© 2016 IJSRSET | Volume 2 | Issue 5 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

An Irreversible Transition towards Multicore Platform in Safety-Critical Domain for the Aviation Industries

Nagalakshmi K*[1], Gomathi N[2]

*[1]Computer Science and Engineering, Hindusthan Institute of Technology, Coimbatore, Tamilnadu, India

[2]Computer Science and Engineering, Vel Tech Dr.RR & Dr.SR Technical University, Chennai, Tamilnadu, India

ABSTRACT

In modern safety-related application domains, the shift from unicore to multicore processors is becoming inevitable to keep pace with the growing importance of computational capacity and to satisfy the functional consolidation trend while decreasing energy consumption and thermal hotspots. Nevertheless, typical multicore processors are mostly intended to enhance the system performance, whereas safety-critical systems (SCS) have very different demands in terms of safety, reliability, quality of service, predictability and timing correctness. Hence, the move towards multicore processors imposes many significant challenges the computing industry has to tackle. These challenges are involved in designing of certifiable multicore architectures, the organization of common resources and assimilation of concurrent software. Hence, these are encountered at all phases of the specification, design, development, testing, and certification processes. Hence, both multicore industrialists and the real-time community have to fill the gap to meet the requirements enforced by SCS. The objective of this paper is to initiate such a discussion as an effort to fill the gap between the two domains and to substantially increase the cognizance of the obstacles and issues that need to be handled in the safety-critical domain.

Keywords: Avionics System, Mixed-Criticality, Multicore Processor, Safety-Critical

I. INTRODUCTION

A system is considered as being safety-critical whose failure might endanger human life or create significant impairment to property/environment. Almost all the safety-critical applications are mixed-criticality (MC) systems which co-host multiple applications with different degrees of criticality (i.e., importance) into a shared hardware platform. Inevitably, such a high integration trend has been witnessed in numerous industrial sectors (e.g., automotive, medical, aerospace, nuclear power station, etc.). The functional consolidation and radically increased computing demands are key enablers to driving chip manufacturers towards multicore architectures in SCS. A good example of MC system is the Unmanned Aerial Vehicles (UAV), commonly called as pilotless airplanes or drones [1]. The pilotless airplane is a remotely controlled aerial vehicle and finding novel applications in military operations, traffic surveillance, wilderness search and rescue, precision agriculture, fire control and many others. It will have warfare and reconnaissance facilities surpassing those of presently used manned aircraft.

Based on a market survey from an independent business information provider for military application, the global aviation electronics market for appropriate safety products in 2009 reaches US$ 5.1 billion. Over the projected period of 2010 - 2020, the cumulative drone markets will almost US$ 71 billion [2]. Expected profits over the epoch of 2010 - 2015 are projected to be nearly US$ 62 billion with estimated US$ 5.5 billion expended worldwide in 2010 alone [3, 4]. In Australia, civilian usage of drones has been encouraging with AUS 2 million went to field surveys where 90% of 500000 hectares crop monitoring is covered by pilotless aircraft [5].

With the rising demand across a wide range of applications, drones with varying dimensions and capabilities have been designed based on the application demand and with economic consideration. Due to the cost and performance benefits of MC system in microelectronics industries, the arena of general-purpose and embedded computing shows an extraordinary trend towards multicore architectures. Currently, multicore processors are becoming an unavoidable choice in SCS [6] as exemplified by the current and ongoing projects, such as Quest-V [7], Framework Programme (FP7) funded parMERASA [8], CERTAINTY [9], P-SOCRATES [10] projects and ARTEMIS funded EMC^2 [11] project.

Drone assimilates tasks (i.e., workloads) of varying degrees of rigorousness and performs them onto a common embedded system. The computational workloads of UAV can be categorized into three classes:

1. Safety-oriented workloads: High-level workloads that perform safety relevant activities of the vehicle, such as flight cruise control and trajectory planning to maintain the stability of the vehicle, dropping which a UAV cannot be operated safely. The malfunctions of these workloads can lead catastrophic effects for the vehicle, and therefore must be performed with the maximum level of guarantee.
2. Mission-oriented workloads: Low-level workloads, which are concerned with reconnaissance missions such as localization of targets, direction-finding operations and parking assistance, dropping which the vehicle is still deemed secure. Failure (i.e., timing overruns) on these workloads leads slight service interruption in the system that is not fatal.
3. Non-critical workloads: Workloads that perform the least significant background functions, and they do not affect the safety of the drone. An example would be determining an optimal route through an area under unfriendly radar surveillance.

Criticality is the level of necessary fortification against the malfunction of computational subsystems. It is measured in terms of safety metrics such as DAL (Design Assurance Levels) or SIL (Safety Integrity Levels) in the avionics industry and ASIL (Automotive Safety Integrity Levels) in the automotive industry. These SILs reveal the necessary level of risk reduction in scheming safety relevant applications and hence involve various steps of designing, implementing, testing, and certification processes. For instance, in aviation standard DO-I78B, there are five DALs, categorized by their degree of threatening produced by the failure of the workload: catastrophic; hazardous; major; minor; no effect [12]. Workload with higher DAL designates that a greater risk reduction is required for the functional and timing correctness of the chip’s operation. For example, in the control system of a UAV executing surveillance task, it is more important to validate the correctness of safety-oriented functionalities such that the aircraft does not crash, than for mission-oriented functions like capturing and transferring photos.

During the engineering of a SCS, the risks are detected, their rigorousness is investigated, and suitable hazard control techniques are implemented to decrease jeopardy to an acceptable level. Traditionally, safety-critical functionalities have to be validated based on their confidence levels by different Certification Authorities (CA) (e.g., Federal Aviation Authority (FAA) in US and the European Aviation Safety Agency (EASA) in Europe for airborne systems [13]. Certification is an industrial practice for ensuring the correctness of the critical components against stringent safety standards. In order to perform risk analysis, different standards are available in the avionics system. ARP 4754 [14] defines system level certification features of aggressively consolidated or sophisticated systems implemented in airplanes. ARP 4754 considers the entire environment it will operate in and assorted levels of criticality. This standard dictates hardware/software design methodologies and timing analysis tools for airborne systems. DO-254 [15] defines electronic hardware certification imperatives. A complete description of the guidelines for software developers can be found in DO-178B [16].

In order to guarantee system safety, CA dictates far more pessimistic assumptions regarding the execution of the system, which are very unlikely to befall in practice – their sole concern is with the correctness of the safety-related component of the vehicle. It is not important for them whether the mission-oriented tasks are executed in time or not. On the other hand, the complete system, including non-critical components, must be validated by the system engineers or other regulatory organizations, who usually exploit a less severe safety standard than the one dictated by aviation authorities [17].

Modern real-time applications have rigorous timing and safety requirements, and the adoption of multicore processors raises even greater challenges. We outline two major reasons. First, the functional consolidation ensuing undue interferences across parallel tasks in the integrated platform [18] that may lead to malfunctions of the system and persuade impulsive overloads, which reveal further latencies perhaps, violates the real-time constraints of the workloads. Second, multicore processors with commercial off-the-shelf (COTS) subsystems are designed to enhance the average system performance and not the worst-case execution behavior. This presents various uncertainties and makes the common cause failure analysis of such systems highly problematic, otherwise impractical. Standard multicore processors remain very attractive for use since they are reasonably priced. Excessively limiting such systems to perform in the worst case will increase determinism substantially, but there is a simultaneous restriction of the potential advantage.

