Microprocessors and Microsystems 38 (2014) 921–932

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

MultiPARTES: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems

Salvador Trujillo [a,*], Alfons Crespo [b], Alejandro Alonso [c], Jon Pérez [a]

[a] IK4-IKERLAN, Arrasate-Mondragon, Basque Country, Spain
[b] Universidad Politécnica de Valencia, Valencia, Spain
[c] Universidad Politécnica de Madrid, Madrid, Spain

ARTICLE INFO

Article history:
Received 20 October 2013
Revised 4 August 2014
Accepted 26 September 2014
Available online 5 October 2014

Keywords:
Mixed-criticality
Multi-core
Partitioning
Hardware virtualization
Certification
Temporal and spatial separation

ABSTRACT

The consumer market is continuously pushing for smarter, faster, more durable and cheaper products with ever more complex and sophisticated functionality. Other fields such as safety–critical and dependable applications are not unaware of these requirements, and even impose others (e.g. certification). In the current multi-core era, industry and research entities are facing the important challenge of fulfilling all these requirements, which often impose the necessity for integrating components with different levels of dependability in a single hardware platform. In this scenario, new concerns appear with respect to safety certification of the resulting mixed-criticality systems (e.g. temporal and spatial isolation). This article describes the research effort that is being conducted within the FP7 MultiPARTES project, which is one of the initiatives launched by the European Commission to explore new solutions for developing certifiable mixed-criticality systems using heterogeneous multi-cores. The article explains the proposed development toolset for such systems, presents a proof-of-concept implementation and shows its applicability in a real-world application that needs to be certified, namely a wind-power turbine.

1. Introduction

Currently there is an increasingly important trend for using mixed-criticality systems, where multiple components with different dependability, real-time and certification assurance levels (e.g. safety–critical and consumer functionality) are integrated into a shared computing platform [1]. The reasons behind the trend for mixed criticality are mainly non-functional: reducing costs, volume, weight and power consumption, and can be found in a multitude of different domains such as industrial control, airborne, automotive systems and space avionics, to cite only the most notable ones.

Certification is the process of issuing a certificate to indicate conformance with a standard, set of guidelines or some similar document. It is mandatory for some types of safety systems, whose failure may cause injury or death to human beings or important environmental damages. In order to meet the requirements imposed by major certification bodies (e.g. to establish fault containment in the shared computing platform and avoid unintended side-effects between the components that are being integrated), the use of mechanisms for temporal and spatial partitioning in mixed-criticality systems is mandatory. For example, it is of utmost importance to guarantee that a failure affecting the in-flight entertainment system on a plane does not affect the engine control system. Partitions must thus encapsulate system resources temporally (e.g. latency, jitter, duration of availability during a scheduled access) and spatially (e.g. preventing a partition from altering the code or private data of other partitions). This is precisely the purpose of the Integrated Modular Avionics (IMA) paradigm [2] and the architectural guidelines in the AUTOSAR automotive initiative [3].

Hence, mixed-criticality systems allow for dealing with ever greater complex designs, which can indeed be partitioned and developed separately and later integrated without having to consider all possible interactions that may occur among them, which give rise to most of the safety concerns and verification costs. This type of systems supports composability, as their applications can be assembled in various combinations, without the need to modify them. In addition, the target type of the systems is real-time, as their correctness depends not only on the validity of their outputs, but also on the time when they are produced.

At the same time mixed-criticality systems are proliferating, and computing platforms are migrating from single-core to

* Corresponding author. Tel.: +34 943712400; fax: +34 943796944.
E-mail address: strujillo@ikerlan.es (S. Trujillo).

http://dx.doi.org/10.1016/j.micpro.2014.09.004
0141-9331/

multi-core and, in the future, many-core architectures [4–7]. It is estimated that multi-cores will be used in about 45% of industrial applications by 2015, up to 95% of which will combine different mixed-criticality levels [8]. Multi-cores and many-cores open new opportunities to develop robust mixed-criticality systems at a competitive price, but they also create new challenges that must firstly be addressed. The fact is that most of the existing commercial multi-core processors have not been designed with a focus on hard real-time but on the maximal average performance, thus posing multiple temporal isolation challenges [9,10]. In order to guarantee the predictability required by highly critical applications, nowadays it is common to sacrifice most of the performance delivered by a multi-core processor and use only one of its cores [11]. Therefore, it becomes essential to develop new methods and techniques to enable the exploitation of the computing benefits offered by multi-cores, while coping with their associated complexity and particularities (e.g. the need for communication and synchronisation between cores). The European Commission has launched several research projects in the context of its FP7 and Artemis programs to come up with new solutions to this problem. These include MultiPARTES [12], RECOMP [13], ACROSS [14], CERTAINTY [15], parMERASA [16], T-CREST [17] and VIRTICAL [18].

