Parallel Many-Core Avionics Systems

Miloš Panić[*,†], Eduardo Quiñones[†], Pavel G. Zaykov[ψ],
Carles Hernandez[†], Jaume Abella[†], Francisco J. Cazorla[†,‡]

[*]Universitat Politècnica de Catalunya (Spain)
[†]Barcelona Supercomputing Center (Spain)
[ψ]Honeywell International (Czech Republic)
[‡]Spanish National Research Council (IIIA-CSIC) (Spain)

{mpanic, equinone,chernand,jabella,fcazorla}@bsc.es, {pavel.zaykov}@honeywell.com

ABSTRACT

Integrated Modular Avionics (IMA) enables incremental qualification by encapsulating avionics applications into *software partitions* (SWPs), as defined by the ARINC 653 standard. SWPs, when running on top of single-core processors, provide *robust time partitioning* as a means to isolate SWPs timing behavior from each other. However, when moving towards parallel execution in many-core processors, the simultaneous accesses to shared hardware and software resources influence the timing behavior of SWPs, defying the purpose of time partitioning to provide isolation among applications. In this paper, we extend the concept of SWP by introducing *parallel software partitions* (pSWP) specification that describes the behavior of SWPs required when running in a many-core to enable incremental qualification. pSWP are supported by a new hardware feature called *guaranteed resource partition* (GRP) that defines an execution environment in which SWPs run and that controls interferences in the accesses to shared hardware resources among SWPs such that time composability can be guaranteed.

1. INTRODUCTION

Safety critical real-time systems in avionics and automotive domains use increasingly more sophisticated (complex) functionality, which requires higher and higher levels of computing power. Many-core[1] processors are considered to cope with the performance and cost constraints imposed by future safety critical real-time systems. On one hand, many-cores allow scheduling mixed-criticality applications into the same processor, maximizing the hardware utilization while meeting size, weight and power constraints. On the other hand, many-cores improve application performance by exploiting *task level parallelism*.

[1]We use the term many-core to refer to a processor integrating high number of cores (e.g. more than 16). The problems shown in this paper for the adoption of many-core in future avionics systems also arise, to a lesser extent, in multi-core processors.

ESWEEK'14, October 12-17, 2014, New Delhi, India

http://dx.doi.org/10.1145/2656045.2656063.

Safety critical real-time systems require evidence on the functional and timing correctness of their system components. The use of many-cores poses important challenges in providing such evidences, mainly in the timing domain. Hence, despite the advantages of many-cores and the fact that they are nowadays a reality in the embedded system domain (e.g. Tilera Pro64, Kalray MPPA), their use in real-time environments relies on finding efficient ways to deal with timing correctness issues.

Safety-critical real-time systems rely on *incremental qualification* that allows each system component to be subject to formal certification in isolation and independently from other components, with obvious benefits for cost, time and effort. Current safety critical real-time systems enable incremental qualification by using standardized system software architectures, such as the *Integrated Modular Avionics* (IMA) [4] in the avionics domain. IMA guarantees incremental qualification by providing *robust space and time partitioning* that make the functional and timing behavior of each application be unaffected by other applications, reducing the cost of the certification process. To do so, applications are encapsulated into *software partitions* (SWPs), as defined in the ARINC 653 standard [4].

SWPs have been devised for running in single-core platforms. Each SWP is assigned a dedicated time window in which it enjoys exclusive access to processor resources (e.g., bus and memory). Unfortunately, when moving towards parallel execution on many-cores, *SWPs do not provide the desired time isolation properties.* The fact that several SWPs can simultaneously access shared processor resources creates time interferences among them and, as a result, the use of a dedicated time window per SWP fails in guaranteeing time isolation. This directly impacts timing certification, increasing its cost since when a new SWP is added or changed the entire system needs to be validated. Providing isolation among applications is key to exploit the performance opportunities of many-cores into safety critical real-time avionics systems while containing timing certification costs. We make the following contributions:

- We introduce *parallel software partitions* (pSWP) that specify how the interaction among SWPs in the accesses to software and hardware (computation and communication) resources is controlled to enable incremental qualification. pSWP specification maintains single-core ARINC653 principles in many-cores, enables deriving time-composable WCET estimates and reduces integration-time effort, while taking into account the complexity of hardware implementing the pSWP spec-

ification.

