 26262) but also tools for high-level simulation, performance analysis and debugging. In this context, we discuss the role of virtual platforms in managing complexity of hardware-software interactions and accelerating the design of multicore systems for automotive applications.**

I. Introduction

Multi-processor Systems on Chip (SoC) have been in wide use for about ten years now. Today, there are many examples of power-efficient architectures suitable for embedded applications, such as the STMicroelectronics STHorm platform [1], the He-P2012 platform [2] which extends STHorm with hardware accelerators and the KALRAY MPPA manycore processor [3]. In the general purpose processing (GPP) and high-performance computing (HPC) communities many new standards and libraries for programming these parallel machines have appeared, including OpenMP, OpenACC, OpenCL and Intel Threading Building Blocks (see for example [4]). The embedded community, with its more application-specific nature, has been keener on model-based approaches to parallel programming [5], [6]. In particular, we have seen an increasingly growing community working on different flavors of dataflow parallel programming models [7]. This paper looks at programming methodologies from the academic community, reflects about current industrial adoption and presents requirements from the automotive domain.

It is fair to say that the academic community has had considerable success at devising methods to program heterogeneous embedded multi-processor platforms. This paper starts by presenting a survey of model-based approaches, including new challenges for real-time systems, in Section II. In Section III, we then look at particular programming methodologies and tools for today and upcoming heterogeneous systems. In the last years, we have seen how the transfer of some of these methods to industry has intensified, possibly indicating an inflection point at which hardware complexity renders manual approaches impractical. However, the relative low success-rate of these transfers invites reflection. Two key aspects that hinder industrial adoption of academic methodologies are discussed in Section IV and Section V, namely (*i*) the lack of coverage of new industrial requirements and (*ii*) the sometimes low performance of automatically generated code. In particular, Section IV describes early experiences in commercializing programming tools for multicores, focusing on industrial requirements not addressed by academic approaches. Section V, in turn, discusses the performance gap between automatic and manual approaches for complex architectures. The section focuses on single processors within a multicore with support for SIMD instructions, and proposes programming extensions to expose complex architectural features to low-level programmers.

Section VI and Section VII focus on applications from the automotive domain —a domain in which multicore architectures started to gain interest in the last years. In particular, Section VI discusses the challenges that come with multicores for infotainment and safety critical automotive applications. These challenges together with automotive standards introduce new requirements for programming tools and methodologies. Thereafter, Section VII elaborates on how virtual prototypes can help to design automotive systems, by providing a simulation vehicle to not only validate functionality but also analyze performance bottlenecks and debug code in a non-intrusive way. Finally, we summarize and synthesize the different views presented throughout this paper in Section VIII.

II. Model-based System Design

A. Context and Requirements

Many-core SoCs are of increasing significance in the domain of high-performance embedded computing systems where high performance requirements meet stringent timing constraints. However, the available computing power does not necessarily translate into high performance as many-core SoCs may face thermal issues, they may not react to runtime variations of applications and environment, they may not leverage the peculiarities of the individual processing cores, and they may suffer from task interferences on shared resources such as memory and communication systems.

Embedded applications are often centered around the processing of data streams. Examples include real-time speech recognition, embedded computer vision, and all kinds of advanced features that are nowadays ubiquitous in cars as, for instance, the anti-lock braking system, electronic stability control, or adaptive cruise control. Moreover, with autonomous cars on the horizon, embedded systems must soon be able to solve a multitude of computer vision and artificial intelligent problems in real-time. As the functionality of the system can change over time, embedded systems must be able to deal with complex dynamic interactions between the applications. With each application having its own real-time constraints and quality of service requirements, the system designer has to make sure that each application meets its individual requirements independently of other applications. In the following, some recent approaches to utilize the potential of new many-core SoCs for these applications will be described more closely.

B. Model-Based Design and Challenges

A recently advocated strategy to design reliable and efficient systems is to restrict the application to a certain (high-level) programming model, thereby enabling the automatic generation and optimization of the platform-dependent implementation by leveraging properties of the programming model. Such a strategy is commonly referred to as model-driven software development [6].

The Y-chart paradigm [8] is a design flow model that implements the ideas of a model-driven software development. It is based on the orthogonalization of concerns, which proposes to master the complexity of embedded systems by separating parts of the design process. The basic idea of the Y-chart paradigm, schematically outlined in Figure 1, is to separate the specification of the application, architecture, and mapping, whereby the mapping specification describes how the application is executed spatially and temporally on the architecture. During the design process, the application, architecture, and mapping specifications are iteratively refined until the quality of service requirements and the real-time constraints of the system are fulfilled. Due to the popularity of streaming applications, various models of computation to specify streaming applications have been proposed. Most of these models have in common that they split the streaming application into autonomous processes and their interconnections. Kahn Process Networks (KPNs) for instance, specify an application as a set of autonomous processes that communicate through point-to-point First-In First-Out (FIFO) channels. In a KPN, each process represents a monotonic mapping of one or more input streams to one or more output streams whereby monotonicity is obtained by having blocking and destructive read access to the channels.

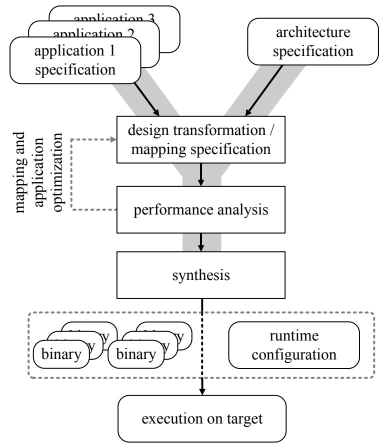
application 3
application 2
application 1 specification
architecture specification
design transformation / mapping specification
mapping and application optimization
performance analysis
synthesis
binary
binary
runtime configuration
execution on target

Fig. 1: Y-chart paradigm [8] to design embedded systems.

