Implications of Multi-Core Processors on Safety-Critical Operating System Architectures

Stefan Burger, Kevin Müller, Oliver Hanka, Michael Paulitsch
Airbus Group Innovations
Munich, Germany

Andrea Bastoni, Henrik Theiling
SYSGO AG
Klein-Winternheim, Germany

Matthias Heinisch
Airbus
Buxtehude, Germany

Abstract—**As additional hardware in airplanes increases their weight and in turn their fuel consumption, multi-core platforms are an interesting potential solution to achieve more processing capabilities onboard while avoiding carrying additional weight. However, compared to single-core platforms, multi-core platforms entail the additional price of requiring more complex components with tailored timing and communication strategies for the processes running on different cores at the same time. This paper presents the developed strategies and the lessons learnt from porting, deploying, and implementing on a multi-core platform a recent cabin management software of an actual passenger airplane and a security gateway for real-time application. As no standards and best-practices exist in the current industrial landscape, this work sets an important industrial basis for implementing and deploying safety-critical applications in multi-core environments.**

Keywords-component: Operating systems, industrial best-practice; ARINC 653; time partitioning; Avionics Software; Method Evaluation; Multi-Core Platforms

I. INTRODUCTION

There is a clear trend that future architectures of high-performance data processing and computing will be multi-core or even many-core processor platforms [1]. With increasing integration due to weight saving requirements and increasing needs of computing power due to functional integration, avionics – the electronics in aircrafts – needs to adapt to multi-core computing platforms as well. The need for functional integration of systems and the addition of new functionality (ideally without any additional weight) pushes processing performance to its limits. An example is represented by the various monitoring and support functions that have been integrated recently into many aircrafts in order to release the pilot from his routine tasks and improve airplane safety. Furthermore, customers i.e., airlines and ultimately passengers, constantly demand that more features –like high-definition in-flight entertainment (HD-IFE), Cabin Wi-Fi, etc. [21]– be available on airplanes.

However, the introduction of multi-core processing platforms into avionics is not without risks as it poses significant challenges at various levels. On single-core platforms, former issues of avionic architectures concerned system-level scheduling, communication links, and communication delays. In today's multi-core environments those issues become challenges at operating system (OS) level. We will outline this in more detail in the next section.

Nowadays, most of the costs for an airline are operational costs like fuel. Recent experience has shown that modern system design approaches can save nearly one ton of weight [2], which is a significant factor for saving fuel and hence for reducing operational costs. Multi-core CPUs have the potential to save even more Space, Weight and Power (SWaP), which are sparse in aviation platforms.

The main contributions of this paper are the investigation and evaluation of the influences from multi-core platforms into the real-time scheduling of real-world industrial applications. Such evaluations investigate multiple case studies that map real-world applications from the aerospace domain to multi-core processing platforms, and discuss the implications of possible OS-related (in detail ARINC-653-related) timing schemes and scheme changes that should be supported in the future. We present sufficient requirements and challenges for more flexible scheduling approaches in aviation systems. Given that no industrial standards and practical scheduling approaches exist so far for multi-core processors used in the safety-critical aerospace domain, this work is unprecedented and sets the basis for future development and discussions.

The analyzed case studies originate from porting an experimental cabin management and monitoring system (CMMS) and implementing a security gateway application on a multi-core platform. Such real-world applications are sufficiently challenging to evaluate different complex alternatives for timing scheduling solutions within a real-time OS. Our contributions will explain the lessons learnt and make suggestions for future multi-core avionic systems and time-partitioned OS architectures suitable for the avionics domain. Although possible scheduling solutions have been covered in literature for performance-oriented applications, the currently adopted approach in the safety-critical domain – with statically allocated execution slots as defined in the standard ARINC 653 [26] – requires a considerable configuration effort and is often not intuitive. Additionally, in order to minimize the certification costs (which correlate to complexity), real-time OS vendors adopt simple and lightweight solutions, thus further reducing many deployment scenarios that may be assumed straight forward in other domains.

The remainder of this paper has the following structure. Section II describes background information on our work. Section III gives an overview about the foundations of our evaluation and related work. Section IV explains the environment in which the evaluation has taken place. Section V illustrates our evaluation scenarios before we present the lessons learnt in Section VI. In the last section, we discuss planed future steps, and conclude our work.

II. Introduction of Multi-Core Computers to Avionics and Related Challenges

The introduction of multi-core systems in avionics causes challenges where, due to safety regulations, independent computing resources are still an important requirement. Shared resources such as memory, busses, system level caches, and chip input/output blocks – often praised as cost-saving factors in consumer electronics – can pose significant challenges in safety-driven computing cultures. A real concern is whether full *time*- and *space-partitioning* – i.e., the segregation of resources – can be guaranteed even in worst-case scenarios [3][30]. Additionally, with the reduced feature sizes of today's multi-core production processes, single event effects become more prevalent [15]. Only in recent years, the aviation certification authorities such as the Federal Aviation Administration (FAA) or the European Aviation Safety Agency (EASA) have started clarifying their positions with respect to complex electronics such as multi-core processors [4]. Nevertheless, the need for more information and experience with commercial off-the-shelf (COTS) multi-core platforms and their influence on safety-relevant functions is required in order to design hard real-time applications and to certify those systems [8]. Several projects have started gathering the required knowledge to allow future certification of multi-core platforms (e.g., RECOMP [12], EMC2 [13], ARAMiS [14][13]).