This article focuses on the FP7 MultiPARTES project. It describes the most remarkable advances that have been made within this project, which range from a set of hardware and software methods for supporting the building of mixed-criticality systems with temporal and spatial separation to a methodology for easing the design of multi-core based mixed-criticality systems. At hardware level MultiPARTES uses a specifically designed heterogeneous multi-core platform that combines high-performance X86 cores and highly-reliable SPARC-LEON3 cores, interconnected by means of a predictable Time-Triggered Network-on-Chip (TTNoC) [19,20]. At software level, MultiPARTES relies on a virtualization layer, which is offered by the XtratuM [21], to guarantee safe and efficient sharing of the underlying hardware platform among a number of temporally and spatially separated partitions. The article is completed with a case study drawn from a real-world wind power application that illustrates the contributions brought about by MultiPARTES to the certification of mixed-criticality systems.

The remainder of this article is organised as follows. After summing up the key concepts and related work in Section 2, Section 3 outlines the MultiPARTES approach. Then, Sections 4 and 5 describe the MultiPARTES hardware platform and virtualization layer respectively, while Section 6 explains the proposed development toolset. Finally, Section 7 presents the wind power case study and Section 8 discusses the concluding remarks and future work.

2. Background and related work

This section introduces some important aspects related to mixed-criticality systems, with special focus on system partitioning and virtualization. In addition, it sums up the most significant advances in this field.

2.1. Hardware support for virtualization

Virtualization refers to the creation of a virtual machine or partition that acts like a real computer with operating system (OS), but executing the software applications separately from the underlying hardware resources. Among other objectives, virtualization is intended to support system partitioning and to protect the execution time and memory space of each application.

Poper and Goldberg introduced a set of requirements to support efficient virtualization as early as 1974 [22]. The most important requirement established that critical instructions that could affect the correctness of system functioning (e.g. change of sensitive registers and memory locations, such as a clock register or interrupt registers) should trap and pass control to the partitions, enabling them to 'emulate' the desired effect in the 'guest OSs'.

Today several processors provide an Instruction Set Architecture (ISA) which meets Poper and Goldberg's requirement, including Intel VT-X [23], AMD-V [24] and ARM TrustZone [25]. This hardware support for virtualization leads to reduced overhead and footprint of virtualization software and improved performance. Most current processors distinguish between user and privileged modes when executing instructions, and include a Memory Management Unit (MMU) with the dual objective of translating virtual addresses to physical addresses and preventing unwanted memory accesses. Some processor architectures, such as Intel VT-d, AMD-Vi and SPARC V8, also support device and Input/Output (I/O) virtualization. An I/O Memory Management Unit (IOMMU) enables partitions to directly use peripheral devices through DMA I/O bus and interrupt remapping. Besides, several hardware extensions for device virtualization improve networking and I/O throughput for partitions (e.g. PCI-SIG I/O, network virtualization of Intel VT-c, single-root I/O virtualization SR-IOV). However, the integration of I/O into an architecture with time and space partitioning remains a research challenge [26]. Examples of open issues are the impact of I/O activities on CPU and memory sharing, and safe and seamless communication across different on-chip and off-chip communication networks.

A number of virtualization techniques for on-chip and off-chip communication networks are available. Time-triggered networks use Time-Division Multiplexing (TDM) to establish virtual communication links where components cannot interfere with each other in the value and time domain. TTNoC [19,20] and Aethereal [27] are examples of on-chip networks for time and space partitioning. Examples of time-triggered communication protocols at the cluster level are TTEthenet and FlexRay [28]. A major challenge here is the seamless virtualization of resources at chip and cluster level. For example, the access to a remote I/O resource located on another chip should be made via gateways involving different on-chip and off-chip networks (i.e. vertical integration), and possibly gateways between different types of off-chip networks (i.e. horizontal integration).

2.2. Software support for partitioning

Software support for partitioning is provided by hypervisors. These are layers of software that exploit the features of the hardware platform to establish independent execution environments. There exist several virtualization solutions, including full virtualization, OS level virtualization, OS virtualization support and bare metal hypervisors.

Full virtualization relies on the processor's virtualization hardware to trap critical instructions and 'emulate' their execution in the partitions. The advantages of this method are that guest OSs are not modified, thus supporting both mono-core and multi-core OSs. On the other hand, the disadvantages associated to this method are low performance, absence of communication mechanisms between partitions, and lack of any scheduling policies and guarantees for real-time. Examples of full virtualization solutions are VMWare Server, Virtual Box and Qemu.

OS level virtualization allows achieving an adequate performance, but it does not offer support for simultaneous execution of multiple OSs, nor for real-time performance. Further, only Linux distributions are supported. Linux VServer, Solaris Zones & Containers, FreeVPS and openVZ are examples of solutions using OS level virtualization.