The progression of embedded field is increasing from “federated” to “integrated” and “partitioned” architectures. In traditional federated architectures, each core or processing element (PE) executes at most one task, and the applications are loosely related [19]. As more and more workloads are executed concurrently using SCS, the number of such PEs increased (e.g., modern premium vehicles comprise over 70 -100 PEs), alongside with their cost, the number of physical components (e.g., wires and connectors) and SWaP (size, weight and power) concerns. A method to decrease the SWaP of the system is realizing the system by means of an integrated platform, by consolidating more and more tasks onto the same core [20].

In order to enable strict isolation between tasks of different criticality levels, embedded system architects are relying on partitioned architectures, which deliver partitioning tools at the platform level. Separation (Partitioning) is the fundamental notion to circumvent any interference among diverse applications in space and time. It delivers risk containment equivalent to a flawless system in which each partition is assigned to a predefined core and concomitant I/O devices, and each inter-partition interaction is carried on dedicated lines [21]. Compared to the integrated architecture, the partitioned architecture provides sufficient isolation among workloads of different SILs, thus the workloads can be generated and certified based on their initial SIL, decreasing the costs. Isolation can be achieved in ways: spatial and temporal. The temporal isolation ensures that a task scheduled to common resources cannot be affected by an application in another partition [21]. Spatial isolation ensures that an application in a partition will not interfere with the code and data of another partition [21]. However, the effect of functional integration in multicore platforms is not completely investigated yet [22].

The major objective of our work is not to give a complete description of safety-critical or multicore systems, but rather desire to investigate the integration scenario of them. In this paper, we address new safety relevant challenges exist for critical domains when considering the adoption of multicore systems, with a focus on aerospace systems. First, in Section II, we review some examples of commercial multicore architectures, the openings, and the issues threw down by multicore processors in the aerospace industries, where diverse functionalities can benefit from the functional consolidation on an integrated platform. We present SYSGO’s PikeOS [23] and eSOL’s eMCOS with their application, specific demands including certification, performance and possible methods of consolidation in Section III. Next, we demonstrate how the challenges ensuing from multicore systems can be handled in software by regulating the characteristics of a Real-Time Operating System (RTOS). In Section IV, we present a representative multicore architecture, Kalray MPPA-256 Bostan processor [24], which is widely used in the avionics system. We address the design, employment, and programming models have been intended to execute MC applications. Section V provides a comprehensive study of an ongoing P-SOCRATES project. Finally, we conclude this paper in Section VI.

II. METHODS AND MATERIAL

1. **Analyzing Multicores In SCS**

In this section, we first present some examples of commercial multicore architectures for SCS and then identify challenges along with new opportunities regarding the safety of the system. From a technical perspective, multicore processors can deliver two valuable benefits in the context of safety-related applications: (i) achieve higher processor/ resource utilization, and (ii) provide performance guarantees to highly critical tasks without degrading the performance of other non-critical tasks. However, the current industrial practice for the exploitation of multicore architectures is inadequate and is still being examined, since numerous crucial problems have to be solved.

A. Examples of Multicore architectures for Safety-critical Systems

Current critical domains integrate tens of computing cores to achieve secure and efficient control systems. A number of research projects like EMC^2, CERTAINTY and P-SOCRATES use multicore processors for executing safety-critical applications. The FP7 funding project parMERASA (Multi-Core Execution of Parallelized Hard Real-Time Applications Supporting Analysability) [8] explores different parallelization techniques, OS virtualization and competent synchronization techniques for safety-related environments and fabricating a timing analysable multicore platform with up to 64 cores [25]. In parMERASA, the common resources are allotted in favour of Hard Real-Time (HRT) applications over Soft Real-Time (SRT) applications [8]. Even though SRT tasks can miss the deadlines occasionally, the system is deemed safe but with degraded performance guarantees. In HRT systems, missing a deadline of the tasks can cause failure of the system. An HRT application has its own dedicated cache and is assigned maximum priority when accessing system resources. When executing SRT in a multicore environment, the maximum access time of HRT task to common resources is limited. Nowotsch et al. assessed the performance of multicore processors in an aerospace domain and explained the possible interferences to execution from parallel workloads or simultaneous access to common hardware resources [26]. They exploit a spatial isolation technique for the cache and main memory and temporal isolation for the communication bus. However, these methods cannot separate on-chip traffic without incurring considerable overheads.

Several safety-relevant issues can be solved by the emerging multicore designs. This is because, in a multicore platform, a number of safety features (e.g., resource redundancy, partitioning, etc.,) can be applied instinctively. All multicore chip vendors are aware of this. They are promoting their devices with dedicated architectures ensuring the demands of safety standards. For example, Freescale and Intel provide software as well as hardware products for safety-critical systems. Similarly, Wind River [27] delivers multicore software solution, which includes their support for concurrent programming, virtualization, software hypervisor, VxWorks Cert and Workbench.

MPC564xL, a dual-core embedded system introduced by Freescale, is primarily designed for automobiles. Recently, it can also be applied in safety-related applications in avionics systems. It comprises of several safety-related properties. MPC564xL is targeted towards IEC 61508 international industrial standard specifies four SILs for the safety of electric, electronic and programmable electronic (E/E/PE) [28] instruments and devices. It targets to define a set of measures for fault avoidance and failure protection for the certification process. An obvious example is in-built Error Correction Codes (ECC). MPC564xL contains two PowerPC cores and enable either lockstep mode or asymmetric processing mode. The former mode enables the widespread use of hardware redundancy, while the later operating mode makes the software diversity conceivable. It is obvious that this processor along with its inherent safety-related functions is an exciting architecture. Nevertheless, a comprehensive investigation is required to tap the full potential of this architecture in the avionics systems.

Blackfin, a dual-core processor introduced by Analog Devices, is widely used in real-time communication technologies. Since this architecture is not explicitly promoting the processor for SCS, it is not emphasized on common cause failures. The cores of this architecture are homogeneous and share the same computing platform except for the level 1cache (L1). While it is possible to use various OS's on each core, from a safety perspective, the diversity is not present. Malfunction of certain components (e.g., power supply, memory management unit, communication bus, etc.,) make both cores unable to execute tasks. This is susceptible to the single point of failures. An important feature of this architecture is the ability of one core to monitor or validate the correctness of the other core. This could be done by executing the same workload on both cores and one core compares the results and identifies the fault of another core at appropriate instances of its execution.

Furthermore, one core is dedicated to run memory testing procedures or other self-diagnostic programs. In critical domains, which involve higher performance, this architecture would be inspiring to assess. If complex and time-consuming diagnostics like online memory diagnostic could be offloaded, this architecture is most appropriated time-critical applications.

At present, the Kalray MPPA-256 Bostan processor is well adapted to aviation electronics. The increasing trend towards certified aircraft functionalities executed on MPPA-256 (Multi-Purpose Processor Array -256) architecture relies on spatial isolation technique that may be implemented over hundreds of PEs. In this architecture, the CC (compute cluster) provides a natural boundary to evade non-intended interactions across tasks at various confidence levels. This processor also facilitates dynamic barrier synchronizations between PEs, whether located within the same cluster or between various clusters. We will explain this architecture in more detail later in section IV.