- We propose a hardware realization that satisfies pSWP requirements for clustered processor architectures. Our realizations are based on the novel concept of *Guaranteed Resource Partitions* (GRPs), which defines an execution environment composed of a set of resources (cores, memory devices, etc.) in which a SWP runs, avoiding or bounding interferences among parallel applications.

Overall, the combined use of pSWP and GRPs enables incremental qualification in the time domain for IMA systems running on many-core and better exploits performance potential of many-core. We present a many-core architecture that supports GRPs, focusing on the design of two key hardware shared resources: the network-on-chip (NoC) and the memory controller. Moreover, we evaluate our proposal with a system comprising two *real* ARINC653-compliant parallel avionics applications, 3D Path Planning (*3DPP*) and Stereo Navigation (*SteroNav*), demonstrating that pSWP and GRP fully isolate intra-SWP activities among different SWPs, while inter-SWP effect is reduced to less than 1%.

2. INTEGRATED MODULAR AVIONICS

IMA enables incremental qualification by providing *robust space and time partitioning* to avionics applications. In a time- and space-partitioned avionics system the functional and the timing behavior of each application is isolated from the other applications. This makes applications' behavior *composable* and not affected when the other applications of the system are updated or changed. *Functional isolation* (e.g. space partitioning) prevents any unauthorized service to access to the data sources of other services. *Time isolation* guarantees that the timing behavior, and so the Worst-Case Execution Time (WCET) estimate of an application, is not affected by the presence of other applications. Time isolation in IMA is mainly driven by the ARINC 653 standard, which encapsulates each avionics application into a SWP as shown in Figure 1(a). A SWP comprises one or several processes that share the same memory address space The interaction among processes of the same and different SWPs, in a single core system, is subject to the following ARINC 653 principles that preserve time isolation.

2.1 Interference among SWPs

ARINC 653 imposes time and space isolation among SWPs, i.e. among processes associated with different SWPs. Isolation covers both communication and computation activities.

Communication. For communication among SWPs ARINC 653 defines inter-SWP communication means that use queues and messages to exchange data among applications. The destination of an inter-SWP communication is a SWP, not a process within it. The source, destination, size and deadline for inter-SWP communications is contained in the configuration tables developed and maintained by the system integrator. Inter-SWP communication among SWPs imposes the order in which SWPs are executed. This makes the *scheduling of SWPs fixed and pre-defined at system integration time.* As guaranteed by ARINC 653, SWP developers can assume that the data coming from other SWPs are available before the execution of their SWP. This is fundamental to ensure that the timing behavior of a SWP is independent of the SWP producing its input data.

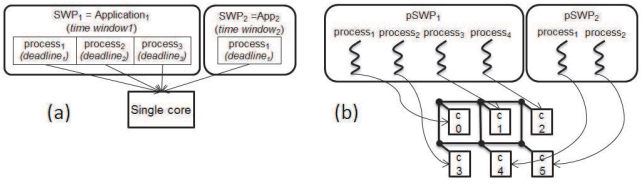


Figure 1: (a) Time partitioning as defined in ARINC653. (b) 2 pSWP comprising 4 and 2 processes and their mapping to a 6-core multi-core

Computation. For each SWP, ARINC 653 assigns a *CPU capacity*, implemented in the form of a time window. Each SWP is allocated one time window during which the system *exclusively* executes processes belonging to that SWP, with no interferences from processes of other SWPs.

2.2 Interference among processes of a SWP

It is worth noting that, processes belonging to the same SWP require no time partitioning, hence it is possible that, within a SWP, processes interfere with each other.

Communication. Intra-SWP communication, i.e. communication among the processes in the same SWP, uses buffers, commonly implemented with global variables. ARINC653 provides mutual exclusion and synchronization mechanisms for accessing to those communication buffers.

Computation. Each of the processes comprising the SWP has an associated *deadline.* Scheduling a process in a given SWP occurs exclusively during the time window of that SWP. ARINC does not specify how processes are scheduled in SWP.

3. ARINC 653 AND MANY-CORES

In single-core execution models, the use of time partitioning mechanisms (named *time capacity* in ARINC 653 nomenclature) provide time isolation to SWPs. This is so because shared (hardware) resources such as the communication bus, memory controller or peripherals are accessed by only one given SWP during its time window. Serialization also simplifies the access to software shared resources like buffers. Unfortunately, this is not the case in many-core execution models in which the simultaneous execution of SWPs makes software and hardware resources to be shared at the same time by multiple processes belonging to different SWPs (see Figure 1(b)). This makes that the timing behavior of one SWP can be affected by other SWPs due to interferences accessing shared resources, hence breaking time partitioning provided by ARINC 653 to enable incremental qualification. Therefore, certain control when accessing to shared resources is required. Although, some hardware resources can be replicated so that interferences among SWPs do not arise, this principle cannot be extended to other resources such as chip pins – one of the most expensive resources in a processor–, hence inevitably other mechanisms are required to deal with inter-SWP interferences.