Based on this paradigm, several systems that allow the mapping of (sets of) algorithms to many-core embedded systems have been proposed, e.g. CA-MPSoC, CompSOC, the DAEDALUS design flow, the Distributed Operation Layer (DOL) framework [9], Koski, the MAPS design flow [10], the PeaCE framework, SHIM, the StreamIt design flow and the Distributed Application Layer (DAL) [11]. A much more detailed overview about is described in [12], [13].

In the following section we will describe some associated design challenges as well as corresponding approaches.

C. Providing Guarantees

1) Programming models and adaptivity: The resources available to a single application can change over time, which in turn complicates the selection of the right degree of application parallelism at the time that the application is specified. On the one hand, programming the application with too many parallel processes might result in inefficient implementations of the application due to overheads in scheduling and interprocess communication. On the other hand, setting the number of parallel processes too small limits the number of cores that the application can use at once, see e.g. [14], [15]. Therefore, the optimal degree of application parallelism for maximum performance depends on the available computing resources and may change over time.

2) Peak temperature and timing constraints: Reactive thermal management techniques, which are considered efficient tools for temperature control in general-purpose computing systems, keep the maximum temperature under a given threshold by stalling or slowing down the processor, see e.g. [16]. However, causing a significant performance degradation, reactive thermal management techniques are often undesirable in embedded systems, in particular when real-time constraints are tackled, see e.g. [17]–[19]. Therefore, providing guarantees on maximum temperature is as important as functional correctness and timeliness when designing embedded many-core SoCs.

3) Interference-free designs for timing predictability: Tasks that are running concurrently on a many-core platform will interact on shared resources such as caches, shared memory, buses or communication network. If real-time requirements are to be satisfied, the worst-case interferences need to be reduced by means of a proper system architecture that encompasses hardware and software as well as all abstraction layers. Otherwise, the guaranteed performance of a many-core design can be even worse in comparison to that of a single processor architecture.

Most of the existing research on multicore real-time scheduling ignores the effects of resource sharing on the response times of applications. Prerequisite for efficient and predictable execution of applications is the careful selection and design of the underlying hardware architectures in terms of timing anomalies as well as the support for partitioning of shared resources, see e.g. [3], [20]. For bounding interference on the shared memory Yun et al. proposed a software-based memory throttling mechanism [21]. Similarly, the authors of [22] suggested novel memory controller designs for mixed hard realtime and soft real-time systems. These methods require special hardware support as opposed to [23], where task scheduling prevents interference on the shared memory path. Recently, a scheduling strategy was proposed, which explicitly accounts for these interference effects. This policy was combined with an optimization method for the partitioning of tasks to cores as well as the static mapping of memory blocks, i.e., task data and communication buffers, to the banks of a shared memory architecture, see [24].

III. Dealing with Heterogeneity

As discussed in the previous section, model-based programming models are a promising way to deal with the complexity of concurrent execution. Today, models and methodologies have to address two important aspects, namely, application *dynamics* and platform *heterogeneity*. Embedded applications are no longer static (e.g., a simple filter), but include data-dependent processing chains (e.g., a complex video decoder and the complex application described in Section II). Platform heterogeneity is a result of hardware specialization to meet performance and energy constraints. A deeply embedded platform would consist of different types of processing elements, heterogeneous interconnect and memory subsystems. This section describes a typical, practical programming flow that address these aspects. We then introduce new sources of heterogeneity and discuss how they must be dealt with by programming models and HW/SW stacks.

A. Practical Programming Flows

To deal with application dynamics, recent programming flows have focused on expressive programming models, such as those based on Dynamic Dataflow Graphs or KPNs [25]. For these expressive models, programming flows must integrate elaborate code analysis, good frameworks for performance estimation and connections to electronic-system-level simulators

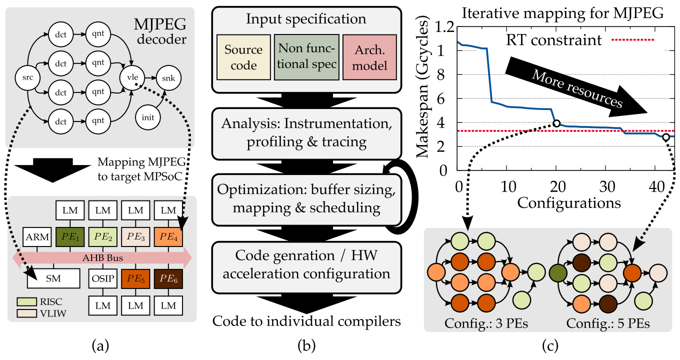

Fig. 2: Practical programming flows. (a) Mapping a KPN application onto a heterogeneous platform. (b) Generic compilation flow. (c) Example mapping configurations and trade-off performance/resource utilization.

(see Section VII). In the presence of heterogeneous platforms, programming flows must include powerful optimization techniques to transform the specification and compute a mapping of the application to the platform resources. Examples approaches include mapping flows with analytical performance estimation in [9], mapping heuristics based on execution traces in [26], and recent improvements to evolutionary techniques to improve their speed in [27]. Methodologies have been also proposed to deal with platforms with extensive hardware acceleration [28]. There are many other flows, including approaches for run-time mapping, and optimization for energy or reliability (see survey in [13]).