B. Opportunities

Functional integration in aviation industries (e.g., Integrated Modular Avionics (IMA) specifications [29], and the growing requirements for additional computational capacity are opening new possibilities for innovative research works on multicore systems. As spacecraft platforms are responsible for different activities such as direction-finding, stability control, data communications, air traffic control, passenger entertainment, etc., their computational complexity remains increasing. Hence, multicore processors can be utilized to focus several independent tasks on an integrated platform or to execute tasks demanding for maximum computational efficiency. Indeed, a combination of scenarios is also feasible.

1) Hosting various functionalities within an integrated platform:

Integrating several tasks onto a single, secure computing platform allow us to map each task to a predefined core. The valuable benefits of functional integration are as follows: (i) Power consumption and heat dissipation limitations are driving manufacturers towards multicore devices. The power consumption of a single multicore platform is usually lower than that of multiple unicore solutions. In a multicore platform, only one power supply unit is efficient enough to fulfil the requirements of the system's power budget as compared to multiple power supply units for multiple unicore systems; (ii) lower power density directly leads to lower cooling requirements, which is a salient feature of the aerospace system; (iii) decreasing the number of PEs, supply units and cooling equipment evidently reduces the SWaP requirements of the system architecture; and (iv) this, in line, leads to a reduced amount of fuel ingestion (i.e. less fuel has to be lifted at flight take-off), and eventually leads to significant economic gains.

2) Ever-growing Demand for Computing Performance:

Performance is the motivating force behind computing technologies. Tasks with increasing computational necessities can benefit from multicore technologies by dividing a complicated application into independent tasks being performed concurrently on various cores. These performance demands could only be realized by highly sophisticated unicore processors with increasing demands on power supply and cooling. Regrettably, these sophisticated unicore processors do not fulfil the DO-254 specification [30] and hence cannot be used. System engineers select the preferred design alternative, a single-chip multicore containing a higher number of less sophisticated cores, to provide such performance. This entails extra effort to build parallelization techniques, compilers and analysis tools for safety-critical applications. Multicore processors can deliver their superior performance only if an appropriate parallelization is performed.

3) Supporting time-predictability:

Time constraints and parameters associated with the data processing phases (i.e., acquisition, computation, forwarding, buffering, synchronization and delivery) are used to specify the time-critical tasks. The major potentials anticipated from a multicore system for time-critical computing and stated by the safety standards are determinism, predictability and composability of the performed computations. The graceful degradation of timing features and maximum utilization of computing resources are deemed other quality metrics for such systems. In multicore environments, interactions among tasks avert the system to be composable, predictable or even deterministic. The timing predictability cannot be guaranteed whenever cores contend to access common hardware means, e.g., system buses, global clocks, shared RAM and I/O peripherals. Owing to resource sharing, the task assigned to one core can interfere with the task executing on other core, albeit both tasks are autonomous. The inter-core interference can cause not only execution delay but can also make execution behavior less deterministic. Examples of interference channels include (i) peripherals, L2 caches and internal buses shared between cores (see Figure 1); (ii) Direct Memory Access (DMA) controller may interrupt the communication buses asynchronously; (iii) L1 caches may need to be preserved in a steady state by means of cache coherency protocol, (i.e., although L1 caches belong to different cores, they are not autonomous); and (iv) pipelines are shared between cores for hyper threading processors.

Figure 1 shows a quad-core architecture analogous to NXP QorIQ T2080 [31] architecture which is commonly used in avionics systems. In this configuration, each PE has its own pipeline and L1 cache (L1$). It also consists of shared on-chip L2 caches (L2$). Though the cores have the multi-threading capability, most of the aerospace systems do not utilize this facility. Therefore, an application assigned to a core has private access to L1$. The common resources are L2$, Memory Controller (MC) and NoC. Common resources are accessible by all the cores and fine-grain hardware techniques such as placement, fetch and replacement policies of cache memories, define in what way they are accessed. Even though such techniques work sound in practice, they are extremely unpredictable and do not assure required system behavior for safety-related applications.

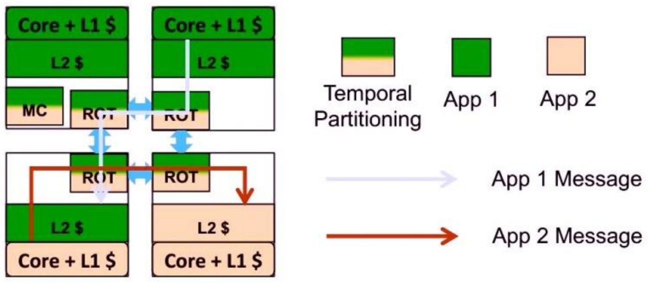

Figure 1: Interference Channels in Multicore Platform

Recent multicore processors pose several issues that have to be tackled in advance. Most of them are related with highly consolidated system-on-chips (SoC) that comprise many I/O devices. Many certification methods and processes exploit only a small portion of these resources. Hence, how can we ensure that idle resources do not interrupt the system (e.g., "babbling idiot" failure)? How can we ensure that idle PEs do not have any action on the buses? Furthermore, some performance enhancing techniques (e.g., speculations, caching policies, memory pre-fetching techniques and so on) cause non-deterministic characteristics of system behavior. In an SCS this can cause may cause tasks to violate their timing constraints resulting in malfunction of the system and in the worst-case behavior the potential loss of human lives or equipment. The major problem of using multicore architectures in SCS is ensuring the timing correctness of the system. If the hardware and software are extremely constrained to impose the worst-case execution behavior, then it makes the application less efficient on the multicore platform than on the unicore processor.

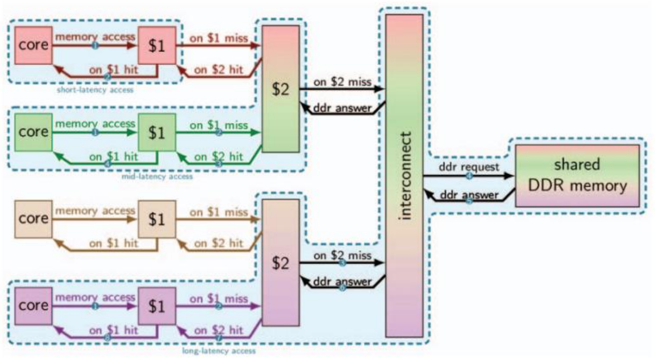

Figure 2 : Multicore architecture with cache coherent memory.

Important issues in multicore processors are memory hierarchies, coherency and latencies. The memory hierarchy of a traditional multicore system with a common memory and cache-based architecture is shown in Figure 2. A series of caches are traversed before accessing the common DDR memory. Even though the L1caches are completely owned by their corresponding cores, their data is often unpredictable and so is their timing behavior. Hence, the concurrent access of the cores to the same resources can cause unanticipated interferences. The key benefit of traditional multicore systems is the potential to run workstation and server software with a minute or no changes/advances.