A second challenge is the migration of the currently used aviation software architectures from single-core to multi-core processors. Together with the transition from the federated architecture (*fedArch*) to Integrated Modular Avionics (*IMA*) [7], the transition to multi-core platforms can be clearly considered another challenging industrial paradigm change. In a fedArch, each system function uses its own resources i.e., each function utilizes a dedicated board (CPU, RAM, I/O etc.). IMA allows a better usage of computing resources: instead of deploying one board per function, in an IMA architecture, functions utilize a common, shared computing farm – which includes I/O – and several logical functions are encapsulated in so-called *partitions*. The most widely used OS standard for IMA development is ARINC 653 [26], defining the Avionics Application Standard Software Interface. This standard describes a layered OS environment [5] where separated partitions host different functionality with different criticality levels. These partitions are virtual containers hosting separated software. Each partition works on an individual subset of system resources such as CPU cores, I/O, and RAM. A strict, a-priori known, *static time scheduling* controls the execution-time of the partitions and allows to set hard real-time execution constraints for every application. The first airplanes using an IMA approach were the Airbus A380 and Boeing's B777. Nowadays, all modern airplanes –like the A350 or the B787– have adopted IMA system architectures in combination with ARINC 653-compliant OSs.

The next generations of airplanes will have to adapt multi-core platforms in combination with IMA and ARINC 653. However, this will force developers and system integrators to change current software architectures, in particular their communication models and scheduling processing schemes. Another challenge when using partitioned systems in combination with multi-core platforms is how to correctly devise the time scheduling approach for several partitions, the scheduling of all partitions together within one hyperperiod (*major time frame*), and the partition time frame of parallel executing partitions [10]. On a multi-core platform, partitions will be executed in parallel and therefore, in order to allow an optimized usage of resources and communication, the system designer needs to find a suitable static time-scheduling schema that fulfils the different timing requirements. Furthermore, on a multi-core platform, communication between partitions is more complex, since the software designer needs to respect the different timings in combination with applications executed in parallel. For example, he has to take into account how long a message transfer requires to reach its destination partition (possibly executing on a different core) to guarantee worst-case timing requirements.

In order to fulfill all needed safety and security requirements and therefore fulfilling the certification requirements, while at the same time containing certification costs, OSs need to reduce the configuration complexity of systems deployed on multi-core platforms. Limiting the possible timing schema options is a viable and very promising approach to enable relative straightforward configurations and to satisfy certification requirement. In section V, we present and discuss the timing schema options that would fit both the above configuration and certification requirements in the context of a recent IMA cabin management software.

III. Foundations and Related Work

This section explains all foundations and fundamentals of the evaluation system, state-of-the-art, and used techniques.

A. Native Separation-support in OSs

Given the importance of ensuring time- and space-segregated execution of partitions, it is fundamental to select an appropriate execution environment for the partitions. A microkernel-based OS meets this requirement by offering a *separation kernel* with support for scheduling of active entities, separation of memory and access control, separation of external devices, interrupt handling, and inter-thread communications. A microkernel-based OS is therefore the natural platform for the evaluation presented in this paper. Specifically, to practically discuss our approach we have chosen SYSGO's PikeOS (Version 3.2), which allows asymmetric multiprocessing (AMP) and symmetric multiprocessing (SMP) [9] in combination with ARINC 653 partitioning [6]. SYSGO's PikeOS is a commercially available microkernel based real-time OS that provides hypervisor-like virtualization capabilities and ensures strict time and resource partitioning.

Separation is achieved in PikeOS through the concepts of resource and time partitions. Resource partitions encapsulate one or more tasks,[1] which in turn encapsulate one or more threads that constitute the active scheduling entity in PikeOS [25]. A resource partition in PikeOS is a container of a set of physical resources and privileges for user applications – implemented using one or more tasks or threads. In other words a resource partition is a virtual-machine environment for

[1]A task identifies a separate address space shared by all threads assigned to it.

guest applications spacing from simple tasks to complete guest OSs. PikeOS' security and safety functionality lie in the ability of strictly separating the physical resources assigned to a resource partition.

Furthermore, each thread in PikeOS is assigned to a ***time partition***. Each resource partition can be assigned to one time partition while each time partition can be associated with several resource partitions. The relations are illustrated in Figure 1. Processing time is allocated to time partitions and threads inside one time partition share the allocated processing time, which is defined by a *static* time-window-based scheduling. [5]

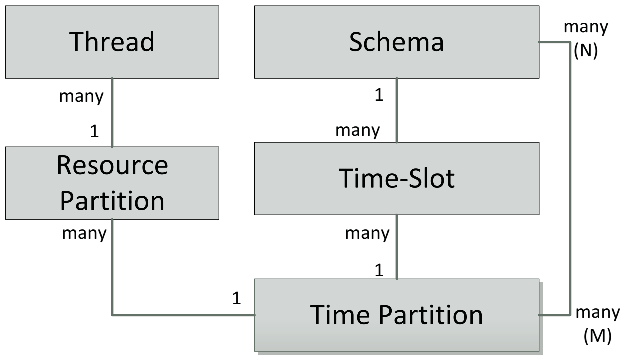

Figure 1: Relationship among resource and time partitions in PikeOS. A Time Partition can comprise multiple time slots, which form a Schema. A Schema contains therefore several Time Partitions. Threads are assigned to Resource Partitions, which in turn are assigned to Time Partitions.