Figure 2 shows an example of a typical compilation flow. The mapping problem is illustrated in Figure 2a), where both the processes and the channels of the application specification have to be mapped to processors (PE_i) and memories (local and share memories —LM and SM) of a heterogeneous platform. Due to the impact of interconnect and memory architectures on performance, mapping inter-task communication has become a major focus of mapping heuristics [29], [30]. Figure 2b) shows a generic flow, in which an application is analyzed to understand how processes in the KPN interact with each other. This flow is a materialization of the Y-Chart of Figure 1 (but with no intention to modify the hardware description). The information collected during the analysis phase is then used in heuristics algorithms to iteratively compute a mapping in reasonable time. Apart from the mapping problem, the compiler must also decide on the size of communication buffers, typical in process networks and dataflow application specifications. The iteration in the figure suggests that candidate solutions are evaluated in the real hardware, simulator or with abstract models in order to check if constraints are met. An example of the result of a typical exploration process is shown in Figure 2b), where different mapping configurations represent a trade-off between resources utilization and application runtime. This allows to select the configuration that uses the least resources to meet a given constraint, thereby reducing energy consumption. In the example, the configuration that uses 5 processors is below the real-time constraint, requiring about 3.6 Gcycles for MJPEG decoding. After a software configuration is determined, source code is generated accordingly by using source-to-source compilation. The generated code is finally passed to the compilers of the individual processors to generate binary images. Code generation is very important phase, which account for the vast amount of different architectures across different multi/many-core SoCs.

B. New Sources of Heterogeneity

New technologies and systems with higher processor counts will bring new challenges to the way platforms are programmed. In the context of the German project "Center for Advancing Electronics Dresden" (cfaed.tu-dresden.de), we are investigating *wildly* heterogeneous systems that integrate CMOS with Post-CMOS technologies, with radically different memory architectures as well as integrated wireless and optical interconnects [31], [32]. Memories will keep contributing to system heterogeneity, not only due to different read/write latencies, but also due to different error tolerance/correction properties. In-system wireless communication, in turn, provides a new dimension of flexibility, allowing the software to control the bandwidth available for communication by changing the power allocation to the integrated antennae. New storage and interconnect technology, combined with new ways of processing on Post-CMOS technologies require rethinking the HW/SW stack and the way we program these future many-core systems:

i) Programming models: for scalability and flexibility, programming models such as those described in Section II must support implicit and reconfigurable parallelism (see for example [15], [33]).
ii) Software synthesis and many-core compilers: to bridge the gap, compilers need to understand application-specific semantics, include models of the underlying architecture (see [34]) and closely interact with the runtime.
iii) Operating and runtime systems: these systems must provide layers that hide, manage heterogeneity and deal with isolation (for resources and time).
iv) HW extensions: to reduce the overhead of concurrent execution, the HW should support resource management, ISA-specialization, protection and security (see [35]).

IV. Bridging the Gap: Multi-core Compiler Research and Industry

"A full consistent heterogeneous parallel programming environment is estimated to require 80M€ initial development budget." said Martijn De Lange (ACE) in the Software Tools for Next Generation Computing organized by the European Commission in June 2014. Programming multicore computing systems has been known as a challenging problem for decades. As mentioned earlier in this paper, a great amount of effort has been spent to solve the problem by academia with numerous publications. Nevertheless, the industry is yet to benefit from the research achievements and software development for multicores remains largely manual. In this section, we analyze the *gap* between the state-of-the-art research results and industrial requirements for multicore programming tools from a start-up perspective. By sharing our early experience of commercializing our multicore compiler research results through the spin-off Silexica [36], we explain how to bridge the gap *technically* and *operationally*. This short experience report is hopefully valuable in preparing and planning future research roadmap in multicore computing research.

It is worthwhile to state one obvious fact clearly in the first place. That is, universities and start-up companies share different roles in the brewing pipeline of new technologies. Universities focus more on innovation in both identifying problems and proposing new solutions, while companies strive to provide practical solutions with sufficient quality and performance. This role separation explains some differences in practice by universities and companies naturally. For instance, academic software tools usually have limitations or have not gone through thorough testing, which prevents them from applying to industrial applications directly. In terms of software life cycle, they tend to be bound with project funding or PhD contracts (typically 3-5 years), which cannot generate enough continuum of technology supply that industry demands. Those are commonly seen in all domains as contributions for the gap between academic achievements and industrial requirements.

We take a closer look at what happens in the multicore compiler domain in the following, and present some observations which might result in widening the gap even further:

i) Legacy sequential software vs. new language approach: the fundamental problem why current programming languages fall short for multicore computing systems is lacking of parallelism support. A natural solution, which many academic approaches pursued, is to design new languages which explicitly support parallelism and necessary runtime information. While the introduction of new information is absolutely necessary, the new language approaches may hinder industry adoption.
ii) Introducing parallelism support, especially into the existing programming and compilation flow which is designed for sequential applications, incurs use-model change for end users. In industrial practices, it means that often the existing sequential software (or software building environment) has to be modified either to enable compiler analysis for discovering parallelism or deploy parallelized results. The use-model change will ideally need to be kept as minimal as possible. However, academic tools tend to overlook this problem. The published experimental results are often carried out for standalone, single C file benchmarks, while industrial applications usually involve multiple files, system libraries, mixed language layers, etc. This simplification leads to large overhead or extra work required to adapt the use-model if industrial partners would evaluate the academic works. The first user experience by early industrial adopters suffers to some extent due to this.
iii) Evaluation of academic works in multicore compiler research by industrial companies is extremely difficult. Unlike the relatively small number of architecture types in the uniprocessor time, the complexity and diversity nowadays in multicore platforms is unprecedented (for example see previous Sections I-III). Most of those platforms are either expensive to acquire or company proprietary, which makes them hardly accessible by universities. What makes it even worse is that there is no established benchmark suites for multicore compilers which allow horizontal comparison.