An example memory hierarchy of a multicore system appropriate for time-critical embedded systems (e.g., Digital Signal Processors (DSP)) is given in Figure 3. In this configuration, the first level of the hierarchy consists of private SRAM with cache lockdown features. A Large amount of data transfers is delegated to DMA controllers, whose computational time can be constrained more precisely than general-purpose PEs. Interference arises in the components of the interconnection network and the DDR memory controller that are accessed by multiple applications. Such systems need significant reconfiguration to accommodate the bounded storage capacity of local memories and the usage of DMA controllers.

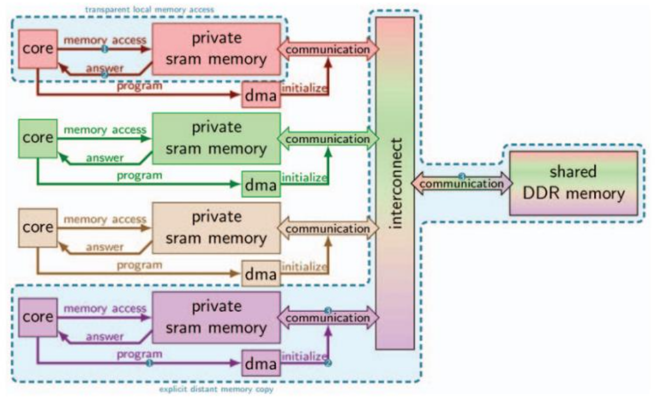

Figure 3: Multicore architecture with local memories

C. Challenges

As stated in the preceding section, non-intended interactions onto a common platform can have a strong influence on predictability and the dictated composable behaviour of safety-oriented applications. At present, both EASA [32] and FAA [33] enable SCS developers to use a dual-core processor for performing a single application. Clearly, this restricts the potential advantage of multicore processors. To fill this gap, lots of expertise is essential to have a complete knowledge about all the timing behaviours of platforms to make them appropriate for critical domains. This effort should be shared among the hardware manufacturers, the tool vendors and the application designers. In this section, we discuss the major challenges that should be tackled at the hardware as well as software level.

1) Architectural properties of the of Cores:

A significant dispute that is frequently ignored in the context of the multicore environment is the architectural properties of each core. As safety-oriented components strive to satisfy the most stringent constraints, the processors used must deliver a temporally deterministic behaviour. This implies that cores exhibiting timing anomalies, dynamic branch prediction, or sophisticated caching policies make WCET analysis more complex. Therefore, cores fabricated with such features do not build a desirable foundation for multicore platforms in the aerospace system. However, multicore processor containing cores with lower complexity as compared to general purpose high-performance computing are classified as highly sophisticated COTS devices [32]. In most cases, this classification makes certification process incredible regarding cost and time.

2) Interconnection Network (Int.Net):

In a multicore system, several PEs are connected using interconnection network such as TTEthernet protocol [34]. The Int.Net must also be responsible for guaranteeing predictable system behaviour. Moreover, facilitating higher bandwidth and lower execution delay are not efficient enough as far as no assurances on the correctness of the safety-critical part can be given. The specified assurances need to reflect the actual execution behaviour as precisely as possible. Excessive assurances will lead to higher overheads, which in turn adversely affect the performance of the multicore systems.

3) Fault management model:

The functional safety of the system demands several methods for enhancing reliability or at least to identify the timing overruns. Existing multicore processors are already fortified with ECC techniques to access an external RAM, Flash and internal memories; however, most of them are inappropriate for future avionics systems because their complexity leads to further risks. As multicore processors are designed to satisfy as many use cases as possible, they have several configuration registers. Error in a single configuration bit, ensuing from a Single Event Upset (SEU), may strongly affect the behaviour of the application. For instance, if the configuration of the access policy to the Int.Net varies, the timing properties of all tasks executing on the platform will be affected since the Int.Net is not operating correctly any longer. The timing overruns cannot be determined by WCET analysis. In unicore systems, faults can be detected by various redundant subsystems with compare stages before the results leave the system. Since unicore processors execute applications serially and in the same sequence on redundant components, comparing the result is quite easy to realize. However, the multicore processor can yield diverse results based on the possible interleaving of concurrent workloads. An appropriate compare unit is required to manage these results accordingly, which increases the system complexity.

4) I/O Devices:

Generally, I/O devices can be accessed by any active component within the system. This access can be intentional or occur inadvertently, for example, because of misbehaviour of a low-criticality task or a Single Event Upset in the Int.Net. Therefore, procedures for protecting I/O peripherals from inadvertent access and techniques are required for controlling/ synchronizing the envisioned accesses.

5) Programming tools:

Current programming models are not adequately flexible, reliable and scalable to be capable of tackling the growing size and heterogeneity of the multicore platform. Development of new programming models for critical unicore processors already needs special cognizance and practical experience. Besides this knowledge, designing critical concurrent software requires additional knowledge about what happens in a multicore system and what can happen inadvertently. Particularly, synchronization of workloads, the location of code and data, updating data and the utilization of I/O peripherals needs extra effort. In the average case, violating timing guarantees may yield inefficient systems; but in the worst case, it can also cause catastrophic consequences. Recently, appropriate synchronization mechanisms have been predominantly studied (e.g., parMERASA project [8]). Special care has to be taken when the multicore processor dealing with special micro coded routines. These routines can be complex instructions or I/O activities that execute expedient functions such as testing a subsystem in an efficient way, which would otherwise be impossible to test.

6) Software analysis

Running several workloads from the same or diverse application simultaneously raises a new query that is not answered by conventional software testing tools: what happens in parallel? Although running various partitions needs strict isolation, applications need to access peripherals frequently. If these peripherals are shared by several applications, it must be assured that no malfunction happens when a peripheral is locked by another application. Perhaps more importantly, in the case of concurrent execution, it must be assured that no race conditions can happen and that induced synchronization operates correctly. Furthermore, synchronization overhead and waiting times need to be considered during the WCET analysis.

2. Certified RTOS for Safety-Critical Embedded Platform

a. Necessity of certified RTOS in a safety-critical domain

Due to the familiarization of multicore hardware/software designs and new techniques such as virtualization and partitioning, the embedded system designers have more design-freedom in fabricating their systems. Since several safety-critical applications also use an embedded RTOS, it has become inevitable that critical subsystems with an RTOS are being designed to follow safety standards. The RTOS schedules the tasks of software as well as the functionalities of safety monitors and safety functions. The real-time software developers already provide certified RTOS (e.g., INTEGRITY-178B RTOS, VxWork Cert, or SCIOPTA's RTOS) that are initially certified to the industrial safety standard DO-178B. Since then, it is routinely re-certified to IEC 61508 on several compiler/processor combinations. The key benefits of using a certified RTOS are as follows: (i) it will reduce risk, cost and time-to-market of safety-critical products; (ii) it will provide a safety guidebook, which gives guidance on in what way to use the OS efficiently to ensure safety; and (iii) it will deliver some remarkable methods like memory protection that will make it simple to embed safety features into application.

b. Pike OS

Isolation is a significant requirement for safety-related applications, especially as we assimilate several tasks with diverse criticalities in a common platform, we would like to separate them according to their importance levels. In a multicore platform, this isolation is really difficult to realize since the hardware components are not sovereign. An imperative notion required to simplify the design and certification is a severe and vigorous partitioning, i.e. a set of principles applied to hardware/software resources which thwart interactions among tasks, exactly as if each task executes on its own virtual resources with assured performances whatever the execution of other tasks. Hence, we need the support of the OS to provide architects sufficient flexibility to the system configuration whilst coercing the architecture to enable the certification process. PikeOS is ARINC-653 compatible OS; it delivers complete partitioning in both time and space for several tasks executing with various importance levels.

i. Key Principles of PikeOS:

In a multicore platform, an efficient scheduling algorithm is required to implement not only to exploit software budgets effectively but also to reduce the interferences among concurrent application. These challenges are the key enabler to SYSGOs engineers to develop PikeOS [23]. The PikeOS is a new and prominent para-virtualization RTOS based on partitioning microkernel architecture for forthcoming avionics systems. As stated earlier, the progression of embedded system from federated to integrated architectures has led to environments where several tasks with different confidence levels share a common processing element (PE) or node. In order to circumvent any unintended interferences among these tasks the system needs to facilitate strict partition.