An appropriate assignment of resource partitions to time partitions allows to implement of the execution model implied by the ARINC 653's major time frame concept.[2]

B. *Challenges of Scheduling a Multi-Core Systems*

Today's IMA platforms typically use single-core processors like the IBM 75x or Freescale MPC744X family. These processors have a relatively simple caching and pipeline architecture. Often, they are pure host processors requiring separate bridge chips, which are specifically developed for an individual airplane system. In contrast, most multi-core processors are COTS components with complex instruction pipelines, branch prediction, multi-level caches and cache coherency modules with included I/O and DMA controllers. These optimizations lead to better performance but lower the timing determinism.

On a single-core processor, no thread parallelism is allowed, while on a multi-core processor, parallel execution of multiple threads is the normal case. This leads to interference between applications running on different cores when hardware resources are *shared* between the cores. Resource sharing is detrimental for worst case performance: as shown in Schliecker [19] and in Fuchsen [17], the performance of one core can drop by 50% for specific applications if memory or PCI bus is excessively used by other cores. Similar results are shown in Nowotsch [3] for memory and platform-level caches.

Common multi-core processors have multi-level caches for performance optimization and they use cache coherency modules and crossbars to interconnect the cores. Software running on different cores is therefore not executing independently. Even in the absence of explicit software data or control flows between cores, resource-coupling exists at platform level due to shared hardware. A switch between resource partitions comprises a context switch on the specific cores. Caches are likely to have to be flushed and synchronized for coherency.

C. *Related Work*

Carpenter et al. [22] described challenges and complexity in theory and practice of scheduling in a safety-critical environment, which is likely to run on a multi-core processor. King [23] recently explained techniques, such as slack partitioning and cached scheduling, for the usage in safety-critical software on multi-core platforms. He argues that this is a powerful approach as applications can utilize remaining computing bandwidth in a systematic manner. Carnevali et al. [18] have presented a formal approach to design and verify two-level hierarchical scheduling systems, as used in ARINC 653. The approach includes all necessary steps from design to development of real-time systems. Schliecker et al. [19] introduced an analytical approach for calculating worst-case response times in automotive real-time system using tasks and shared resources. Nowotsch et al. 0 introduced an approach to manage multi-core Worst-Case Execution Time (WCET).

IV. Case Study Environment

In this section, we explain the two systems we used to evaluate time partitioning on multi-core platforms. Furthermore, this section introduces the used hardware and software platforms.

A. *Cabin Management System*

The cabin management and monitoring system (CMMS) of commercial passenger airplanes is the major controlling device for functionalities like cabin light, crew communication, passenger announcement, climate, water and waste etc. For the enhancement of in-flight accommodation, airlines and passengers recently demanded for more up-to-date technologies such as Cabin Wi-Fi, HD-Inflight-Entertainment Systems, or Cabin Video Monitoring. All such demanded technologies need more processing power. DO-214 [20] describes some of the minimum audio operational performance requirements of a CMMS, which rise challenging time and maximum communication delay requirements. Some of the CMMS's functions are safety-critical, i.e., the communication between the cabin server and the end devices has to be guaranteed. In order to guarantee communication in hard real-time, currently one application periodically (<100µs) sends data to the end devices. Designing a timing scheduling for a single-core platform is quite easily compared to a multi-core platform. A CMMS system has several applications with different WCETs. This generates several hundred safety requirements that must be satisfied by the system. Although on

[2] A major time frame is a repeating, fixed-length period during which each partition is executed at least once; the major time frame is therefore a multiple of the hyperperiod of all partition periods.

a single-core the time-scheduling is linear and clearly defined, it is currently a challenge to devise an optimized time-scheduling schema for new system configurations holding around 30 cabin-related applications.

The mentioned trend of deploying more cabin functionality onto one platform requires more computing performance. Current systems use single-core CPUs. Increasing the number of single-core computers to reach better performance would require more space, power consumption and weight, with undesirable impacts on the airplane. Multi-core platforms would allow minimizing size and weight in both current and future systems. Leveraging a multi-core-capable ARINC 653 OS [27], a new cabin server architecture that holds several functions integrated on one board is feasible and desired.

B. *Hardware Platforms*

The new experimental CMMS we implemented as a case study uses several Freescale MPC8641D CPUs with two e600 cores as evaluation platform. All CPU cards are using Ethernet for inter-CPU communication. This experimental platform implements only some important cabin functionalities, like the cabin light and crew communication, with the intention to evaluate a first version of a new ARINC 653 compliant multi-core OS.

C. *Software Platform*

As discussed in section III, the OS chosen for our software platform is SYSGO's PikeOS 3.2. PikeOS offers AMP and SMP support, complies with the ARINC 653 standard and is certifiable according to DO-178b [24] at the required design assurance level (DAL) (level B or C depending on CMMS system architecture and application). Furthermore, it has been already used in several safety-critical avionics devices.