Granted that the gap is wide due to the sheer complexity

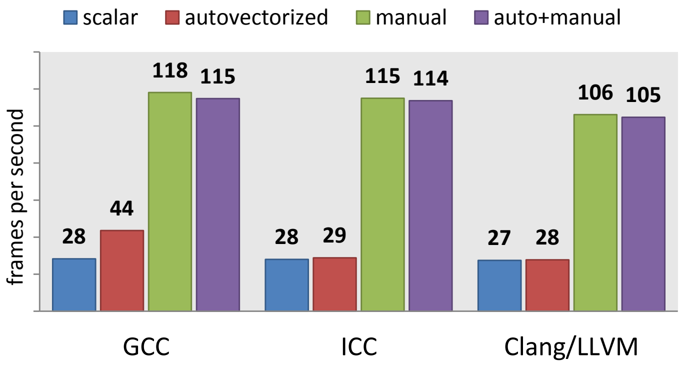

Fig. 3: Performance of a real-world HEVC decoder across C/C++ compilers

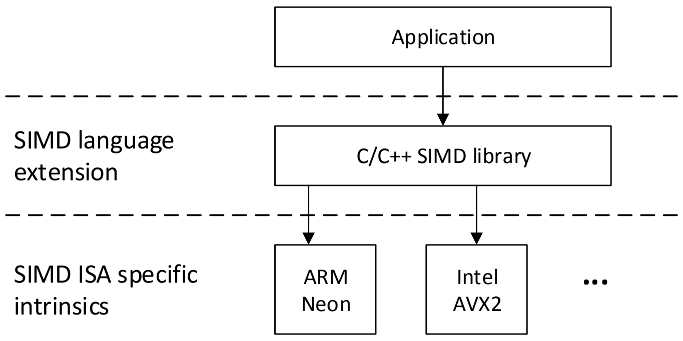

Fig. 4: Abstracting SIMD programming from architecture specifics

of multicore programming, we have taken technical and operational measures that could be helpful in bridging the gap based on our experience. Some of those are listed below which address the aforementioned observations correspondingly:

i) Though a new language approach could be much cleaner and more elegant in expressing parallelism semantic, the pragmatic first step is to support parallelism by extending existing programming languages in reality. We developed a lightweight C-based dialect to introduce model-based parallelism (details in [37]) and provide a partitioning tool in addition which helps end users to gradually convert and parallelize legacy C sequential code to the new dialect. This closeness to C and gradual software migration path are favored by industry partners and certainly help opening door to further opportunities to improve the tools.

ii) To study the use-model change by intruding new compilation tools, one possible solution is to collaborate with industry partners as early as possible in the tool development. Multicore platforms have emerged in virtually all computing industry segments. Nevertheless, different industry segments have drastically different requirements (see Section VI for an example of automobile industry) and therefore a one-model-fit-all solution is unlikely. Our experience showed that early engagements with industrial collaborators are tremendously beneficial in identifying the plausible use-model for end users. As the compilation process for multicore platforms is complex and long, another important factor is to provide users an intuitive (preferably graphical) user interface with comprehensive debugging information to improve user experience.

iii) Though multicore benchmark suites (e.g. MultiBench from EEMBC) have started to get attention recently, it is hard to expect that those benchmarks would be ported to a fairly large number of target platforms due to amount of work required. In general, it is difficult, if not impossible, to benchmark heterogeneous multicore platforms using simple metrics through common benchmarks. To overcome the problem of lacking reference comparison for academia to gain credit from industry, similar to the point above, it would be beneficial to be able to use industry benchmarks possibly through collaboration or professional associations early in the research cycle.

V. Revisiting SIMD Vectorization

The previous section addressed model-based approaches to program multi-cores, which are well suited for expressing high-level parallelism (i.e., task-level parallelism). However, in order to obtain maximum performance in today's multi-cores, application parallelism must be exploited at multiple levels, i.e., looking at low-level parallelism available within the actors or processes in a model-based program. In this section we revisit the problem of exploiting low-level data parallelism to capitalize on Single Instruction, Multiple Data (SIMD) processor extensions.

The vast majority of current multi-core processors are equipped with such SIMD units, offering significant performance gain for applications where the same operation is applied to multiple independent data chunks; one of the most prominent examples for these applications being video codecs.

Since introducing this technology to microprocessors in 1994 [38], however, one of the major downsides had been the significant effort that was required to utilize SIMD instructions. Because DLP had to be explored manually using compiler *intrinsics* and deep knowledge of the application and architecture, the process was error-prone and costly in terms of development time. Furthermore, due to the great variety and incompatibility of SIMD ISAs, intrinsics are different for every one of them. Thus, in order to achieve speed-up on all platforms, multiple versions of the code had to be created and maintained - a developer's nightmare.

Hence, the concept of autovectorization was explored in this context, taking over the time-consuming task of code vectorization and selecting appropriate SIMD instructions. Major compilers, such as the GNU Compiler Collection (GCC), Intel C++ Compiler (ICC) or Clang/LLVM, support autovectorization [39], [40], and efforts have been going on to enable SIMD for interpreted languages such as JavaScript [41] and JIT languages such as Java [42].

Unfortunately, the benefits of autovectorization rarely apply to real-world applications. To illustrate this, Figure 3 depicts the performance difference between a scalar, an autovectorized, and a manually tuned HEVC video decoder [43] for three common C/C++ compilers. Comparing the frames per second of a test-video (4K resolution, 8 parallel threads on an Intel i7-4770 core with AVX2), GCC's autovectorization is by far ahead of the competition. It achieves a significant performance boost of 57% compared to scalar code, while other compilers accomplish a mere 4%. Yet the hand-tuned version is 4.2x faster, emphasizing the need for manual code optimizations. This research therefore focuses on simplifying the usage of SIMD instructions, and proposes a solution to avoid specialized code for each SIMD ISA.

A promising way is separating the code vectorization from the hardware extension. In other words, let the programmer, who knows the application, exploit its parallelism, and let the compiler, who knows the hardware, do the optimization for the target architecture. This will allow the programmer to focus on identifying data level parallelism in the code, without having to deal with architecture specifics; these are taken care of automatically. In combination with maintaining only one version of the source code, a significant speed-up of manual code optimization should be obtained with only minimal performance impact.