ARINC-653 [35] is a well-known example of partitioned communication model dictating firm isolation in the context of safety applications. The ARINC-653 defines the baseline-operating environment with strong time and space separations. It imposes time and space constraints statically before execution of each partition. Each separation is a distinct application with a devoted memory region, thus providing spatial separation. With time separations, each task is allowed to execute only within specified time budget or partition slices, assigned to each core. An additional OS layer named Virtual Machine Monitor (VMM) or hypervisor guarantees the partitioning among various separations. VMM is a software layer (or a mixture of hardware/software) that enables different OS's to run on the same processor as shown in Figure 4. The function of the VMM is to virtualize the existing system resources effectively. Furthermore, the VMM has to be certified at the highest SIL.

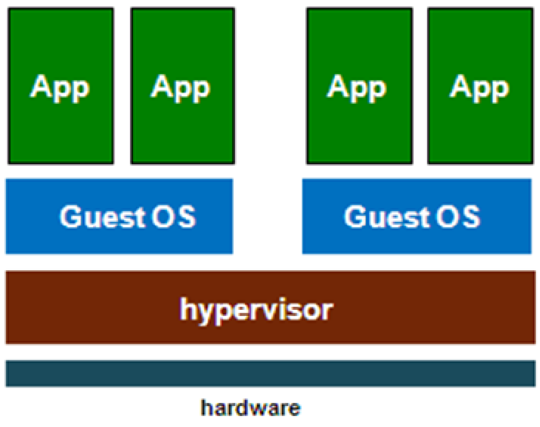

Figure 4: ARINC-653 Partitioned environment

PikeOS provides Many Independent Levels of Security (MILS) to embedded avionics systems, and its separations allow the safe and secure integration of large range of personalities of the market [23], including guest OS, Run-Time Environments (RTE), APIs, Native, Linux, ARINC-653, POSIX, Android, real-time Java, and others as shown in Figure 5.

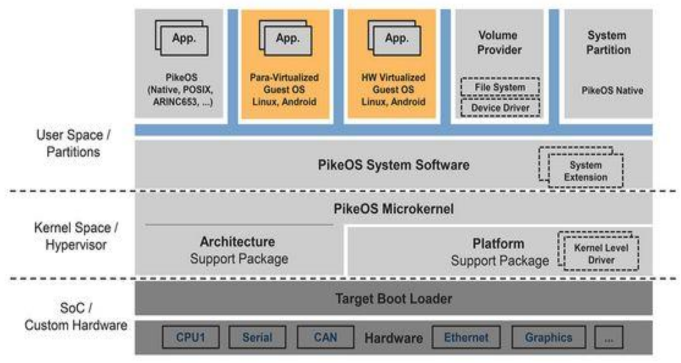

Figure 5: Architecture of PikeOS

The hypervisor model of this RTOS incorporates guests within a virtual machine (VM) with their dedicated memory region, computing resources and application set. Programs integrated on one VM perform absolutely independent of those in other machines. PikeOS has the facility to distinct code both spatially and temporally so system engineers have the capability to control not only where the code is stored but also when it executes. PikeOS and its associated software form a thin layer of the trusted code that virtualizes the critical components of the application to generate several isolated separations. The VMM is responsible for handling the physical resources of systems, and implement the spatial and temporal partitioning of the guests. Direct access to the native hardware is not endorsed.

ii. Hardware support for PikeOS:

Many OS, particularly those in avionics systems, still does not directly impose any partitioning techniques due to its excessively slow software emulation. In such cases, it is possible to program hardware devices to implement the required partitioning. For example, the OS enables Memory Management Units (MMUs) to provide access to a specified memory region, and the hardware is responsible for implementing that. Else, an exception is upraised and the operating system regains control. Reasonable exploitation of hardware support for partitioning is definitely most efficient with respect to performance and deterministic execution. Regrettably, most of the recent hardware does not support that. A prime example of hardware support is the evolving Input–Output Memory Management Unit (IOMMU) [36] implemented to DMA-capable I/O devices. Another example of limited support is virtualized PCI (Peripheral Component Interconnect) drivers [37] providing spatial isolation, however by exploiting the same PCI bus, there still exists interference impacts on the remaining parts of the system.

In Graphics Processing Units (GPU), we are still at the beginning since current hardware has no facility to reduce the processing time of shaders for, in order to partition rendering time in the GPU. Finally, access to peripherals can be virtualized by virtualizing software drivers. They provide flexible configuration and separation potentials, and for various devices, they are sufficiently fast. PikeOS exploits this method frequently for Ethernet devices. If hardware partitioning support does not exist, one approach may be to evade dependency by exploiting other partitioning resources, e.g., if direct bus bandwidth isolation does not exist, then the tasks may be isolated temporally so that no interaction can befall. It is essential for an OS to deliver the flexibility of the configuration for arranging a system in the way the designer realizes fit.

iii. PikeOS Separation Kernel for Multicore platforms

PikeOS is a separation micro-kernel that supports spatial and temporal isolation, where each separation in the system can be statically allocated to space and time constraints, ensured by the OS at run-time. Spatial isolation includes memory and peripheral access. This is applied by means of the existing MMUs of the cores. PikeOS delivers fine-grained memory management usage for the separations in the platform. PikeOS derives the concept of temporal isolation from ARINC 653 separation standards. A dedicated hardware switches the control back to the OS at defined synchronization points so that the OS can execute the separation.

During a separation process, executing functions achieve exclusive access to the common means. This needs all the synchronized cores to have a similar perception of time or synchronization with a dedicated protocol in case of asynchronous modes. In addition to the temporal isolation, this supports fine-grained control about what partition runs on which core at what time. This makes it feasible to reason about partitioning, even in a multicore environment.

For full control of the utilization of the processor, PikeOS provides a flexible methodology to the user who can choose an execution model ranging from asymmetric multi-processing to symmetric multi-processing [23]. The Asymmetric Multi-Processing (AMP) model is analogous to multiple unicore environments. There is a strong tendency for more interference channels among cores owing to the tight coupling of the cores. In this model, various cores or set of cores operated by various instances of an RTOS, possibly even different processors. It is also possible that the RTOS's could also be different resulting in partitioning being disseminated across various RTOSes. In this model, execution is absolutely asynchronous since each RTOS has distinct virtual address spaces so the MMU synchronization is not required. Memory spaces are only shared for dedicated regions for core interaction.