V. EVALUATION SCENARIOS

In order to evaluate a multi-core test environment platform, we have designed four scenarios based on already used applications in the current CMMS. All scenarios are designed for a dual-core CPU platform, which was chosen as fundamental configuration for the next generation of experimental CMMS. Furthermore, all scenarios use four partitions. These partitions can have different WCET and hold the implementation for a CMMS use case. It is important to mention that the current application timings have been reused without considering constraints given by the current OS. This approach allowed to investigate the limits of current software and hardware, and to make suggestions for the next generation of CMMS systems. The analysis explicitly focuses on the communication overheads entailed by the different solutions, and on the trade-offs that are required due to the constraints imposed by each solution. Please note that without restricting the discussion and the presented solutions, for simplicity, multirate systems (as defined in ARINC 653) are not discussed in this paper.

A. *Fixed Partition Time Frame Durations*

The basic idea of the first evaluation scenario is to fix execution times (partition time frames [10]) for all partitions on both cores running at the same time. The time partitions are defined as T1 and T2, whereby the amount of execution time of T1 and T2 is variable. Our approach is using four (resource) partitions (P1, …, P4) on a dual core platform, so that every core has assigned two partitions. These numbers of applications were chosen by selecting a number of relevant CMMS applications implementing basic cabin use cases. At a given time, two partitions are executed in parallel and two partitions are idle. The parallel-running partitions are assigned to the same time partition, i.e., P1 and P3 are assigned to T1. After the end of one time partition (e.g., T1), the cores switch to the next partitions. This scenario requires that one pair of partitions ends at the same time and that the partition change must occur on both cores at the same time. illustrates the mentioned scenario. In this figure, time is progressing (cyclically) from left to right.

In this scenario, a communication within a pair (e.g., between P1 and P3 or P2 and P4) of partitions can be done without large delays. A communication between P1 and P4 needs at most the complete duration of T1 to reach P4, and an answer from P4 to P1 can require up to the duration of T2. Thus, the overall communication would need, in the worst case, T1 + T2. In this example, the major frame has a period of 250µs (T1 + T2).

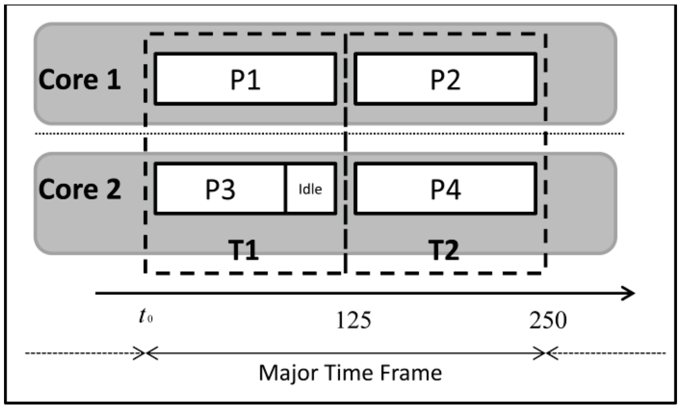

Figure 2: Schema Fix Partition Time Frames

Due to the need of switching between T1 and T2 at the same time on both cores, partition's durations has to be extended to the duration of the longest partition: if P1 has a WCET of 125 µs while P3 requires only 100µs, then P3 has to be "extended" to 125µs as well. Therefore, 25 µs of P3's execution on Core 2 will be not utilized (idle).

B. *Individual Partition Time Frame Durations*

This scenario uses one individual time frame for every application, which is allocated to a resource partition. All these partitions (P4, …, P7) have a different WCET. In order to allow different partition time frames on every core, every core has an independent scheduling time schema and resource partitions are assigned to independent time partitions (T3, …, T6). The different partition time frames on every core generate different major time frames: P4 (assigned to T3) on core 1 has a longer execution time than P6 (assigned to T5) on core 2. From the perspective of the system designer both cores can be seen as two independent CPUs. illustrates this scenario.

As an example, if the WCETs for T3, T4, T5, and T6 are 80µs, 50µs, 60µs, and 60µs respectively, then the major frame on core 1 is 130µs while it is 120µs on core 2. This approach influences the communication between partitions. The major time frame of core 1 is 10µs longer than the major frame on core 2. After 12 time frame periods, core 2 has executed one complete cycle more than core 1. A guaranteed direct communication between two partitions is very complex in this scenario. In a real deployment, there is the need of finding an effective tradeoff between simplifying communication and a reasonable, contained amount of idle time. A possibility would be to maintain a global major period on both cores, with different partitions executing on each core with different periodicity. In the example above, this would lead to a global major time frame of 130µs with a 10µs idle time on core 2.

