A software stack for next-generation automotive systems on many-core heterogeneous platforms

Paolo Burgio*, Marko Bertogna*, Ignacio Sañudo Olmedo*, Paolo Gai†, Andrea Marongiu‡, Michal Sojka§
*University of Modena and Reggio Emilia, Modena, Italy – †Evidence srl, Italy
‡Swiss Federal Institute of Technology (ETH), Zurich, Switzerland – §Czech Technical University in Prague, Czech Republic
Email: {paolo.burgio, marko.bertogna, ignacio.sanudoolmedo}@unimore.it, pj@evidence.eu.com, a.marongiu@iis.ee.ethz.ch, sojkam1@fel.cvut.cz

***Abstract*—The advent of commercial-of-the-shelf (COTS) heterogeneous many-core platforms is opening up a series of opportunities in the embedded computing market. Integrating multiple computing elements running at smaller frequencies allows obtaining impressive performance capabilities at a reduced power consumption. These platforms can be successfully adopted to build the next-generation of self-driving vehicles, where *Advanced Driver Assistance Systems (ADAS)* need to process unprecedently higher computing workloads at low power budgets. Unfortunately, the current methodologies for providing real-time guarantees are uneffective when applied to the complex architectures of modern many-cores. Having impressive average performances with no guaranteed bounds on the response times of the critical computing activities is of little if no use to these applications. Project HERCULES will provide the required technological infrastructure to obtain an order-of-magnitude improvement in the cost and power consumption of next generation automotive systems. This paper presents the integrated software framework of the project, which allows achieving predictable performance on top of cutting-edge heterogeneous COTS platforms. The proposed software stack will let both real-time and non real-time application coexist on next-generation, power-efficient embedded platform, with preserved timing guarantees.**

I. Introduction

Automotive systems are entering a new era, where vehicles integrate more and more functionalities: they are heavily connected with the surrounding environment and the internet-of-things, and most importantly, they become capable of independent decisions, that is, they are more "intelligent". This revolutionary change in the way we build our cars requires a technological shift also in the computing platforms, opening up a number of opportunities for innovation and research. All the main players of the automotive market are spending an increasing amount of resources in this direction. Major OEM and Tier-1s such as BMW, Volvo, Tesla Motors, and General Motors are already developing the necessary know-how and technological background to build the next generation of embedded automotive systems. Recently, even companies from other markets, such as Apple and Google, are entering this challenge (see the GoogleCar [1]). The next challenge is to build – and especially, to commercialize and sell – a completely automated self-driving car, supported by a *Advanced Driver Assistance Systems (ADAS)*. In order to build fully-featured ADAS, system engineers face a number of challenges and unprecedented requirements, which are far from being satisfied. Such a system must:

1) manage heavy sensor-fusion and image-processing;
2) run with reduced power consumption, allowing smaller batteries and renewable power sources to be put onboard the vehicle;
3) quickly interact with the environment, requiring a prompt elaboration of sensor data;
4) automatically run highest criticality workloads, replacing safety-critical human activities.

The converging needs for predictable high-performance at low power call for a "real-time embedded super-computing platform", i.e., a platform capable of predictably providing real-time guarantees to applications running on top of power-efficient embedded hardware.

Modern Commercial-Off-The-Shelf heterogeneous architectures (COTS) based on multi- and many-core accelerators can satisfy this need for energy-efficient performance. Integrating multiple computing elements running at lower frequencies allows obtaining impressive performance capabilities at a reduced power consumption, while architectural heterogeneity enhances platform flexibility. Examples of such platforms are the NVIDIA Tegra X1 [2], a GPU-based SoC (described in Figure 1), and the Xilinx Ultrascale [3], which also embeds programmable logics. Unluckily, their tremendous potential in terms of performance/Watt comes at the price of increased architectural complexity, which ultimately makes it extremely difficult to write efficient code for them (**poor programmability**). Even more importantly, for the automotive domain, established methodologies and tools to provide real-time guarantees are born for single-core systems; when applied to the complex designs of heterogeneous many-cores, they lead to over-conservative and pessimistic timing bounds that ultimately prevent their adoption within industrial settings (**poor predictability**). For this reason, the design methodologies and software stack for automotive systems must be heavily modified, and to some extent re-designed, to cope with the next generation of platforms.