In a Symmetric Multi-Processing (SMP) model, a single OS controls all cores and partitioning process. The cores are tightly coupled via resource locks and synchronization mechanisms. The RTOS with SMP design needs to certify that there are no interference channels among partitions due to cross resource locking techniques. This idea is new for certification. Critical components of the time frame can perform in uniprocessor mode. In contrast to AMP, symmetric multi-processing model needs MMU synchronization, which will have some albeit a small hit on performance.
An illustration of a resource and temporal separation in PikeOS is given in Figure 6. Time partitions Tp_$_1$, Tp_$_2$ and Tp_$_3$, are scheduled to repeat at regular intervals. Resources partitions 1, 2, 3 and 4 are mapped to temporal separations and then cores are assigned to resource partitions. The lower picture demonstrates when and on which cores the partition will then be assigned. This way, partitioning can be realized via flexible configuration. This mode provides a simple, integrated platform where a single application can use multiple cores.

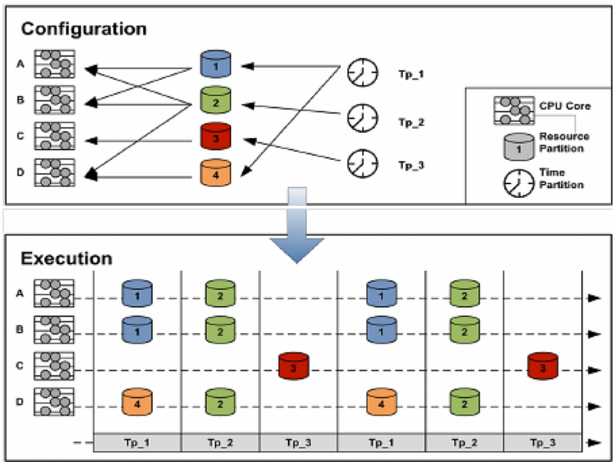

Figure 6 : Example of partitioning in PikeOS

Along with partitioning, the RTOS should provide an opportunity to refresh caches and other data structures (e.g., Translation Lookaside Buffer (TLBs)), at every time partition shifts. While this is typically extravagant, it will make software performing after refresh more predictably. In order to deal with DMA accesses that might interrupt the system, IOMMUs are now available. PikeOS provides support for these facilities to ensure partitioning of such devices, too.

In this study, we have also explored the techniques to alleviate partitioning complications. These include exploiting hardware support to bound bus utilization for some partitions, so as to make some assurances for other cores. The techniques are auspicious, though actual control of bus utilization time in hardware would be the decisive way to recover control of partitioning a bus. In order to certify, a safety-critical application, all the existing approaches are valuable. Only a cautious system design, including considerations about safety-critical guarantees, can be realized in the way the system needs it. This is made conceivable by flexible configuration joined with the implementation of the most recent hardware/software innovations for dealing the increased risk of interference.

c. eMCOS for a safety-critical environment

The eMCOS is the first viable manycore real-time OS for safety-critical platforms from eSOL in 2015. In an embedded market, eSOL is recognized as the leading vendor for RTOS, development tools and basic/application middleware. The noteworthy product of eSOL is the eT-Kernel RTOS, which is certified ISO 26262 and IEC 61508 at the maximum SIL. In contrast to any existing RTOS architecture, eMCOS can make the efficient utilization of manycore processor with tens or hundreds of different cores since it does not rely on any cache coherency protocol required by most currently available RTOSes.

eMCOS employs distributed micro-kernel architecture that is diverse from currently used RTOSes. The microkernel is fortified with only minimal utilities and is really compact, which enables it to run the MPPA I/O clusters as well as CCs. A micro-kernel is implemented in each core to provide fundamental services (e.g., inter-core communication, thread scheduling, thread migration, etc.,) (see Figure 7). The eMCOS for MPPA architectures also supports OpenMP 3.0 (Open Multi-Processing version 3). eSOL's semi-priority-based scheduling algorithm provides timeliness guarantees for safety-critical applications, which is always expected in embedded systems along with the superior throughput and scalability anticipated from multicore architectures. eMCOS allows the application developers to follow existing development styles since it implements the same programming frameworks and Application Programming Interfaces (API) like widely used RTOSes for unicore /multicore systems.

The eMCOS's scheduling algorithm exploits two different schedulers that work simultaneously. One scheduler satisfies the real-time guarantees by assigning higher priority threads to each core. These scheduled threads are always serviced first in order to for guarantee the timing constraints. Another scheduler allocates the remaining lower priority threads across all the cores based on their priorities. Obviously, this load distribution ability enables higher throughput.

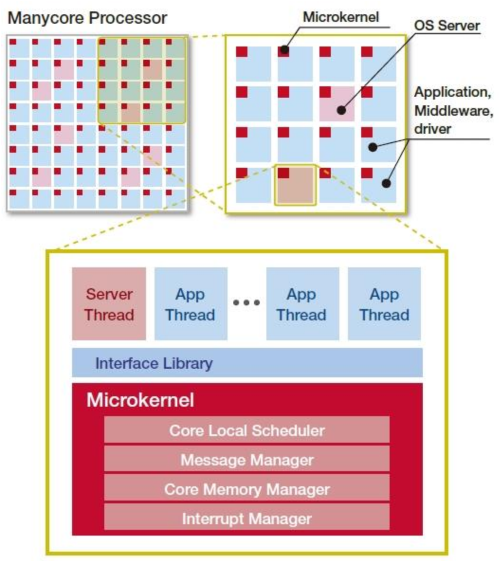

Figure 7 : eMCOS real-time thread management

III. RESULTS AND DISCUSSION

1. Example Multicore Architecture for Safety-Critical Domain

A. The Kalray MPPA-256 Bostan processor

The foundation of Kalray's supercomputing on a single-chip relies on its innovation, patented MPPA manycore architecture. This revolutionary architecture allows multiple cores to operate in parallel at high performance, low power and extremely low latency. It assimilates 288 homogeneous 32-bit/64-bit Very Long

Instruction Word (VLIW) computing nodes and 128 crypto co-processors on a die. More specifically, it consists of 256 application cores (or PEs), used to execute the user threads and 32 resource managers (RMs), and privileged to execute kernel routines. The computing nodes are disseminated across sixteen compute clusters and four I/O clusters. Each compute cluster comprises of 17 cores (i.e., 16 PEs + 1 RM). In addition to this, there are four quad-core I/O clusters, each having 4 RMs. The I/O clusters are in charge for communications with external devices, which act as controllers for the computing nodes. Each resource managers: (i) uses an RTEMS (Real-Time Executive for Multiprocessor Systems) OS; (ii) is provided with a common 512 KB 16-banked SDRAM; (iii) has its privileged 32 KB I-cache; and (iv) shares a 128 KB D-cache, which assures coherency among the computing nodes. The functions of RM include task management, communication control and data transfers between both external connectors (PCIe Gen3 8-lane interfaces) and SDRAM. For this purpose, resource managers have dedicated links to NoC interfaces. The Kalray MPPA-256 Bostan architecture is depicted in Figure 8.