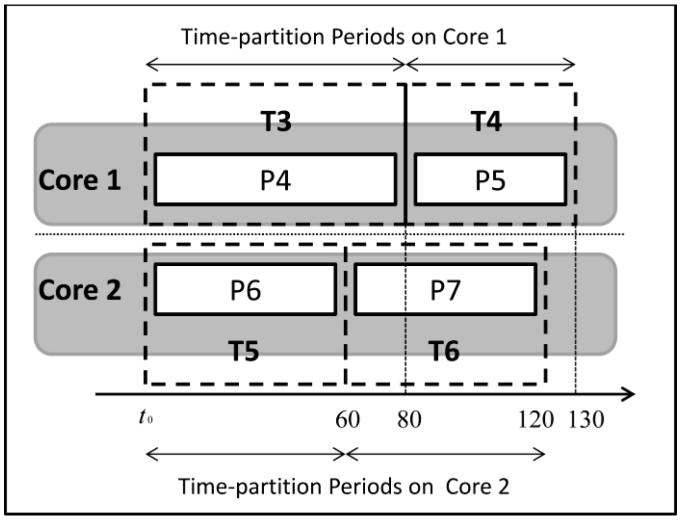
Time-partition Periods on Core 1
T3
T4
Core 1
P4
P5
Core 2
P6
P7
T5
T6
t_0
60
80
120
130
Time-partition Periods on Core 2

Figure 3: Individual Partition Time Frames

C. One Application, Two or More Cores

The third evaluation scenario focuses on the timing of an application using more than one core. This scenario is based on the idea that an application needs more performance than those achievable on one core only. Possible examples for this scenario are image processing applications for video monitoring, communication applications for cabin phone conferences, or in general, parallelizable applications with very short execution times (e.g., around 20µs) and the requirement of a high amount of processing power. In this case, one time partition (T7) spans on all cores, and the single parallelizable application is mapped on a single resource partition (P8), allocated to both cores. The scenario is illustrated in Figure 4 (note that in the following figures, time is progressing from top to bottom).

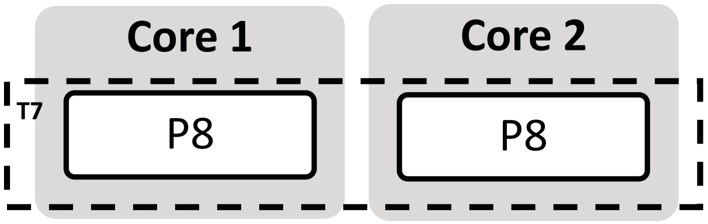

Figure 4: Schema for one Application on two Cores

From communication point of view, this scenario is straightforward: since the partition is executed on both cores, a shared memory solution is an efficient solution for the communication model. The major drawback of this solution is that it is only applicable for those applications that can be easily parallelized, while most of today's legacy applications are optimized for single-core execution. Furthermore, if dynamic elements are used in the deployment of tasks on different core, safety certification may become harder.

D. Security Gateway Use Case

In order to integrate more than one security domain (potentially at different criticality) in a system, and allowing the domains to securely communicate, a gateway (an additional software application) can be used to control the data flow between resource partitions in accordance with the system security policy. To realize an appropriate scheduling of such a gateway, two contradicting paradigms should be followed. While on the one hand, the gateway should execute as little as possible to avoid performance loss and allow to meet the demanding real-time requirements of the controlled applications, on the other hand, the whole system must remain deterministic and secure. Thus, the gateway needs to process the maximum amount of communication data within one scheduling period.

On single-core architectures there are two possible scheduling solutions. In the first solution, the gateway (P9) can run in its own time partition that is scheduled directly between the other communicating partitions (P10 and P11) –see . This generates a complex time-scheduling schema and may lead to some idling time overhead since the applications may not send the same amount of data in each period. The second approach is to "steal" processing time directly from the time-partition that initiated the communication. This approach adds more complexity to the WCET analysis of an application that is required by the certification process of high-criticality applications.

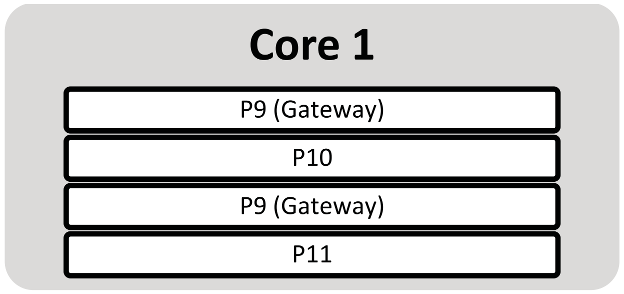

Figure 5: Single-core scheduling with Gateway processing time slot between application partitions

By using multi-core architectures it is possible to run the gateway (P9) on one dedicated core concurrently with the applications on other cores (P10 and P11) – see Figure 6. Although this approach may under-utilize the core where the gateway is allocated (as P10 and P11 do not continuously exchange messages), a higher utilization of the gateway's core can be restored by allocating lower priority tasks (or non-real-time tasks) on the same core. Thus, processing time will be employed to process the low-priority background tasks that are promptly interrupted by the gateway when a communication takes place. Although appropriate to avoid under-utilized cores, this approach could expose timing covert channel inside the system, and therefore, it requires the background low-priority tasks to be trustworthy.

A possible method to realize the described scheduling behavior within PikeOS is to use the time partition T0. In contrast to standard time partitions, threads assigned to T0 are always eligible to be scheduled concurrently with the threads belonging to the current active standard time partition. The eligible thread with the highest priority between the active standard time partition and T0 is scheduled on the core [5]. Hence, T0 can be used for event-driven real-time applications or applications without real-time requirements that can run when higher priority threads have completed. This approach is somewhat similar to the background server and slack scheduling approaches (e.g., [28], [29]) for multi-core platforms.