These are the motivations behind the HERCULES ("High-Performance Real-time Architectures for Low-Power Embedded Systems") Project [4]. The ambitious goal of HERCULES is to obtain an order-of-magnitude improvement in the cost and power consumption of next generation real-time applications for safety-critical domains.

This paper introduces the software stack envisioned in HERCULES. To do so, we show some of the design choices characterizing current automotive systems, guided by industrial requirements from project partners, namely Airbus AGI, Magneti Marelli S.p.A. and Pitom snc. We first describe the

platform template considered in the project (Section II) and the chosen programming models (Section III). These choices also take into account industrial needs, like the reuse of *legacy code and libraries on heterogeneous many-core devices without requiring heavy code refactoring*. This allows maximizing the industrial impact of the HERCULES framework (methodologies, tools and software), simplifying the technological transfer to existing application scenarios. In Section IV we describe the full software stack. Finally, Section V draws some conclusions.

II. Target Architecture

The choice of the target computing platforms is crucial and it affects every part of the technological stack of the project. With very limited exceptions, real-time systems have been based on embedded architectures which have not (or at least not exclusively) been thought to address the predictability and analysability requirements of time-critical applications. The few platforms designed for being fully timing analysable became quickly obsoleted by later process technologies. For this reason, HERCULES will employ Commercial-Off-The-Shelf (COTS) components.

The advantage of using COTS are multiple: they are cheaper than custom-made solutions; more robust to hardware and timing faults; fully supported by the hardware provider; and easily available for market exploitation.

HERCULES targets heterogeneous architectures, made by a "traditional" high-performance host cores (such as ARM Cortex or Intel i*X*) and a many-core accelerator, such as GPUs [2], in case coupled with FPGA logics [3]. The techniques, rationales and tools developed within HERCULES will be designed to be easily portable on future platforms. This is possible thanks to the software stack and highly expressive programming models, which allow hiding the complexity of the hardware architecture, and ensuring application portability.

The **ARM Big.LITTLE** [5] (2011) represents the tate-of-the-art for the target "host" subsystem. It couples powerful "big" cores such as ARMv8 Cortex-A57 and slower yet more power-efficient "little" cores such as Cortex-A53 (ARMv8 architecture). Since the two subset of cores share the on-chip memory banks, and caches are coherent, workloads may migrate between them almost on-the-fly. This is typically performed transparently by the OS, e.g., in case of Linux, by the `cpufreq` infrastructure. HERCULES will employ a *Heterogeneous Multi-Processing (HMP)* model, which enables concurrently exploiting all physical cores at once, as opposite to the "traditional" *clustered switching* model, where only one subsystem is active at the same time.

Tegra [2] is a family of NVIDIA SoCs explicitly targeting embedded systems such as tablets and smartphones. It is shown in Figure 1. It couples one or more ARM cores with a General Purpose GPU (GP-GPU) in a single package. On one side, poorly parallel, control-based and I/O computations are typically executed on the host subsystem, while highly parallel workloads are offloaded to the power-efficient many-core accelerator. The latest release of the family is the Tegra X1 [2] platform which embeds an octa-core host with Big.LITTLE configuration and a Maxwell GPU with 256 CUDA cores. The platform is claimed to achieve 1 TFLOP of computing power,