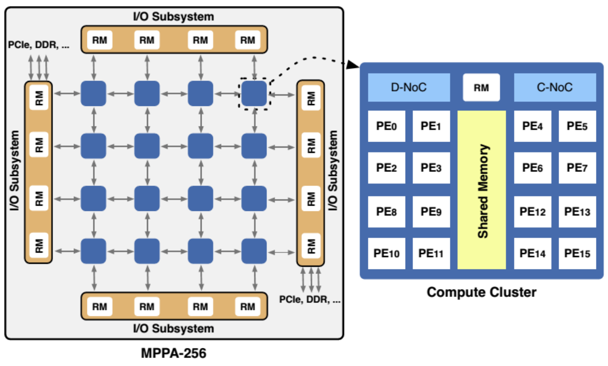

Figure 8 : Kalray MPPA-256 Bostan processor Architecture overview

The Kalray processor works between 400 MHz and 800 MHz and typically consumes 25W. Its maximum floating-point performances at 600 MHz are 634 GFLOPS / 316 GFLOPS for single/double precision correspondingly. Two memory controllers at 2133 MT/s deliver an external memory bandwidth of 34 Gbps. This memory hierarchy is energy-efficient and thwarts inter-cluster interferences, except for the explicit data transferring via NoC. The system also uses 2 PCIe connectors, 8 Ethernet 10 Gbps interfaces and direct access to the NoC in order to reduce processing delay. Applications developed for this architecture commence their execution on the I/O subsystems; consequently, it denies computation to the CCs through the network-on-chip interfaces. Communication with peripherals is achieved through several interfaces like PCIe connector and DDR3 channels.

B. Kalray software development Environments

The Kalray programming environment is composed of two parts, one dedicated to multicore programming and the other to manycore programming. On the I/O subsystems, the memory is directly accessible by the cluster cores; however, on the compute cluster, direct memory access by the cores must be emulated by a run-time system. One more fundamental dichotomy is that data caches are not coherent in the CCs, whereas each core within the I/O subsystems shares them. The multicore programming model consists of modern GCC/G++ compilers with OpenMP support, multithreaded GNU debugger and an optional Eclipse C/C++ Development Tools. A single process model with POSIX threads and timers is provided for each CC, limited to one thread per processing core. On the I/O subsystems, a Linux OS with dynamic loading and shared libraries are available, running on one of the quad-core CPUs. In the CC, the utilization of the OpenMP and P-Threads ensure that caches are coherent at synchronization points.

The manycore software development environments explore the features of the programming model in the context of the MPPA-256 processor. A Low-Level programming environment is provided to realize the maximum performances or determinism of the safety-critical applications. It also supports OS, RTEs and middleware from 3rd party software developers. The Low-Level programming (LLP) environment virtualizes event and trap managing for simpler use by guest OS. Virtualization is applied by an exo-kernel type of hypervisor, where computing resources are managed and protected but are neither scheduled for use nor abstracted in an OS-specific way [38]. An overview of the LLP environment is given in Figure 9.

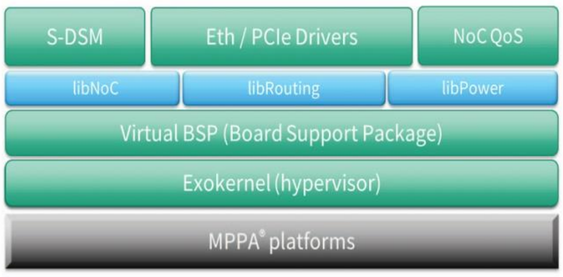

Figure 9 : The Kalray LLP environment

The LLP also delivers the nitty-gritty of a Software Distributed Shared Memory (S-DSM) system. This S-DSM provides shared memory abstractions for clusters. It enables the MMU of each core to achieve direct accessing of the external DDR memory, effectively transferring the content of the private memory of each CC into a last-level cache. The other manycore programming environments such as a POSIX-Level environment and an OpenCL environment utilize this.

Of late, the MPPA-256 processor is implemented in aviation-embedded systems. This requires that the entire system and application be certified according to the DO-178 avionic standard and the DO-245 airborne electronic hardware standard. Because of its clustered architecture, the MPPA-256 also effectively satisfies the imperatives of partitioning. The extensive roadmap towards certified spacecraft applications executing on this processor is based on embedding PikeOS on the I/O subsystems to support the Kalray-supplied Linux, and on realizing spacecraft certification for core elements of the Low-Level programming environment.

C. An Example Ongoing Research Project

FP7 ongoing project P-SOCRATES (Parallel Software framework for Time-critical Manycore Systems) [39] is combining the essential expertise from High-performance computing (HPC) and Embedded Computing (EC) platforms to cooperatively mitigate the complications of enabling timeliness assurances to applications with an increasing demand for computing performance. Thus, P-SOCRATES will allow the implementation of manycore architecture either in HPC or in EC systems. The main objective of P-SOCRATES is to execute OpenMP 4 applications on I/O clusters with offloading to the CCs, thus exporting a simple interface for real-time programming of the MPPA architectures.

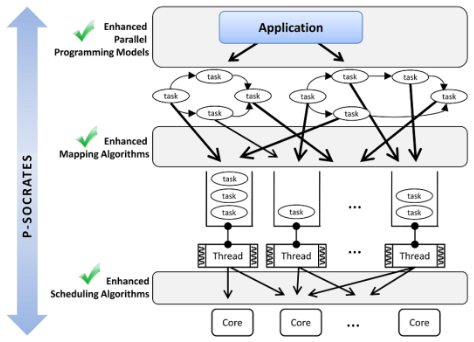
Enhanced Parallel Programming Models
Application
task
Enhanced Mapping Algorithms
Thread
...
Enhanced Scheduling Algorithms
Core
P-SOCRATES

Figure 10: P-SOCRATES integrating approach

To handle the predictability issues, this integrating approach provides a comprehensive software solution, able to fill the gap between application design and physical environment by implementing efficient parallel programming model. The concurrent software stack integrates bin-packing techniques with scheduling algorithms in order to achieve parallelization of tasks. This concurrent software framework is being extended for the use in safety-critical embedded platform. Figure 10 illustrates the software stack implemented in P-SOCRATES. It deduces a Task Dependency Graph (TDG) from the user application and allocates each task to the OS's threads statically; then these threads are dynamically scheduled on targeted manycore architecture [39].

Improved concurrent software solutions are being examined, integrating innovative principles and compiler technologies to build an extended TDG comprising the data dependencies between tasks and related information to address the influence of shared resources on real-time behavior of the system. Bin packing and task scheduling tools to choose an appropriate core mapping techniques then utilize this statistics. The bin-packing technique statically forms the efficient run-time configuration, effectively allocates tasks to OS's threads to provide timeliness guarantees without compromising the system performance. Then, the task scheduling tool interprets the task-to-thread allocation strategy into an effective thread-to-core mapping algorithm.

II. CONCLUSION

In modern embedded platforms, multicore processors have developed towards worldwide safety-critical applications. In this work, we explained some of the major issues which will decelerate our progress towards multicores from unicore in the context of the critical aerospace system. In summary, it is not enough for the aviation system to have a better understanding of concepts related to multicore processors; it is also essential for the system manufacturers and tool vendors to gain knowledge about critical functionalities and their requirements for multicore processors. Therefore, many significant efforts are presently under way or have been made in a series of research projects. We highlight the experience of both SYSGO and Kalray, who contributed many of these projects, in providing complete hardware design solutions and efficient OS level software solutions to make multicore core systems appropriate for safety-oriented applications. Several research efforts are still required to mitigate the problem of certification of SCS. So far, no complete value chain has been recognized in the aviation electronics for multicore processors. A large variety of research regarding this combined effort is still need to be addressed to adopt multicore architectures in safety-critical embedded systems.