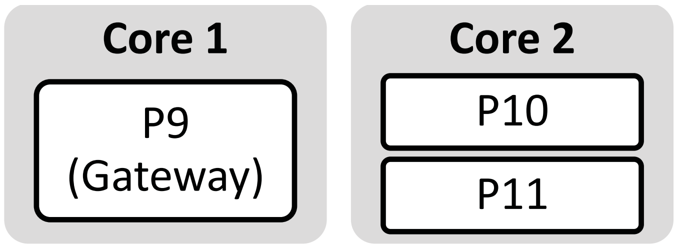

Figure 6: Multi-core scheduling with Dedicated Core for the Gateway

VI. LESSONS LEARNT

In this section, we summarize the evaluation of the scenarios mentioned in the last section. In order to evaluate the scenarios, we implemented prototypes of the CMMS applications and of the gateway application. The prototypes are sufficient to demonstrate and evaluate the usage of multi-core platforms and SYSGO's PikeOS 3.2.

A. Fixed Time Frame Durations

The implementation of this scenario shows that it is very challenging to find applications with the same or at least compatible WCET. All implemented applications had very different execution times and this generated high variance in the time-partitions' durations, leading to high idle times and under-utilized cores. As a "worst-case" example, during the implementation we faced the challenge of allocating on different core monitoring applications with a WCET of $250\mu s$ to be run almost concurrently with an audio streaming application with a WCET of $10ms$.

An additional issue exposed by this solution is high latency entailed by concurrently flushing caches on time-partition switches on multiple cores. This is required to ensure the correct initial conditions and deterministic execution of highly safety-critical applications. Unfortunately, writing back caches into memory on all the partitions at the same point in time leads to longer write-back times due to concurrent use of the common memory and buses resources.

Another difficulty was to deal with a periodic partition that must execute after a specific amount of time. In this scenario, if a periodic application is combined with a non-periodic application, the non-periodic application must also be scheduled when the periodic partition is activated. Although this may seem a trivial task, most periodic applications have either high or low priorities. Therefore, in case of low priority periodic application, the system may waste computing time for the unnecessary non-periodic application. The opposite situation occurs in the case of high priority periodic application, as the periodic application may be executed due to the activation of the non-periodic one. In some cases, it is impossible to combine non-periodic and periodic application on the some core, because their scheduling requirements are completely incompatible.

B. Individual Time Frame Durations

The idea of the second time schema scenarios or variants thereof is currently not supported by any ARINC-653-compatible OS. Nonetheless, this scenario is fully flexible and would fulfill most of the needed requirements of application designers. Every core has its own scheduling schema, and time-partition assignment is independent on every core. One of the main challenges is related to the communication between applications assigned to different time-partitions on different cores. The synchronization of such communication is very challenging as each core may execute independent sequences of time-partitions. The application designer is therefore forced to closely monitor the timing of data transfer between different applications. Such task is complex and hard to configure and control as the major execution cycle of one core can be faster than the cycle(s) of the other core(s). Therefore, in this scenario, important and complex communications behaviors are more difficult to implement. Another major challenge is the current lack of precise WCET tools supporting applications running within multiple ARINC 653 partitions [16]. This makes the task of precise evaluation of the length of each independently executing time partition even harder. Improvement in this field would allow future systems to calculate the best scheduling schema by using information from applications and partitions.

C. One Application, Two or More Cores

As noted, the major difficulties of implementing this scenario are related to the parallelization of legacy, already certified, single-core applications. Furthermore, parallel-executing threads are not independent from each other, and may cause interference through, e.g., competing OS's system calls. Consequently, the timing of applications executing on other cores is affected. Although interference on other partitions and cores can be minimized by for example, synchronizing partition scheduling on all cores, high WCET penalties are unavoidable. It should be noted that newer versions of PikeOS (e.g., version 3.4) enable a synchronized partition switching on multiple cores, but such a feature was not available when the implementation was carried out.

D. Security Gateway Use Case

In case of our gateway application, we clearly see the potential benefit of using multi-core architectures for those upcoming safe and secure system designs that adds data flow controlling components. These gateway components need to execute whenever a communication between two applications belonging to differently classified security domains occurs. In order to avoid changes of currently established single-core scheduling schemas, the gateway can be processed on a dedicated core. However, since we cannot determine for all applications when communications exactly occur, the gateway is required to be executed whenever a communication request is initiated. To avoid the high utilization penalty of exclusively using a core for the gateway only, an event-driven real-time application should share the core with uncritical software (if possible and available). PikeOS' time partition T0 is a possible solution to operatively realize such a scenario and to deal with otherwise unused idle time.

T0 has been used as an always-active background time partition. Since it can contain multiple resource partitions with different priority levels, low-priority resource partitions (and therefore applications) have been assigned to T0 so that they could be scheduled during the idle time of the other time-partitions. The deterministic scheduling of other time partitions is not affected and only spare, otherwise-idle time is spent for low-priority resource partitions within T0. It should be noted that high-priority event-driven resource partitions can be assigned to T0 as well to minimize the response time after an event occurrence. In fact, such high priority partitions can preempt any lower priority partitions (in both T0 or in any standard time-partitions) at any time thus ensuring fastest possible response times. In case of this event/driven scheduling the system architect has to evaluate the potential impact on time determinism.