Providing full timing isolation among SWPs in many-core systems is complex if at all possible. Instead, in order to contain qualification costs in many-core environments, our approach provides time composability (1) to handle the local activities among the processes of a given SWP; and (2) to account for the effect among global activities from different SWPs. Our approach also provides time compositionality

to account for the effect that the global activities of a given SWP may incur on the local activities of others, ensuring that the timing behavior of different SWPs can be easily integrated when deploying many-cores. To that end we introduce two concepts: *Parallel Software Partition* (pSWP) and *Guaranteed Resource Partition* (GRP).

pSWP specifies the properties required in the timing behavior of SWPs and their processes when executed in many-core architectures. pSWP ensures that a WCET estimate for each process can be derived disregarding the time impact of inter-SWP interferences. pSWP covers the access to shared software (logical) resources such as buffers and queues and the access to shared hardware resources (e.g. bus).

For software resources shared among the processes of a SWP, ARINC 653 defines synchronization mechanisms (e.g. semaphores) for guaranteeing an exclusive access. In this case, pSWP leaves the timing analysis tool accounting of the impact on the WCET estimation of access contention [27, 19]. For software resources shared among SWPs, pSWP specification does not add any constraint since ARINC 653 already prevents it, guaranteeing that SWPs runs to completion. To do so, all data required to execute the SWP must be available before the SWP starts executing.

For hardware resources, pSWP specify that inter-SWP impact must have an *additive* nature such that the application WCET in isolation can be easily augmented at integration time with such inter-SWP effect, Δ_{inter}.

We propose that the actual enforcement of pSWP properties is done by the hardware. GRP is a hardware *island of execution* in which pSWP execute, guaranteeing that interactions among SWP are not allowed, while the interaction among processes in different islands is limited to facilitate the estimation of their WCET. Hence, GRPs remove the need to control interactions among all running SWPs when computing the WCET of each SWP. In this paper we present many-core architecture based on the concept of GRP.

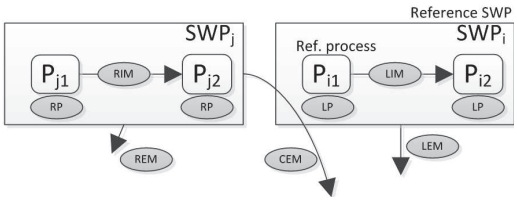
4. PARALLEL SOFTWARE PARTITIONS

Next we illustrate how SWP interaction in shared hardware and software is controlled under pSWP specification.

4.1 Shared Software Resources

ARINC653 allows processes of the same SWP to interfere with each other. In that respect, pSWP add no extra constraints. Hence, if the processes of a given pSWP share some logical resource, their accesses have to be controlled by using ARINC synchronisation mechanisms, e.g. semaphores. The implementation of these synchronisation mechanisms must be predictable, like in [10, 9] so timing bounds of the application execution can be derived. The use of synchronisation mechanisms in parallel execution must be taken into account by the WCET estimation analysis [27].

The communication across SWPs occurs through shared queues. Compliance with ARINC653 standard ensures that by the time the consumer SWP starts, its producer SWP has ended, preventing any conflict in the accesses to the software resources. It may be the case that a given SWP_i receives input messages from two parallel-running SWPs, SWP_j and SWP_k. The fact that there is a different queue for every pair of communicating SWPs, i.e. $SWP_j \rightarrow SWP_i$ and $SWP_k \rightarrow SWP_i$, also prevents any conflict in the accesses to inter-SWP shared software resources among SWPs.



(a)

Reference proc. (P_{i1})→ / offending proc./SWP↓	Computation (LP)	Intra-SWP Comm (LIM)	Inter-SWP Comm (LEM)
Computation (LP)	Bound or carry out parallel WCET estimation		Remove
Local Intra-SWP Comm. (LIM)			
Remote Intra-SW Comp. (RP)	Remove		
Remote Intra-SWP Comm. (RIM)			
Remote Inter-SWP Comm (REM)			Bound
Crossing Inter-SWP Comm (CEM)	Make additive		

(b)

Figure 2: Different conflicts among two SWPs and how they are handled by pSWP specification.