III. REFERENCES

[1] Valavanis, K.P., “Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy”, Springer Publishing Company, Incorporated, 2007

[2] Visiongain, "The Unmanned Aerial Vehicles (UAV) Market 2011-2021: Technologies for ISR and Counter-Insurgency," Vision gain, 2011

[3] S. Vriters, "UAV Market Exceeds Five Billion Dollars In 2010," Space Dally, London, 2010,

[4] Media, Market Research, "U.S. Military Unmanned Aerial Vehicles (UA V) Market Forecast 2010-2015," Market Research Media Ltd, 2011.

[5] K. Wong, "Survey of Regional Develoments : Civil Applications," University of Sydney, Sydney, 2001

[6] Durrieu, G., Faug`ere, M., Girbal, S., Gracia P˜arez, D., Pagetti, C., and Puffitsch, W., “Predictable flight management system implementation on a multicore processor”, In Embedded Real Time Software and Systems, ERTS ’14, 2014.

[7] Richard West, Ye Li and Eric Missimer, “A Virtualized Separation Kernel for Mixed-Criticality Systems”, ACM Transactions on Computer Systems (TOCS), Vol.34(3), September 2016, Article No. 8

[8] parMERASA (2013), “Multi-core execution of parallelized hard real-time applications supporting analyzability”, Available [Online]:http://www.parmerasa.eu

[9] CERTAINTY (2013), “Certification of real time applications designed for mixed criticality,” Available [Online]: http://www. certainty-project.eu.

[10] Luis Miguel Pinho, Eduardo Quiñones, Marko Bertogna, Modena, Jorge Pereira Carlos, “P-SOCRATES: A Parallel Software Framework for Time-Critical Many-Core Systems” in 17th Euromicro Conference on Digital System Design (DSD), 2014

[11] ARTEMIS, Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments, Available [Online]:http://www.artemis-emc2.eu

[12] European Organization for Civil Aviation Equipment (1992), “DO-178B, Software Consideration in Airborne Systems and Equipment Certification. EUROCAE”, Available: [Online] https://en.wikipedia.org/wiki/DO-178B

[13] Guan, N., Ekberg, P., Stigge, M., & Yi, W. (2011), “Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems”, 32nd IEEE Real-Time Systems Symposium, pp.13 – 23. doi: 10.1109/RTSS.2011.10

[14] S. of Automotive Engineers (SAE). Arp 4754: (aerospace recommended practice) - certification considerations for highly integrated or complex aircraft systems, 2010

[15] RTCA. ”do-254/ed-80 - design assurance guidance for airborne electronic hardware”. Technical report, RTCA, Inc, 19th April 2000

[16] RTCA. do-178b/ed-12b - software considerations in air-borne systems and equipment certification”. Technical report, RTCA, Inc, 1st December 1992

[17] Baruah, S.K., “Bipasa Chattopadhyay, Haohan Li, and Insik Shin. Mixed-criticality scheduling on multiprocessors. Real-Time Systems, 50(1):142–177, 2014

[18] Boniol, F., “New challenges for future avionic architectures”, In Modelling Approaches and Algorithms for Advanced Computer Applications, volume 488 of Studies in Computational Intelligence, page 1. Springer, 2013

[19] John Rushby, “Partitioning for avionics architectures: Requirements, mechanisms, and assurance”, NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999

[20] Hermann Kopetz, “An integrated architecture for dependable embedded systems”, In Proceedings of the International Symposium on Reliable Distributed Systems, pages 160–161, 2004

[21] John Rushby, “Partitioning for avionics architectures: Requirements, mechanisms, and assurance”, NASA Contractor Report CR-1999-

209347, NASA Langley Research Center, June 1999.

[22] J. Nowotsch and M. Paulitsch, “Leveraging multi-core computing architectures in avionics”, In Dependable Computing Conference (EDCC), 2012 Ninth European, pages 132–143, May 2012

[23] R.Kaiser, “The pikeos concept history and design”, sysgo, white paper, 2007

[24] De Dinechin, B. D., van Amstel, D., Poulhi`es, M., and Lager, G., “Time critical computing on a single-chip massively parallel processor”, In Proceedings of the Conference on Design, Automation & Test in Europe, DATE ’14, pages 97:1–97:6, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association

[25] Ungerer et al., “Merasa: Multicore Execution of Hard Real-Time Applications Supporting Analyzability”, in IEEE Micro, vol. 30, no. 5, pp. 66-75, Sept.-Oct. 2010

[26] Nowotsch, Jan, and Michael Paulitsch, “Leveraging multi-core computing architectures in avionics,” Dependable Computing Conference (EDCC), 2012 Ninth European. IEEE, 2012

[27] [http://www.windriver.com/products/hypervisor/]

[28] International Electro technical Commission, IEC 61508, "Functional safety of electrical/electronic/programmable electronic safety-related systems", Switzerland, 2005

[29] Watkins, C. and Walter, R. “Transitioning from federated avionics architectures to integrated modular avionics”, In Digital Avionics Systems Conference, 2007. DASC’07, pp. 2.A.1–1–2.A.1–10, Oct 2007.

[30] RTCA and EUROCAE, DO-254 / ED-80, Design Assurance Guidance for Airborne Electronic Hardware, 2000

[31] QorIQ T2080 and T2081 communication processors, Available [Online]: http://cache.nxp. com/files/32bit/doc/fact sheet/T2080FS.pdf

[32] EASA. Certification memorandum - development assurance of airborne electronic hardware. In Software and Complex Electronic Hardware section, chapter 9. European Aviation Safety Agency, 11th Aug 2011.

[33] F. Certification Authorities Software Team (CAST). Position paper cast-32, multi-core processors, May 2014. Available [Online]: http://www.faa.gov/aircraft/air_cert#/design_approvals/air_software/cast/ cast_papers/media/ cast-32.pdf

[34] AS 6802. Time-Triggered Ethernet. SAE International, 2011

[35] AEEC, 1996. Avionics Application Software Standard Interface (ARINC-653). Airlines Electronic Eng. Committee.

[36] S. Trujillo, A. Crespo, A. Alonso, and J. P´erez., “Multipartes: Multicore partitioning and virtualization for easing the certification of mixed-criticality systems”, Microprocessors and Microsystems - Embedded Hardware Design, 38(8):921–932, 2014.

[37] K. Sandstrom, A. Vulgarakis, M. Lindgren, and T. Nolte. “Virtualization technologies in embedded real-time systems”, In Emerging Technologies Factory Automation (ETFA), 2013 IEEE 18th Conference on, pages 1–8, Sept 2013.

[38] D.R.Engler, M.F.Kaashoek, and J.O’Toole, Jr., “Exokernel: An operating system architecture for application-level resource management”, In Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, SOSP ’95, 1995

[39] The P-SOCRATES Consortium, P-SOCRATES (Parallel Software Framework for Time-Critical Many-core Systems) Available [Online]: http://p-socrates.eu.