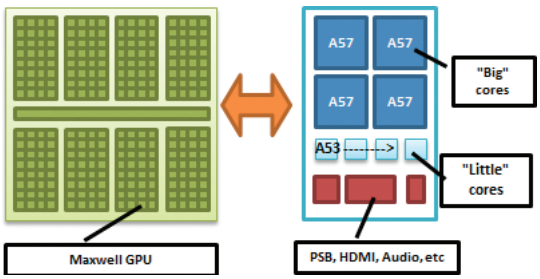

Fig. 1. Scheme of the NVIDIA Tegra X1 platform

within only 15 Watts. Tegra X1 is not certified for *Functional Safety and Road Vehicles Standard (ISO 26262 [6])*. However, NVIDIA declared that the next version of the platform – based on the novel Parker architecture – will be qualified ASIL-B [7]. Since the two platforms are similar from an architectural point of view, the technology produced by HERCULES on the Tegra X1 we will be easily ported to the Parker, as soon as it will be available, i.e., end of 2016.

The architectures depicted in this section do not cover the full spectrum of modern heterogeneous many-core platforms, yet they are quite representative of current market trends and products. The HERCULES project targets future automotive and avionics systems, whose computational platforms are expected to provide hundreds of GOPs within a few Watts. This is the main reason behind the selection of Tegra-like platform as one of the reference architectures of the project.

III. Programming Model

This section describes the programming models envisioned in the project. There is a plethora of *de-facto* or *de-jure* standards for programming heterogeneous architectures, which we want to address to ensure not only compliancy with legacy code and software libraries, but also with *existing methodologies, tools and, most of all, programmers' expertise*. We hereafter introduce the main advantages and drawbacks for each of them.

In order to provide a clean and simple programming interface, we decided to support directive-based programming APIs, such as OpenMP [8]. CUDA [9] or OpenCL [10] will also be supported. Compiler transformation will be employed to convert high-level constructs to lower-level programming routines written in CUDA or OpenCL. As explained in the following, the HERCULES ecosystem will also support legacy hard-real time applications written in AUTOSAR [11].

A. Programming heterogeneous platforms with OpenMP

OpenMP [8] is the *de-facto* standard for programming shared-memory systems. OpenMP was developed at the end of 90's to program regular, loop-based workload on top of simmetric multiprocessors systems (SMP) with shared-memory. More recently [12], it evolved to deal with more irregular and dynamic parallelism, switching from a loop-oriented approach to a *task*-oriented approach. Finally, with specifications 4.5 [8] (2011), it also embraced heterogeneous computing paradigm and execution model based on subroutines (so-called *target* regions) which are offloaded to an accelerator device, partially

relaxing the SMP execution model and, ultimately, the shared-memory model. In its specifications, OpenMP is a set of APIs, pragmas and environmental variables. To implement a full parallel software stack, it relies on a runtime which provides basic functionalities for threading and resource allocation/-management. The actual runtime implementation, and its set of APIs, are compiler-specific: for instance, the most known GNU port relies on the so-called GCC-OpenMP (GOMP) framework [13]. Although some efforts have been made [14], [15], OpenMP is not yet suitable for real-time computing, neither it mentions real-time capabilities in the standard.

B. Programming models for GP-GPUs: CUDA and OpenCL

The preferred solution for programming NVIDIA GP-GPUs are either CUDA [9] or OpenCL [10].

CUDA provides a set of APIs for (massive) threading and hooks for data movement/placing on the GPU device. Application code runs on top of a runtime library + GPU driver which works together with the operating system to provide these services. There are no implicit real-time guarantees in the CUDA standard. Original CUDA runtime and drivers are closed and proprietary. The consortium stipulated an informal agreement with NVIDIA to jointly develop a basic support for real-time computing on their embedded platform. The GPU is currently seen as a non-preemptible, shared resource with run-to-completion semantics. In the project, we will explore the possibility of relaxing these assumptions by adding preemption support and concurrent programming capabilities of a single GPU device. In Figure 2, "real-time CUDA extensions" which might be developed during the project are marked with a star.