4.2 Shared Hardware Resources

The interaction among processes can be on computation (P) resources or communication (M) resources. pSWP specify the effect on processes of intra-SWP (I) and inter-SWP (E) communication and computation activities. Figure 2(a) shows two SWPs that are executed in parallel, from which SWP_i is taken as reference SWP. Both SWPs comprise two processes. Process P_{i1} is taken as reference process. In Figure 2(b) columns show the activity carried out in the reference process P_{i1} in the reference SWP_i, which includes local intra-SWP communication (LIM) and local processing (LP); and locally generated inter-SWP communication (LEM). The first two rows show the activities carried out by other processes in the reference SWP, that is P_{i2} in the example. The remaining rows show the activity carried out in the other SWP (SWP_j), which includes remote processing (RP), remote intra-SWP communication RIM and remote inter-SWP communication, which includes two types. The first type of communication may need some resources of SWP_i to be carried out and it is called crossing inter-SWP communication (CEM). And the second type of inter-SWP communication that does not require any resources assigned to SWP_i is called remote inter-SWP communication (REM).

4.2.1 Intra-SWP interaction

ARINC 653 does not impose any constraint on the interaction in the access to hardware resources among processes of the same SWP. Since only one process can execute at time in single-core processors, the interaction among processes is reduced to run-after effects i.e a process P_{i1} depends on the state left by preceding task P_{i2} in the stateful resources. In a many-core processor, the processes of a given SWP execute in parallel sharing hardware resources and hence causing more interference on each other. pSWP extends ARINC 653 by controlling the interaction in terms of communication and computation of processes of the same SWPs.

Computation. On the one hand, the computation that processes P_{j1} and P_{j2} carry out is considered remote computation (RP) for SWP_i. pSWP specify that RP must not introduce any impact (i.e. must be removed) on local processing (LP) or (local) intra-SWP communication (LIM) of

P_{i1} and P_{i2}. On the other hand, pSWP specify how the interaction among processes in a given SWP is controlled (LP effect on LP). The problem arises in the access to shared hardware resources (e.g. a shared bus) since the slowdown that a process suffers due to contention on that resource depends on the load the other processes put on that resource. As a result, the WCET estimate that may be derived for a process becomes dependent (non time composable) of the behaviour of the other processes.

A set of hardware techniques [21, 22, 6] already exist to bound the maximum delay each request of a task (process in our case) may suffer from other tasks in the access to each shared resource. When deriving the WCET estimate for a task this delay is assumed for each request, hence making the WCET of the task independent of the load the others put on each resource.

Alternatively, combined, a.k.a. multi-process, WCET analysis [16, 14] can be carried out. This requires analysing all processes to be run in parallel in the different cores tracking their accesses to the different shared resources to determine whether the accesses from each task would interact with others. In theory, this analysis leads to tighter WCET estimates but it is more complex and WCET estimates for a process depend on the other processes, so if a process in the SWP changes, all of its processes have to be re-analysed.

pSWP do not enforce any mechanism to control inter-process interaction. pSWP only requires that WCET bounds can be derived taking into account inter-process interferences.

Communication. Communication among two processes P_{j1} and P_{j2} in a different SWP_j is considered remote intra-SWP communication (RIM). pSWP specify that RIM must not introduce any impact (i.e. it must be removed) on local processes (i.e. P_{i1} and P_{i2}) communication or computation. That is RIM is restricted to SWP boundaries so interaction with other SWPs is avoided and time isolation is guaranteed at SWP level. Restricting the effect of intra-SWP communication on other SWP enables scalability and facilitates incremental qualification. At hardware level, segregating the activities of processes in different SWPs is doable and, indeed recommended, since keeping activities local increases performance and power efficiency.

Communication among processes belonging to the same SWP (i.e. local communication) is performed through shared memory. The accesses to communication resources are controlled similarly to computation resources, i.e. by upper-bounding the effect of inter-process interference or carrying out a multi-process WCET analysis.

4.2.2 Inter-SWP interaction

Communication among SWPs is performed through inter-SWP communication methods such as message passing. While intra-SWP activities can be kept local, inter-SWP activities involve at least 2 SWPs: the sender and the receiver. Moreover, the communication among them may require using communication resources assigned to other SWPs.