It should be noted that several challenges arose from the use of the T0 approach. In fact, although T0 can guarantee the execution time of a task, tasks and resource partitions assigned to T0 are preferred to equal-priority resource partitions assigned to standard time-partitions. This may lead to potential starvation of the resource partitions assigned to standard time-partitions.

VII. Future Steps

The scenarios presented in the paper are the first step in a long-term research project that aims to develop a new CMMS version for future airplane generations. One of the next steps will be the definition of new software architecture optimized for ARINC 653 and multi-core platforms. A first proposal of such architecture is presented in [11].

Furthermore, we identified some CMMS applications that could be successfully parallelized to fulfill the scenario in section V.C. Therefore, we have started to evaluate a four-core-CPUs for the experimental CMMS with limited functionality. To implement all original and new functions even more resources are needed.

Our first evaluation has successfully shown that a multi-core platform is able to hold more than the applications currently required for a CMMS. In the next steps, we would like to combine several other systems like video information and passenger Wi-Fi network in one single multi-core platform. However, this approach entails new security concerns such as those implied by combining several different security levels in one platform. In order to provide the highest security level, we will have to define new methods and techniques on the level of applications, OSs and hardware.

Strict time and space partitioning is one solution for dealing with the arising security challenges. However, these approaches can only prevent the leak of information of faulty memory accesses or timing attacks. However, since in every integrated system communication between partitions is required, the needed communication channels would be vulnerable to transfer secure data out of a security domain. An integrated gateway solution for controlling the information flow between partitions belonging to different domains similar to the one presented in scenario in section V.D is a first step towards securing such kind of communications. Finding multi-core optimized software architecture and scheduling approach for such gateways will be a challenging part of our future work.

Multiple accesses to I/O devices are additional complex activities that will require further investigation. During several tests, we discovered that the currently-in-use driver design has to be improved to support multi-core platforms and partitions. A clean driver interface is needed, which can control parallel accesses from partitions.

It should be noted that not all the presented multi-core design possibilities can be operatively realized with the currently available hardware/software combinations. On the one hand, due to the complexity of current multi-core platforms with shared hardware components it is very difficult to estimate accurate-WCET bounds that can be used to characterize the execution requirements needed by some of the identified design solutions. On the other hand, limitations of operating-systems such as the employed version of PikeOS (version 3.2, still optimized for single-core solutions) still do not offer adequate certifiable functionalities and means to efficiently implement all the proposed design solutions. However, newer and upcoming versions of OSs like PikeOS (with new interfaces and improved tools) provides (e.g., in PikeOS version 3.4) and will provide improved support for some of the major constraints identified in the presented scenarios. In particular, the ability of running individual time partitioning schemes on separate cores with the upcoming PikeOS version will allow to significantly simplify the porting and the scheduling of avionics applications to multi-cores.

VIII. Conculsion

In this paper we have presented the investigation of a multi-core platforms usage in the context of a real-world safety critical real-time environment with a strong focus on time scheduling and communication behavior. The main goal of this study was to better understand the operational difficulties and opportunities involved in finding an optimal scheduling for industrially-relevant real-time applications and to illustrate potential approaches when such applications are implemented and deployed on a multi-core platform. We have investigated four scenarios that were derived from a real CMMS taken from a state of the art Airbus plane. These scenarios can be seen as challenging archetypes for most aviation applications in a safety-critical real-time environment.

For each of the investigated scenarios, execution time and periods of the selected applications as well as communication needs between them have been discussed. Our results underline the challenges in migrating current avionics application designs from single-core to multi-core platforms while preserving appropriate time and space separation properties. For each analyzed solution, the major drawbacks have been presented.

Furthermore, we have proposed ideas on how to optimize software designs and communication models for future multi-core systems, not only in the avionics field, but in other fields with similar safety certification and performance requirements. When dealing with upcoming multi-core safety critical systems, an optimized solution for task distribution in a certifiable real-time environment will require a combination of system architecture, software architecture, operating system, and communication modules and further joint development efforts.

ACKNOWLEDGMENT

This work was funded within the projects ARAMiS by the German Federal Ministry for Education and Research (BMBF) with the funding ID 01IS11035 and SeSaM funded by the BMBF under funding ID 01BY1122 and by ARTEMIS research grant RECOMP under the research grant of the BMBF with funding ID 01IS10001.

We would like to thank our colleagues from Airbus Group Innovations in Munich, Airbus in Buxtehude, and SYSGO AG in Klein-Winternheim for their support and numerous discussions.

REFERENCES

[1] G. Lowney, Why Intel is designing multi-core processors, ACM symposium on Parallelism in algorithms and architectures, pp. 113, 2006.

[2] J.W. Ramsey. Integrated Modular Avionics: Less is More. Aviation Today, Feb. 1, 2007. Accessed on Jan. 11, 2012 at http://www.aviationtoday.com/av/commercial/Integrated-Modular-Avionics-Less-is-More_8420.html.

[3] J. Nowotsch and M. Paulitsch.Leveraging Multi-Core Computing Architectures in Avionics.9thEuropean Dependable Computing Conference (EDCC 2012). Accepted for publication.