The first integral part of this solution is a language extension for SIMD operations, which is SIMD ISA independent and thus works on all compute platforms, for example in the form of a language library for C/C++. Using this library, source code is optimized for all SIMD ISAs at once.

The second component is mapping the language extension to each SIMD ISA's compiler intrinsics. Although not all SIMD ISAs provide the same set of functionality, missing functions can be composed with the available instructions. In this case, ease-of-use outweighs possible performance impacts. The proposed concept is shown in Figure 4.

Of course, such a generic approach can again impact the overall benefit of using SIMD instructions. Nonetheless, we believe that development time will be cut to such an extend that manual optimizations will still be beneficial in terms of effort vs. performance gain, even when compared to no-effort autovectorization. It will take further research to provide evidence, though. Future work will also include investigating the feasibility of the proposed concept across languages, targeting a common ground for future SIMD programming that is independent of language and architecture.

VI. Automotive Case: Requirements on Automation

Multicore systems are becoming a growing interest in automotive applications both in infotainment and safety critical functions (e.g. engine control, brake control, airbag control and the multitude of cruise control systems). For the latter, core redundancy has been proven to be a well suited concept for the identification of software errors for some years now. But both fields of applications are driven by the high demand for increased performance with acceptable power consumption. Many applications are already at the limit of processing speed, and the demand from OEMs for more integrated functions is even growing for coming product upgrades.

A. Situation and Challenges

While infotainment systems with multiple screens are already on the way to multi-processor systems on chip, safety critical functions are more conservative. This is mainly due to the fact that for many safety critical functions the reuse of

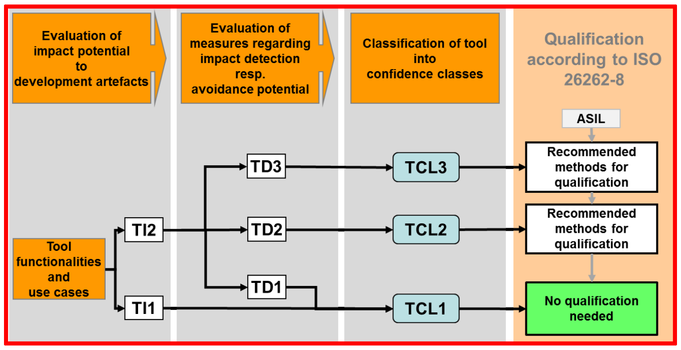
Evaluation of impact potential to development artefacts
Evaluation of measures regarding impact detection resp. avoidance potential
Classification of tool into confidence classes
Qualification according to ISO 26262-8
ASIL
Tool functionalities and use cases
TI2
TI1
TD3
TD2
TD1
TCL3
TCL2
TCL1
Recommended methods for qualification
Recommended methods for qualification
No qualification needed

Fig. 5: Qualification process according to ISO 26262-8.

legacy code, which has proven to be error free in many running systems around the world, is significant.

The ISO 26262 [44], which is an adaption of the IEC 61508 for road vehicles, is a guidance to avoid risks in SW development by providing appropriate methods and processes. For old systems developed before 2011, the norm leaves a loophole called proven in use. According to this norm any modification of safety critical legacy code needs an impact analysis of this modification on the whole system. This can be very cumbersome especially when there is no fully elaborated architecture, which is sometimes true for legacy code. Without an impact analysis one has to work out complete specifications, complete test procedures on all test levels and has to present complete linking between specifications and individual tests. One can even say that the missing of a profound description of the system, which is for example given by an architecture, is a no-go for critical code modifications. But these modifications are unavoidable when running and optimizing different functions on a multicore system.

The development of new safety critical systems in automotive can profit from multicore architectures not only from higher performance, but also for some considerable design aspects. To mention here is the systematic separation of functions with mixed automotive safety integrity levels (ASIL), which have to run simultaneously on the same system without affecting each other, a requirement which goes hand in hand with simultaneous resource sharing. This will come even more into focus, when infotainment systems and safety critical systems run on the same hardware but with different operating systems.

B. Confidence in Software Tools

The methods and tools described in the previous sections and in Section VII greatly simplify software development, however they must adhere to standardization. In this section we shortly describe the qualification process required by the norm ISO 26262-8 to illustrate what tool providers must take into account to make a success entry into safety critical automotive systems.

The ISO 26262-8, does not require the certification or the conformity of development tools, because these tools are not part of the system and are thus out of scope of the norm. But the norm requires the verifiability, that none of the tools used for development of the system harms requirements demanded by ASIL safety classifications of the system. The system development has to deliver substantive documented evidence of conformity for the qualification of the tools in use. Individual tool errors shall not lead to hidden critical errors in the system.

In attempting this, one starts with an evaluation of the impact potential on development artifacts. The process of tool qualification is described in ISO 26262-8. Figure 5 gives a short overview. The tool functionalities and use cases are evaluated to determine whether a malfunction of the tool can introduce errors or fail to detect errors. This is expressed by the classifications TI1 (there is no such possibility) and TI2 (all other cases). Next one makes an evaluation of the measures regarding the impact detection and impact avoidance potential. This is expressed by three detection classes TD1 (high degree of confidence that a malfunction and its corresponding erroneous output will be prevented or detected), TD2 (medium degree), and TD3 (all other cases). Altogether the evaluation allows the classification of the tool into three confidence classes TCL1 – TCL3. For TCL1 no qualification is needed. For the other two, the norm recommends methods for qualification according to the class. The following methods are permissible and recommended depending on the ASIL level:

i) Increased confidence from use.
ii) Evaluation of the tool development process.
iii) Validation of the tool.
iv) Development in compliance with a safety standard.

The result shall be presented in the following documents:

i) Software Tool Qualification Plan.
ii) Software Tool Documentation.
iii) Software Tool Classification Analysis.
iv) Software Tool Qualification Report.