For instance, let us assume 4 SWP (SWP_1, SWP_2, SWP_3 and SWP_4), with SWP_1 communicating with SWP_4 and SWP_2 executing after SWP_1, and SWP_4 executing after SWP_3 (see Figure 3). Under this scenario, the inter-SWP communication between SWP_1 and SWP_4 is a CEM for SWP_3 since it uses resources assigned to SWP_3, e.g. NoC. Next we present three different interaction scenarios:

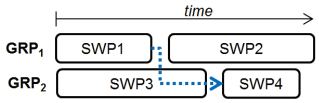


Figure 3: Example of execution of SWP over time

I *Intra-SWP (local) computation and communication effect on Inter-SWP communication ($RIM/LIM \rightarrow REM/CEM$)*

Inter-SWP communication uses both, the resources assigned to other processes in its own SWP and in other SWPs. pSWP specify that the effect that inter-SWP communication may suffer from the processing and intra-SWP communication throughout all of its traversal path to reach the destination memory is removed. This is achieved by a new concept called *transparent execution* provided by GRPs. One way to achieve transparent execution is by giving priority to crossing inter-SWP communication over intra-SWP activities. Section 5 explains in detail transparent execution as implemented in GRPs.

II *Inter-SWP communication effect on intra-SWP (local) activities ($CEM/REM \rightarrow RIM/LIM$)*

Enabling incremental qualification in the timing domain requires being able to derive WCET estimates for the processes of one SWP such that those estimates: 1) do not depend on remote (REM) inter-SWP communication; and 2) the dependence on crossing inter-SWP communication CEM is limited. Failing to do so would imply that WCET estimates would depend on the particular processes running in other SWPs, thus breaking timing composability.

The main idea to achieve this is letting CEM proceed with higher priority without changing the state of shared hardware resources so that intra-SWP activities simply are delayed by the duration of inter-SWP communication and then resumed. The *additive* delay intra-SWP activities suffer can be easily accounted at integration time since **inter-SWP communication characteristics are known at integration time**.

III *Interaction among inter-SWP communication ($CEM/REM \rightarrow CEM/REM$)*

Eventually, several inter-SWP communications can occur simultaneously and compete for shared resources, either idle or in use by another SWP. The solution pSWP use to bound the timing effect that one inter-SWP communication may have on other inter-SWP communications is to use hardware that arbitrates among them in such a way that the effect of interaction can be bounded. This is done, for instance, by using time predictable arbitration policies such as round robin and accounting for the maximum arbitration delay at analysis time[21]. Similarly, TDMA arbitration policies serve the purpose of covering this constraint of pSWP[13].

4.3 WCET and Time composability under pSWP

There are four pillars of pSWP specification, which are enforced by GRPs, that make intra-SWP effect on execution time composable and inter-SWP effect on execution time compositonality: 1) The effect of intra-SWP activities of different SWPs are isolated from each other. 2) The effect of the inter-SWP communication of one SWP over the inter-SWP communication of another SWP is bounded. 3) Inter-SWP activities are prioritized over intra-SWP activities making the former to suffer no impact due to the latter (transparent execution). And 4) the effect that intra-SWP

activities suffer from inter-SWP communications is additive and can accounted for at integration time. This is the consequence of the transparent execution to be provided by GRPs.

Transparent execution makes that the computed WCET estimate of an application in isolation can be simply augmented by an *additive factor* that bounds the increment on the WCET estimate due to system integration, i.e. due to inter-SWP communication when the application is integrated into the system. This is shown in see Equation 1.

$$WCET_{integration} = WCET_{isolation} + \Delta_{inter} \quad (1)$$

$WCET_{integration}$ is the final WCET estimate of the application after system integration, $WCET_{isolation}$ the WCET estimate of the application computed in isolation, while Δ_{inter} bounds inter-SWP interference.

Interestingly, transparent execution makes SWP timing behaviour independent of the *particular pattern of inter-SWP communication* of other crossing SWPs. Instead, once a SWP is computed a Δ_{inter} it is time composable with any other SWP that incurs on the former less than that Δ_{inter}.

The benefits at system integration are that pSWP specification allows computing the WCET estimation for each task in isolation. Each task allocates an interval Δ_{inter_i} to enable crossing inter-SWP communications. The fact that the information of the inter-SWP communications is known at integration time and the fact that pSWP make the effect of inter-SWP communication additive on the WCET computed in isolation, simplifies validating the timing behaviour at integration time.