Open-Computing-Language (OpenCL [10]) is a joint effort by the Khronos Consortium [16] for building an open language for programming accelerator-based platforms. Similarly, to CUDA it provides non real-time APIs for threading and memory management at the application/user level, relying on a runtime+OS+drivers subsystem. Similarly to CUDA, OpenCL increases the complexity of application code, and does not provides real-time guarantees.

C. Automotive programming models and Autosar RTE

The previous subsections introduced a set of programming models for non real-time parallel software. In the automotive domain, on the other hand, we currently see two diverging trends. On one side, the real-time, statically allocated, statically configured AUTOSAR standard [11] proposes a complete software stack including device MCAL drivers, Basic Software, RTOS and Run Time Environment (RTE) to implement a standard software component model. On the other side, the infotainment world typically relies on non real-time versions of Linux, Android, and proprietary solutions coexisting in the same system.

One of the main goals of HERCULES project is to harness the computational power of next-generation parallel embedded platforms, inside a framework where AUTOSAR-compliant real-time applications may concurrently run along with infotainment and non real-time software, without affecting the required timing guarantees.

For this reason, the project aims to support a subset of the AUTOSAR specification by extending the operating system with a minimal RTE support, coupled with appropriate mechanisms allowing the sharing of data as well as the concurrent usage of common peripherals (see Section IV). HERCULES operating systems layer is based on ERIKA Enterprise [17], [18], an open-source OSEK/VDX certified OS, and the real-time patched versions of the Linux kernel. They are discussed more in details in Section IV-B.

By allowing the integration of AUTOSAR components together with high-performance software stacks, the consortium aims to ensure **portability** of code provided by different suppliers with different degrees of real-time support. Special attention is also given to the possibility to run code certified under the ISO 26262 automotive functional safety standard [6]. Although the project does not aim to provide an ASIL certified stack, the potential usage of the architecture proposed in safety applications will be analyzed, and recommendations for the creation of a certifiable stack will be produced.

IV. HERCULES SOFTWARE STACK

Figure 2 depicts the HERCULES software framework. The HERCULES project is not tied to a specific system/SoC, yet it targets a specific architectural template, coupling a certified hard-real time platform and heterogeneous SoC with multi-core host and many-core accelerator/GPU.

As seen in Section III, HERCULES aims to support on the same architecture both real-time AUTOSAR-like applications [11] running on top of ERIKA Enterprise [18], and non-real-time (but high-performance) computations performed partly in the Big.LITTLE-like subsystem and partly in the many-core accelerator. In addition to this requirement, it aims to support ISO 26262 certifications [6] of some of the safety critical parts. Then, it becomes mandatory to guarantee a proper isolation between the hard real-time parts and the rest of the system, in order to obtain the freedom from interference required by the standard in the part 6, annex D.

For these reasons, the hard real-time subsystems have been properly "isolated":

1) For high levels of safety requirements, the project is planning the integration of an AUTOSAR subsystem supporting an external ASIL-D compliant CPU, such as the Tricore AURIX [19]. The external CPU will be connected to the many-core fabric using a predictable communication infrastructure.
2) For lower levels of safety requirements, the real-time subsystem will be integrated in the Big.LITTLE cores. To ensure freedom from interference, we will implement a **hypervisor**, to separate and isolate the real-time subsystem (run under the ERIKA Enterprise kernel) from the rest of the system.

A. Virtualization in HERCULES

As for the hypervisor, the project is currently evaluating several options, starting from existing open-source projects. The choice of the hypervisor will be guided by a set of requirements, such as:

1) possibility of running multiple operating systems (such as Linux and ERIKA Enterprise [18]);
2) possibility to support core assignments (*pinning*) to the single guest OS; It is important to state that we are not