C. Tools for Multicore in Automotive

All together automotive systems can profit thoroughly in different aspects from multicores. But with the increasing number of cores, manual design methodologies will become impractical. Especially the amount of manpower needed will increase essentially with the number of cores. Performance analysis, debugging and testing will become much more extensive. The efficient use of multicore architectures thus needs the help of automation, such as for example mapping and scheduling tools (see Sections II-IV), which integrate all modern aspects of development, e.g. virtual prototyping (see Section VII), variant analysis, and the assistance for complete automatic documentation. For example boundary diagrams or data flow diagrams should be extractable, which are not only needed for documentation, but also for failure analysis (e.g. Failure Mode and Effects Analysis, FMEA).

But the qualification process required by ISO 26262-8 is elaborate and may let tool developers hesitate to support the automotive community, especially for the rather complex task of tool development for multicore architectures. But since this qualification process is a must for all tools in use, one can learn from best practices and from other vendors, who have already been successful in tool qualification (e.g. [45]).

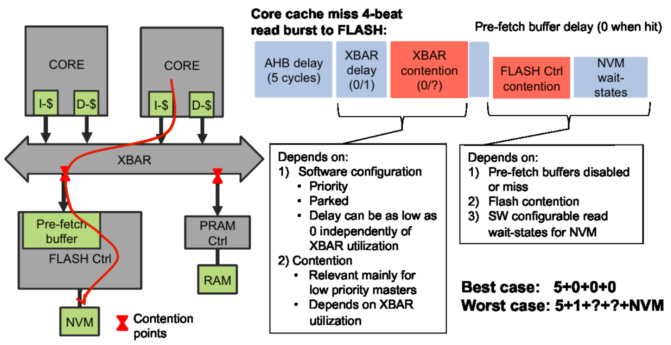
Core cache miss 4-beat read burst to FLASH:
Pre-fetch buffer delay (0 when hit)
CORE
I-$
D-$
CORE
I-$
D-$
XBAR
Pre-fetch buffer
FLASH Ctrl
PRAM Ctrl
RAM
NVM
Contention points
AHB delay (5 cycles)
XBAR delay (0/1)
XBAR contention (0/?)
FLASH Ctrl contention
NVM wait-states
Depends on:
1) Software configuration
• Priority
• Parked
• Delay can be as low as 0 independently of XBAR utilization
2) Contention
• Relevant mainly for low priority masters
• Depends on XBAR utilization
Depends on:
1) Pre-fetch buffers disabled or miss
2) Flash contention
3) SW configurable read wait-states for NVM
Best case: 5+0+0+0
Worst case: 5+1+?+?+NVM

Fig. 6: Performance variations due to SW-HW configurations and run-time conditions

VII. Automotive Case: The Role of Virtual Prototypes

As discussed in the previous section, as the amount of software in a car keeps rising, automotive companies have shifted to multicore architectures as a way to reduce the cost, the complexity and the power consumption associated with solutions that use discrete Electronic Control Unit (ECUs). This shift to multicores requires new programming models, methodologies and tools such as those in Sections II-IV. New programming tools alone are however not the entire solution. Both for old and new programming tool flows, the automotive industry badly needs tools that allow to analyze a running software in terms of functionality (debugging) and performance (finding bottlenecks). In this section we discuss how virtual prototypes can be used to provide the programmer with the insight to analyze automotive applications.

A. SW & HW Aspects that Affect Software

New challenges both from functional and performance angles are introduced when porting an existing software stack to a new multicore architecture. From the functional aspect, new bugs such as race conditions and data coherence are introduced. From the performance angle, different functions running on different cores share hardware resources, such as memories, on-chip busses and peripherals, which may introduce significant jitter on the software execution. Variations in execution times may introduce deadline misses of critical tasks with catastrophic consequences. Finding the appropriate configuration of such hardware resources that provides sufficient performance in all conditions is a daunting task. This is especially important for safety critical applications where “freedom of interference” must be ensured to achieve certification.

In automotive applications several aspects influence performance, including:

i) I/O rates: automotive applications tightly interact with the environment. Typical examples include dealing with input events from A/D converters or network messages (e.g., CAN), and output events to control actuators or signal generation (e.g., PWM). Events have different rates ranging from KHz to MHz and are served by different components in the multicore.

ii) SW configuration: In a multicore environment, the programmer must understand the mapping of tasks to cores, the parameters a real-time schedulers (periods, deadlines and priorities), the primitives used for inter-task communication and synchronization and the mapping of code and data to different memories (e.g, TCMs, RAMs or FLASH).

iii) SW-controlled HW configuration: apart from software-centric configurations, automotive multicores allow for several software-controlled hardware modes such as enabling caches and FLASH pre-fetch buffers or controlling interconnect priorities and clock frequencies.

iv) Run-time dynamics: during execution, the timing of an application varies due to factors that are difficult to analyze statically such as the effect of branch prediction, the effect on cache miss/hit ratio of memory access patterns and bus contention.

Dealing with the aspects above is a cumbersome task. Understanding the impact of event handling in a multicore requires detailed insight into the SW-HW interactions. This is typically hidden in high-level programming models. SW and HW configuration aspects are particularly difficult to understand from a low-level implementation if they were automatically generated with methods such as those in Sections II and III. Finally, run-time aspects require analysis of the running system. Figure 6 shows a simple example to illustrate the impact of these aspects on performance. In a typical multicore architecture, trying to access external memory can lead to different latencies influencing the software performance. Differences between best case and worst case are significant, due to all the factors and parameters involved. For instance, the configuration of the static hardware parameters (bus priorities, level 2 storage), but also the dynamic aspects such as contention, address patterns for previous accesses, etc. All this makes it very difficult to estimate the software performance for all operating conditions.