At deployment time, when a given SWP_j is updated leading to SWP'_j, if SWP'_j's effect on any other SWP SWP_i, $\Delta_{inter(j' \rightarrow i)}$, is smaller than the effect generated by SWP_j, $\Delta_{inter(j \rightarrow i)}$, then SWP'_j can be integrated (composed) in the system without requiring reanalysing any existing SWP. Analogously, SWP'_j should also be able to allocate at least the same time as SWP_j for crossing communications.

5. GUARANTEED RESOURCE PARTITIONS

We propose that many-core processor architectures tailored for use in safety critical real-time systems introduce a new hardware feature called *Guaranteed Resource Partition* (GRP). GRP defines an execution environment composed of a set of processor resources, including cores, NoC resources, memory, etc., in which SWPs run, providing the desirable time composability properties as defined before.

The requests generated among processes belonging to the same SWP, i.e. intra-SWP communication requests, as well as instruction fetch memory requests or process private data accesses, are not allowed to exceed GRP boundaries. This effectively avoids remote intra-SWP activities to interfere with local intra-SWP activities, and thus simplifies deriving time-composable WCET estimates. To that end, each GRP has a private memory region that is accessed by intra-SWP requests without interference from/to other GRPs. Moreover, the NoC design must guarantee that there is no path from cores to memory that exits GRP boundaries.

5.1 Main timing aspects of GRPs

GRPs rest on two main principles: *time predictability* in the access to shared resources and *transparent execution* between intra-SWP and inter-SWP communication. At core level, we assume a design free of timing anomalies [35].

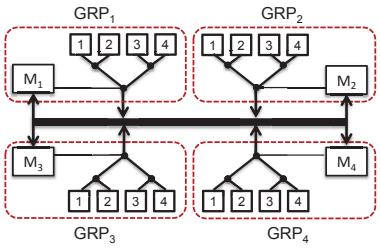


Figure 4: Clustered design (4 clusters each with 4 cores)

1. Time Predictability. A shared hardware resource is said to be time predictable if (1) the time a request takes to be serviced by that resource, once it has been granted access to it, is bounded; and (2) the time a request has to wait to have the access granted is also bounded.

2. Transparent execution. One way to consider the impact of inter-SWP communications on intra-SWP ones (and vice-versa) is assuming that they interfere with each other. This would require assuming that, at analysis time, for every intra-SWP communication a potential conflict may occur with an inter-SWP communication (and vice-versa), which would lead to pessimistic WCET estimates (quantitative figures are provided in Section 6). Instead, we propose transparent execution of inter- and intra-SWP communications where intra-SWP communications are assumed not to compete with any inter-SWP one for WCET estimation. Inter-SWP communication effect is later accounted for at integration time. *This can be done because the applications (SWPs) that are affected by other inter-SWP requests are known at system integration time, when the different SWPs are mapped into the many-core.*

In order to implement transparent execution we propose the memory device and the NoC to provide mechanisms to *freeze* local GRP communications, i.e. intra-SWP requests, instruction requests and process' private data accesses. On the event of an inter-SWP communication, GRP resources are 'frozen' for local requests letting the inter-SWP communication requests to proceed. That is, inter-SWP communications are prioritized over intra-SWP activities. This, on the one hand, makes that inter-SWP communications suffer no slowdown due to interferences in the use of resources. On the other hand, since inter-SWP communication is known statically at system integration and GRP components are time predictable, the impact of inter-SWP communication requests when traversing the NoC and the memory device can be easily determined [21, 31, 22, 6].

Those shared processor resources implementing the freeze mechanism to provide transparent execution must guarantee that the resource state is not affected by the execution of inter-SWP communication requests, so the contribution of intra-SWP communication requests to the WCET estimate remains the same when running the application in isolation and in conjunction with others applications. This is not the case, for instance, for shared caches, in which the access of inter-SWP communication requests may change the cache state, making intra-SWP communication requests vary its timing behavior with respect to running the application in isolation. In this case, the resource would require implementing, for instance, cache partitioning techniques [21].

5.2 Implementation aspects of GRPs

GRP properties can be attained by deploying clustered architectures [1], which organize processor resources into islands such that accesses to local resources are faster than to remote resources, see Figure 4. Communication among cores and from/to memory has to be performed in such a way that the interaction among clusters is controlled (i.e. keeping intra-SWP activity within the bounds of the virtual cluster). We focus on two of the most critical hardware shared resources of a many-core: the Network-on-Chip (NoC) and the memory. In this section we analyze them in the context of physically clustered architectures.