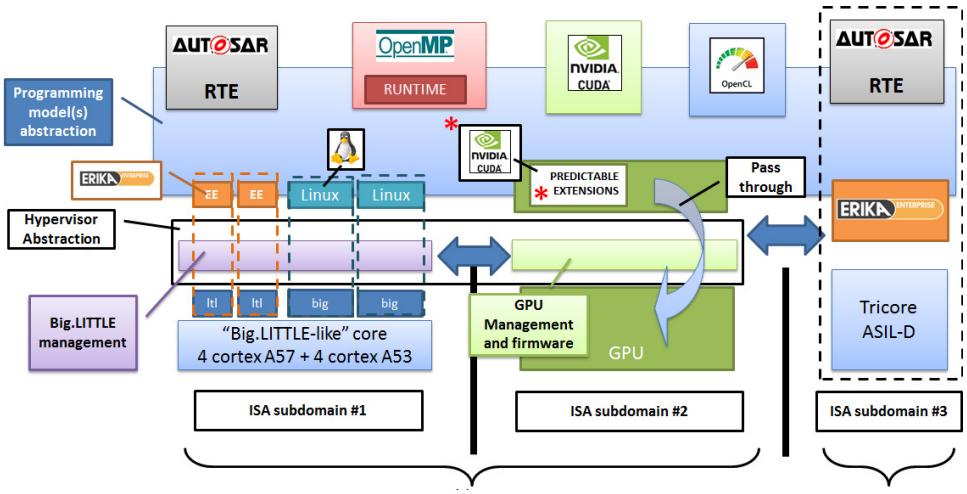

Fig. 2. The HERCULES Software Stack

aiming at the cohexistence of a high number of Virtual machines on the same CPU (as it happens in cloud environments), but rather the typical setup will statically allocate a single OS to one or more CPUs, to limit the virtualization overhead, still maintaining the separation needed by the safety standards;

3) possibility to share peripherals such as graphic cards and busses among domains;
4) possibility to be certified, which typically means choosing hypervisors which have a minimal footprint (in the order of 10k lines of code – LoCs);
5) possibility to support heterogeneous architectures which are typically present in the many-core systems used in the project.

Looking at the bottom of Figure 2, we see how that the platform subsystem might have different Instruction Set Architectures, identifying multiple ISA sub-domains. We hide this complexity to the programmer with a virtualization abstraction, composed of one module for each sub-domain. In order to support this partitioning of the HW/SW system, we will implement predictable communication among the multiple subdomains (the blue horizontal arrows).

1) Virtualization of the host subsytem: Virtualization is widely adopted in general purpose platforms, especially for cloud computing. One of the most widely adopted open source hypervisors is probably Xen. A recent effort to enhance Xen with real-time capabilities is the project **RT-Xen** [20] (in mainline Xen since v4.6). It implements a hierarchical real-time scheduling framework based on global EDF scheduling, within approximately 100K LoCs of C code. Unfortunately, the size of the hypervisor is crucial, as the effort for system verification, hence, certification, grows significantly with the number of LoCs. For this reason, micro-solutions such as Jailhouse or Nova are strongly preferred in this project.

Jailhouse [21], developed by Siemens, is a Linux-based hypervisor oriented to real-time. Jailhouse isolates the virtual machines in small cells with few lines of code (13513 written in C), removing all of the unnecessary features (e.g., hooks for diagnostic tools), and schedules the virtual machines by pinning them to the computing cores. It also allow running bare-metal applications aside to Linux.

NOVA OS Virtualization Architecture [22] is even smaller (approx, 9K LoCs written in C++), and, similarly to Jailhouse, it is capable of running virtual machines and bare metal applications side-by-side. However, unlike Jailhouse, NOVA is not pinning pysical cores, but it implements preemptive priority-driven round-robin scheduler.

2) GPU management and virtualization: As GP-GPUs became mainstream, there was a big interest of virtualizing graphic cards, e.g., for cloud computing. Unfortunately, "hiding" one or multiple GPUs under a hypervisor introduces a serious performance penalty for crossing its software layers, which ultimately might compromise the advantage of many-core acceleration. For this reason, a common solution is to provide a so-called *pass-through* mechanism for the CUDA drivers, which are allowed to bypass the virtualization layers and direct access the device. This mechanism is represented by the arrow in Figure 2 that directly accesses the GPU, and it's currently supported on a limited set of GPUs, and for specific drivers[1]. A number of hypervisors and virtualization schemes exist for GPUs. Interested reader might refer to [24] as a good survey.