B. Virtual Prototypes

To deal with all the sources of functional and performance *bugs* discussed above, a programmer needs a good vehicle to quickly measure software in a non-intrusive way. Besides, the programmer needs tools that allow to quickly identify performance bottleneck and sources of bugs. Both tasks can be performed with virtual prototypes, which are a fully functional representation of the digital hardware. The embedded software sees no differences when executing on a virtual prototype or real hardware. Additionally, virtual prototypes ease platform exploration as mentioned in Section VI and can be embedded in a simulation that includes physical models (e.g., with Simulink).

Figure 7 shows sample traces used in an automotive case study. The figure shows how virtual platforms can be used to correlated hardware events and bottlenecks with software objects (e.g., source code symbols and OS data structures). Typical software views are not sufficient to understand where bottlenecks are on the hardware. Likewise, hardware views are not sufficient to pin point areas on the software that can be candidates for optimization. Correlated views that show hardware bottlenecks in the context of the software structure are the answer here. With these views one can observe when

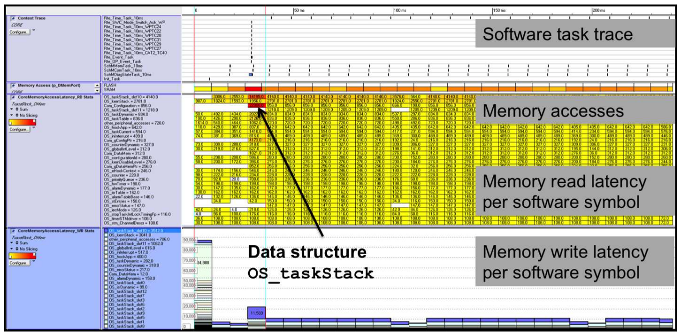
Software task trace
Memory accesses
Memory read latency per software symbol
Data structure OS_taskStack
Memory write latency per software symbol

Fig. 7: Sample traces from automotive case study

and where bottlenecks appear and more importantly what is the reason for them. Using heat maps and stack bar views, the main contributors (with highest likelihood to get a performance improvement when optimized) are quickly identified. Based on these traces, a programmer easily identified data structures such as `OS_taskStack` to be the bottleneck and remapped them to a tightly coupled memory. More advanced uses of virtual prototypes for analysis and debugging can be found in [46], [47].

VIII. Conclusion

In this paper we presented different views of the current standing of multicore and many core programming methods and tools. The authors concur that model-based design is a must to deal with the complexities of today's systems and applications. In academia, we see a trend towards new constructs for adaptability, ideas to deal with thermal issues while keeping real-time constraints, approaches to reduce the interference between different applications sharing resources and an outlook to deal with heterogeneity in future systems. Sections IV-VI put some of the advances in academia in perspective. In particular, Section IV described the gap that most academic tools must bridge if they strive at industrial adoption. Section V showed some problems considered solved in academia, lack actual success in real applications. We described how low-level SIMD parallelization for single-cores can be supported with generic programming constructs, so that the code within an actor of a dataflow network can profit from modern architectural traits. Section VI discussed how important it is for programming tools to be aware of the complex standardization needed to succeed in the automotive domain. Finally, Section VII touched on system analysis and debugging, an aspect that often neglected in academia.

Acknowledgement

Part of the research vision in this paper is supported by the cluster of excellence "Center for Advancing Electronics Dresden" (cfaed).

References

[1] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2012, pp. 983–987.

[2] F. Conti, C. Pilkington, A. Marongiu, and L. Benini, "He-p2012: architectural heterogeneity exploration on a scalable many-core platform," in *Proceedings of the 24th edition of the great lakes symposium on VLSI*. ACM, 2014, pp. 231–232.

[3] B. D. d. Dinechin, P. G. d. Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel, "A distributed run-time environment for the kalray mppa 256 integrated manycore processor," *Procedia Computer Science*, vol. 18, pp. 1654–1663, 2013.

[4] J. Diaz, C. Munoz-Caro, and A. Nino, "A survey of parallel programming models and tools in the multi and many-core era," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 8, pp. 1369–1386, 2012.

[5] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: Formal models, validation, and synthesis," *Readings in hardware/software co-design*, p. 86, 2001.

[6] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, 2003.

[7] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd ed. New York, NY, USA: Marcel Dekker, Inc., 2009.

[8] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf, "An approach for quantitative analysis of application-specific dataflow architectures," in *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*. IEEE, 1997, pp. 338–349.

[9] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping applications to tiled multiprocessor embedded systems," in *Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on*. IEEE, 2007, pp. 29–40.

[10] R. Leupers and J. Castrillon, "Mpsoc programming using the maps compiler," in *ASP-DAC '10: Proceedings of the 15th Asia and South Pacific Design Automation Conference*, 2010, pp. 897–902.

[11] L. Schor, I. Bacivarov, L. G. Murillo, P. S. Paolucci, F. Rousseau, A. E. Antably, R. Buecs, N. Fournel, R. Leupers, D. Rai, L. Thiele, L. Tosoratto, P. Vicini, and J. Weinstock, "Euretile design flow: Dynamic and fault tolerant mapping of multiple applications onto many-tile systems," in *Proc. IEEE International Symposium on Parallel and Distributed Processing with Applications Article (ISPA)*. Milan, Italy: IEEE Computer, Aug 2014, pp. 182–189.

[12] I. Bacivarov, W. Haid, K. Huang, and L. Thiele, "Methods and tools for mapping process networks onto multi-processor systems-on-chip," in *Handbook of Signal Processing Systems*. Springer, 2013, pp. 867–903.

[13] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proceedings of the 50th Annual Design Automation Conference*. ACM, 2013, p. 1.

[14] M. I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5. ACM, 2006, pp. 151–162.

[15] L. Schor, I. Bacivarov, H. Yang, and L. Thiele, "Adapnet: Adapting process networks in response to resource variations," in *Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. New Delhi, India: ACM, Oct 2014, pp. 8.1:1–8.1:10.