5.2.1 Physical GRPs: NoC Design

Clustered architectures usually deploy hierarchical NoCs: a first-level NoC connects cores that compose a cluster and one or several NoC levels connect clusters. Figure 4 shows an example of a clustered design deploying a two-level NoC: a first-level composed of a tree, and a second-level composed of a bus.

A proper hierarchical NoC design provides *isolated communication islands* in which different communication requests, i.e. intra- and inter-SWP, use different NoC levels that do not interfere among them. In Figure 4, the memory address space of the SWP executed in GRP_i resides in memory M_i and so intra-SWP requests will only use its corresponding first-level NoC, i.e. a tree, without interfering with other GRPs. When a SWP wants to communicate with another SWP, the requests traverse the second-level NoC, i.e. a bus, and the message is directly stored in the memory resource corresponding to the GRP in which the destination SWP will run in the future without affecting other clusters. Note, however, that both communication requests (i.e. intra-SWP and inter-SWP) conflict in the memory device. Section 5.2.2 discusses the memory design in the context of GRPs. The different NoC levels must also provide time predictability. In particular, the latency of each request to cross the NoC must be have a *worst case traversal time* ($WCTT$) bound. We consider that communication in the NoC is organized into flows. Each flow comprises a set of packets, which are further split into flits whose size depends on the NoC implementation. A flit is the minimum flow control unit in the NoC. E.g. a load request to memory is a packet of 64 bytes that can be divided into four 16-byte flits. We define L as the number of flits of the packet. The maximum number of flows contending for resources in a given router at a given time instant is called Zc [24]. A given flow Z_i has L_{Z_i} flits.

The $WCTT$ is the sum of two factors: The *zero load latency* (zll) [7] and the *NoC Request Interference Delay* (NoC_{RID}). The former provides the traversal time of a NoC request assuming zero interferences. The latter provides the maximum time a packet may be delayed due to contending flows in the network when accessing the main memory. Moreover, we consider that requests and responses use different networks (as in [34, 25]) so the same analysis can be applied to requests and responses.

The first-level NoC design considers a wormhole-based tree implementing $N-1$ simple pipelined 2-to-1 routers, with a traversal time of D_{router} cycles, to connect N cores [26], so each core requires $log_2(N)$ hops to reach the memory or the second-level NoC. In such a NoC design Zc equals 1, so the maximum time a message is blocked at each hop is determined by the number of flits of the contending message (L_{Z_i}). Equation 2 computes the factors required to derive the $WCTT$ of a tree ($WCTT_{tree} = NoC^{tree}_{RID} + zll_{tree}$).

$$NoC^{tree}_{RID} = \sum_{i=1}^{log_2(N)} L_{Z_i}$$
$$zll_{tree} = (log_2(N) \times D_{router}) + (L-1) \quad (2)$$

For a 4-core cluster with $D_{router} = 2$ and $L = 4$ we have $zll_{tree} = (2\times1)+(4-1) = 5$ and $NoC^{tree}_{RID} = \sum_{x=1}^{log_2(4)} 4 = 8$.

The second-level NoC is for inter-SWP communications, since intra-SWP communications are not allowed to leave each cluster. Inter-SWP communications coming from a cluster are serialised in their access to the bus. For the second-level NoC we use a non-pipelined bus with a latency D_{bus} implementing a round robin arbitration policy [21]. In a bus, the maximum time a message is blocked is set by the latency of the bus (D_{bus}), the number of flits of each contending message (L_{Z_i}) and Zc that equals the number of GRPs minus one ($Zc = N_{cl} - 1$). Equation 3 computes the factors required to derive the $WCTT$ for a bus ($WCTT_{bus} = NoC^{bus}_{RID} + zll_{bus}$). In [13] authors show how round-robin arbitration achieves comparable results to TDMA in terms of average and guaranteed performance. In the setup from Figure 4, we have $zll_{bus} = 8$ and $NoC^{bus}_{RID} = 24$, assuming $D_{bus} = 2$ and $L = 4$.

$$NoC^{bus}_{RID} = \sum_{i=1}^{Zc} D_{bus} \times L_{Z_i}; zll_{bus} = D_{bus} \times L \quad (3)$$

In a clustered architecture the number of cores per GRP is fixed, and this determines the maximum parallelization level that the SWP can exploit. In [20], we show that GRPs can be implemented with regular architectures, such as mesh-based many-cores, as a means to provide more flexibility. The use of XY routing allows defining virtual clusters comprising set of cores and the use virtual channels allow implementation of transparent execution hence accomplishing GRP properties.