[4] EASA, “Certification memorandum - development assurance of airborne electronic hardware,” Software & Complex Electronic Hardware section, European Aviation Safety Agency, CM EASA CM - SWCEH - 001 Issue 01, 11th Aug. 2011.

[5] R.Kaiser, S. Wagner,Evolution of the PikeOS microkernel, First International Workshop on Microkernels for Embedded Systems,2007.

[6] SYSGO AG, PikeOS 3.2 Datasheet, Accessed on Feb 4, 2012 at http://www.sysgo.com/nc/products/pikeos-rtos-and-virtualization-concept/whats-new-with-pikeos-32/?cid=921&did=679&sechash=30a9f019

[7] C. Watkins, R. Walter, Transitioning from federated avionics architectures to Integrated Modular Avionics, IEEE/AIAA 26th Digital Avionics Systems Conference, 2007.

[8] P. Radojkovic, S. Girbal, A. Grasset, E. Quinones, S. Yehia, F. J. Cazorla. *On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments*. ACM Trans. on Architecture and Code Optimization, Jan. 2012.

[9] P.N. Leroux,R. Craig, Easing the Transition to Multi-Core Processors, Information Quarterly, Vol4, pp34-37, 2006.

[10] P. Parkinson, L. Kinnan. *Safety-critical software development for integrated modular avionics*, Embedded System Engineering, Electronic Design Automation Ltd., Vol. 11(7), pp. 40-41, 2003.

[11] S. Burger, E. Heidinger, S. Schneele, O.Hummel, M. Heinisch, W. Fischer: "Towards Higher Changeability for Airplane Cabin Software", in Proceedings of the IASTED International Conference on Software Engineering and Applications, Dallas, 2011.

[12] RECOMP Project. Reduced Certification Costs Using Trusted Multi-core Platforms. ARTEMIS Project. Project web page http://atc.ugr.es/recomp/.

[13] EMC2 Project. Embedded Multi-Core systems for Mixed-Criticality applications in dynamic and changeable real-time environments. ARTEMIS project. Project web page http://www.emc2-project.eu/.

[14] ARAMiS Project. Automotive, Railway and Avionic Multicore System. http://www.offis.de/offis_im_profil/struktur/projekte/ansicht/detail/status/aramis.html. German National Project (BMBF).

[15] C. Constantinescu. *Trends and Challenges in VLSI Circuit Reliability*. IEEE Micro Journal, Vol. 23(4), July 2003.

[16] P.J. Prisaznuk, ARINC 653 Role in Integrated Modular Avionics (IMA).. IEEE. pp. 1—E.

[17] R. Fuchsen, “How to address certification for multi-core based ima platforms: current status and potential solutions”, 29th Digital Avionics Systems Conference, October 3-7, 2010.

[18] L. Carnevali,G. Lipari, A. Pinzuti, , E. Vicario,, “A formal approach to design and verification of two-level hierarchical scheduling systems”, Reliable Software Technologies-Ada-Europe, 2011, Springer, vol.6652, pp.118 -131.

[19] S. Schliecker, M. Negrean, R. Ernst, "Response Time Analysis on Multicore ECUs With Shared Resources," *Industrial Informatics, IEEE Transactions on* , vol.5, no.4, pp.402-413, Nov. 2009

[20] RTCA Inc. *DO-214, Audio Systems Characteristics and Minimum Operational Performance Standards for Aircraft Audio Systems and Equipment.* Issued 3-2-1992.

[21] M. Babb, Wi-Fi in the sky, Computing & Control Engineering Journal, IET, 2004, Vol. 15, No. 2, p. 2

[22] T. Carpenter, K. Driscoll, K. Hoyme, J. Carciofini, Arinc 659 scheduling: Problem definition, Real-Time Systems Symposium, 1994., Proceedings., 1994.

[23] T. King, Slack scheduling enhances multicore performance in safety-critical applications, Electornics Design, Strategy and News (EDN), 2011.

[24] RTCA, 'DO-178B: Software Considerations in Airborne Systems and Equipment Certification', DO-178B/ED-12B, 1992.

[25] SYSGO - Embedding Innovations, PikeOS Fundamentals,Manual, 2011.

[26] Avionics Application Software Standard Interface, ARINC Specification 653, January 1997.

[27] Huyck, P., ARINC 653 and multi-core microprocessors — Considerations and potential impacts, Digital Avionics Systems Conference (DASC), 2012 IEEE/AIAA 31st , pp.6B4-1,6B4-7.

[28] S. Baruah, J. Goossens, and G. Lipari,. Implementing constant-bandwidth servers upon multiprocessor platforms. In Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium, 2002, pp. 154–163.

[29] Faggioli, D., Lipari, G., and Cucinotta, T. The multiprocessor bandwidth inheritance protocol. In Proceedings of the 22nd Euromicro Conference on Real-Time Systems, 2010, pp. 90–99.

[30] O. Kotoba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling. Multi-core in real-time systems - temporal isolation challenges due to shared resources. Proc. of Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems (at DATE Conf.), 2013.

J. Nowotsch, M. Paulitsch, D. Buehler, H. Theiling, S. Wegener and M. Schmidt. *Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement.*Proc. of 26th Euromicro Conference on Real-Time Systems (ECRTS) 2014.