[16] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2. IEEE Computer Society, 2006, pp. 78–88.

[17] L. Schor, I. Bacivarov, H. Yang, and L. Thiele, "Efficient worst-case temperature evaluation for thermal-aware assignment of real-time applications on mpsocs," *Journal of Electronic Testing: Theory and Applications*, vol. 29, no. 4, pp. 521–535, Aug 2013.

[18] L. Thiele, L. Schor, I. Bacivarov, and H. Yang, "Predictability for timing and temperature in multiprocessor system-on-chip platforms," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 1s, pp. 48:1–48:25, Mar. 2013.

[19] P. Kumar and L. Thiele, "Worst-case guarantees on a processor with temperature-based feedback control of speed," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4s, pp. 122:1–122:26, Apr. 2014.

[20] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, J. Wolf *et al.*, "Merasa: Multicore execution of hard real-time applications supporting analyzability," *Micro, IEEE*, vol. 30, no. 5, pp. 66–75, 2010.

[21] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*. IEEE, 2012, pp. 299–308.

[22] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero, "Hardware support for wcet analysis of hard real-time multicore systems," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 57–68.

[23] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele, "Scheduling of mixed-criticality applications on resource-sharing multicore systems," in *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*. IEEE, 2013, pp. 1–15.

[24] ——, "Mapping mixed-criticality applications on multi-core architectures," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–6.

[25] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Information Processing '74: Proceedings of the IFIP Congress*, J. L. Rosenfeld, Ed. New York, NY: North-Holland, 1974, pp. 471–475.

[26] J. Castrillon, R. Leupers, and G. Ascheid, "MAPS: Mapping Concurrent Dataflow Applications to Heterogeneous MPSoCs," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 527 – 545, Feb. 2013, (Online since 10.2011).

[27] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for mpsocs," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 131:1–131:6. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488895

[28] J. Castrillon, S. Schürmans, A. Stulova, W. Sheng, T. Kempf, A. Ishaque, R. Leupers, G. Ascheid, and H. Meyr, "Component-based Waveform Development: the Nucleus Tool Flow for Efficient and Portable SDR," in *2010 Wireless Innovation Conference and Product Exposition (SDR'10)*. Washington D.C., USA: Wireless Innovation Forum, Dec 2010.

[29] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," *Journal of Systems Architecture*, vol. 56, no. 7, pp. 242–255, 2010.

[30] J. Castrillon, A. Tretter, R. Leupers, and G. Ascheid, "Communication-Aware Mapping of KPN Applications onto Heterogeneous MPSoCs," in *DAC '12: Proceedings of the 49th annual conference on Design automation*, 2012.

[31] G. Fettweis, W. Nagel, and W. Lehner, "Pathways to servers of the future: highly adaptive energy efficient computing (haec)," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 1161–1166.

[32] Y. Xie, "Future memory and interconnect technologies," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 964–969.

[33] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting, "Invasive computing: An overview," in *Multiprocessor System-on-Chip*. Springer, 2011, pp. 241–268.

[34] S. Pfennig, E. Franz, F. Ciorba, T. Ilsche, and W. Nagel, "Modeling communication delays for network coding and routing for error-prone transmission," in *Future Generation Communication Technology (FGCT), 2014 Third International Conference on*, Aug 2014, pp. 19–24.

[35] O. Arnold, B. Noethen, and G. Fettweis, "Cm_isa++: An instruction set for dynamic task scheduling units for more than 1000 cores," in *System-on-Chip Conference (SOCC), 2014 27th IEEE International*, Sept 2014, pp. 29–34.

[36] "Silexica Software Solutions GmbH," http://www.silexica.com, accessed: 2014-11.

[37] W. Sheng, S. Schürmans, M. Odendahl, M. Bertsch, V. Volevach, R. Leupers, and G. Ascheid, "A compiler infrastructure for embedded heterogeneous mpsocs," vol. 40, no. 2. Amsterdam, The Netherlands,

The Netherlands: Elsevier Science Publishers B. V., Feb. 2014, pp. 51–68. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2013.11.007

[38] R. B. Lee, "Accelerating Multimedia with Enhanced Microprocessors," *IEEE Micro*, vol. 15, no. 2, pp. 22–32, 1995.

[39] D. Naishlos, "Autovectorization in GCC," in *Proceedings of the 2004 GCC Developers Summit*, 2004, pp. 105–118.

[40] A. J. Bik, *The Software Vectorization Handbook: Applying Intel Multimedia Extensions for Maximum Performance*. Intel Press, 2004.

[41] M. Haghihat. (2013) Bringing SIMD to JavaScript. [Online]. Available: https://01.org/node/1495

[42] J. Parri, D. Shapiro, M. Bolic, and V. Groza, "Returning Control to the Programmer: SIMD Intrinsics for Virtual Machines," *Communications of the ACM*, vol. 54, no. 4, pp. 38–43, 2011.

[43] C. C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, and T. Schierl, "SIMD Acceleration for HEVC Decoding," *IEEE Transactions on Circuits and Systems for Video Technology*, 2014.

[44] "ISO 26262, Road vehicles – Functional safety," 2011.

[45] G. S. Mirko Conrad and P. Munier, "Software Tool Qualification According to ISO 26262," *SAE International*, 2011.

[46] J. Castrillon, A. Shah, L. G. Murillo, R. Leupers, and G. Ascheid, "Backend for virtual platforms with hardware scheduler in the maps framework," in *Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on*. IEEE, 2011, pp. 1–4.

[47] L. G. Murillo, S. Wawroschek, J. Castrillon, R. Leupers, and G. Ascheid, "Automatic detection of concurrency bugs through event ordering constraints," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*. IEEE, 2014, pp. 1–6.