B. Host Operating Systems: Erika and Linux

For the host part, we decided to start from the application requirements we got from partners, in order to build an innovative infrastructure which may obtain good performances while supporting legacy code.

The choice of ERIKA Enterprise enables us to run traditional AUTOSAR applications on top of an open-source implementation. ERIKA Enterprise [18] is currently the only open-source OSEK/VDX certified operating system, and it implements some of the extensions specified in the AUTOSAR OS standard. This opens the possibility to run traditional automotive applications with minimal or no change. For that reason, ERIKA Enterprise will be ported on the Big.LITTLE

[1]For instance, NVIDIA published [23] a list of applications which are certified for this technology (called NVIDIA Grid).

subsystem. ERIKA Enterprise as of today supports directly the Tricore AURIX architecture. HERCULES intends to allow the same interface between the AUTOSAR contents hosted on the Big.LITTLE subsystem and the ASIL components running on a separate Infineon Tricore ECU.

Linux is the best candidate to support the many-core programming models described in Section III, opening the potential of a seamless integration between the Big.LITTLE subsystem and the many-core infrastructure used as accelerator.

C. Scheduling of shared resources

In order to achieve predictable execution on many-core platforms, it is be necessary to control the way how individual cores access the shared resources such as on-chip interconnects and memory buses. The approach proposed in HERCULES is inspired by the so-called PRedictable Execution Model (PREM) [25], [26], where the predictability of memory access from the single software components (*tasks*) is increased using prefetching techniques. The hypervisor will monitor the behavior of application using performance counters, and throttle potentially misbehaving cores, to support PREM-based execution.

V. Conclusion

This paper describes the goals and organization of the HERCULES H2020 project [4], a first attempt of building a complete software stack for automotive systems based on commercial-off-the-shelf components, and including ASIL-D certified subsystem for running legacy, hard-real time workload.

As a summary, we hide the complexity of the underlying platforms by mean of virtualization, and provide support for two kind of operating systems. On one side, the statically configured ERIKA Enterprise (typically pinned to one of the "little" cores), which allows running static real-time applications typical of the automotive market. On the other side, Linux with RT extensions (typically running on all the remaining CPUs of the Big.LITTLE infrastructure) for more computationally intensive dynamic workloads. The small footprint of the adopted hypervisor (few lines of code) opens the possibility of a functional safety certification path following the ISO 26262 specification [6].

The integration of hypervisor, operating system and runtime will enable the HERCULES framework to provide predictable real-time guarantees for next-generation safety-critical applications, supported by a lightweight pragma-based application programming interface. Widely-adopted programming models for heterogenous architectures will be extended with real-time semantic constructs. The ultimate goal of the project is to enable parallel, non real-time and hard/soft real-time workloads to run side-by-side on the same platform, while preserving the required timing guarantees of safety-critical applications with different performance requirements.

References

[1] Google, inc., "Google Self-Driving Car Project." [Online]. Available: {https://www.google.com/selfdrivingcar/}

[2] NVIDIA, "The Tegra X1 Platform," 2015. [Online]. Available: http://www.nvidia.com/object/tegra-x1-processor.html

[3] Xilinx, Inc., , "The Xlinx Ultrascale Architecture." [Online]. Available: {http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf}

[4] The Hercules Consortium, "Hercules – High-Performance Real-time Architectures for Low-Power Embedded Systems." [Online]. Available: http://hercules2020.eu/

[5] ARM Ltd., "The Big.LITTLE architecture." [Online]. Available: {https://www.arm.com/products/processors/technologies/biglittleprocessing.php}

[6] I. O. for Standardization / Technical Committee 22 (ISO/TC 22), "ISO/DIS 26262-1 - Road vehicles Functional safety," International Organization for Standardization / Technical Committee 22 (ISO/TC 22), Geneva, Switzerland, Tech. Rep., Jul. 2009.

[7] Gavin Kistner, "Developing Next Generation Human Machine Interfaces (HMI)." [Online]. Available: http://on-demand.gputechconf.com/siggraph/2013/presentation/SG3110-Developing-Generation-Human-Machine-Interfaces.pdf

[8] "OpenMP Application Program Interface v4," 2011. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP3.1.pdf

[9] NVIDIA Corporation, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, 2007.

[10] Kronos Group, "The OpenCL 1.1 Specifications," 2010. [Online]. Available: http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[11] The AUTOSAR Consortium, "AUTomotive Open System ARchitecture." [Online]. Available: {http://www.autosar.org/index.php}

[12] "OpenMP Application Program Interface v.3.1," 2009. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP3.1.pdf

[13] FSF - The GNU Project, "GOMP - An OpenMP implementation for GCC." [Online]. Available: http://gcc.gnu.org/projects/gomp/

[14] L. M. Pinho, V. Nélis, P. M. Yomsi, E. Quiñones, M. Bertogna, P. Burgio, A. Marongiu, C. Scordino, P. Gai, M. Ramponi, and M. Mardiak, "P-SOCRATES: A parallel software framework for time-critical many-core systems," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 39, no. 8, pp. 1190–1203, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2015.06.004

[15] D. Ferry, J. Li, M. Mahadevan, K. Agrawal, C. Gill, and C. Lu, "A real-time scheduling service for parallel tasks," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013 IEEE 19th*, April 2013, pp. 261–272.

[16] The Khronos Group. [Online]. Available: http://www.khronos.org/

[17] P. Gai, E. Bini, G. Lipari, M. D. Natale, and L. Abeni, "Architecture for a portable open source real time kernel environment," in *In Proceedings of the Second Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial*, 2000.

[18] Evidence srl, "ERIKA Enterprise kernel." [Online]. Available: http://www.erika-enterprise.com/

[19] Infineon Technologies AG, "Tricore AURIX Family." [Online]. Available: {http://www.infineon.com/cms/en/product/channel.html?channel=db3a30433727a44301372b2eefbb48d9&ic=0101033}

[20] S. Xi, M. Xu, C. Lu, L. T. X. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in xen," in *Proceedings of the 14th International Conference on Embedded Software*, ser. EMSOFT '14. New York, NY, USA: ACM, 2014, pp. 27:1–27:10. [Online]. Available: http://doi.acm.org/10.1145/2656045.2656066

[21] Siemens, "The Jailhouse Hypervisor." [Online]. Available: https://github.com/siemens/jailhouse

[22] U. Steinberg and B. Kauer, "NOVA: A Microhypervisor-based Secure Virtualization Architecture," in *Proceedings of the 5th European Conference on Computer Systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 209–222. [Online]. Available: http://doi.acm.org/10.1145/1755913.1755935

[23] NVIDIA, "NVIDIA GRID Remote Workstation Certifications." [Online]. Available: http://www.nvidia.com/content/cloud-computing/pdf/GRID-certification-microsite-v2.pdf

[24] Y. Sukuki, S. Kato, H. Yamada, and K. Kono, "GPUvm: why not virtualizing GPUs at the hypervisor?" in *2014 USENIC Annual Technical Conference*, 2014.

[25] P. Burgio, A. Marongiu, P. Valente, and M. Bertogna, "A memory-centric approach to enable timing-predictability within embedded many-core accelerators," in *Real-Time and Embedded Systems and Technologies (RTEST), 2015 CSI Symposium on*, Oct 2015, pp. 1–8.

[26] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, "A predictable execution model for COTS-based embedded systems," in *Proceedings of the 17th IEEE International Real-Time and Embedded Technology and Applications Symposium*, ser. RTAS '11, April 2011, pp. 269–279.