5.2.2 Memory Design

ARINC 653 communication methods are implemented through memory, so that memory requests generated by inter-SWP communication and local-GRP (intra-SWP) memory accesses, may potentially conflict in the access to memory affecting each other behavior.

We propose a memory controller in which intra-SWP and inter-SWP requests are put in different queues, giving higher priority to inter-SWP requests and freezing intra-SWP requests until pending inter-SWP requests are serviced. Also, our memory controller design has to enable bounding the impact that inter-SWP requests have on other inter-SWP requests, as well as the impact that intra-SWP requests generated by a process may have on the requests of another process of the same SWP. In the case of intra-SWP requests, this is achieved by having one request queue per core – which in fact are the intra-SWP requesters – and using a time-predictable arbitration policy, such as round robin [21]. Similarly, by having one request queue by each potential generator of inter-SWP requests, i.e. clusters, together with the use of a predictable arbitration it is enough

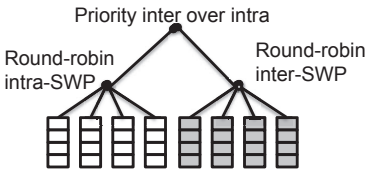


Figure 5: Structure of memory request queues.

to bound the effect of inter-SWP memory requests on other inter-SWP memory requests.

As a result we will have a structure of requests as shown in Figure 5. Requests coming from intra-SWP are organized per core and arbitrated using round-robin. Inter-SWP coming from other clusters are split per cluster and arbitrated using round robin. Finally, to implement transparent execution, inter-SWP requests are prioritized over intra-SWP ones, which are frozen if needed to favor the former. Using separate queues among inter- and intra-SWP requests ensures that, inter-SWP memory requests do not change hardware state visible to intra-SWP requests when they are frozen and resumed. Note that these multiqueue structures can be implemented with a *single* physical queue and the proper use of pointers [33].

Similarly to the NoC, the memory controller must be time predictable, i.e. the WCET estimate accounts for the *memory worst case response time* (Mem_{WCRT}), which expresses the maximum time a memory request can take due to interferences, in our case interference among intra-SWP communication requests from the same SWP and among inter-SWP communication request from different SWP.

The Mem_{WCRT} is composed of the sum of two factors [22]: (1) The *request execution time* (Mem_{RET}) and (2) the *Memory request interference delay* (Mem_{RID}). The former provides the amount of time a request takes to be completed assuming no interferences. The latter provides the maximum time a request may be delayed due to other memory requests. In this paper, we use a notation similar to the one used in [22]. The memory request interference delay is given by $Mem_{RID} = \sum_{x=1}^{NumQ} t_{LID}$ where t_{LID} is the longest issue delay that a memory request may suffer considering the generic timing constraints described in the JEDEC standard [12] and $NumQ$ the number of request queues. In case of intra-SWP communication requests, $NumQ$ will be determined by the number of processes in a GRP that can simultaneously issue a request, which is bounded by the number of cores in the GRP; in case of inter-SWP communication requests, $NumQ$ will be determined by the number of SWPs that can simultaneously issue a request.

Mem_{RET} depends on timing constraints of memory device operations (e.g. row buffer activation, read, write, precharge). It is not shown due to its long expression. We refer the reader to [22] for a detailed explanation of Mem_{RET}.

5.3 From $WCTT$ and MEM_{WCRT} to WCET

5.3.1 Computing the WCET Estimation in Isolation

GRPs enable deriving bounds for every access to the NoC and memory. This allows taking into account the contention in the hardware shared resources with no changes in the static timing analysis tool. The access latency of each request to the NoC and memory is augmented with a factor $UBD_{intra}^{cluster}$ (computed below) that upperbounds the maximum interference a request may suffer due to processes' requests.

If measurement-based timing analysis tools are used, during the testing phase the process under study is run in isolation. Every time an intra-SWP request to memory is ready it is artificially delayed by the architecture so it suffers $UBD_{intra}^{cluster}$, which can be carried out with a technique called worst-case mode [21]. From the traces obtained during this execution in isolation, a WCET estimate for the task is obtained. At deployment time, the hardware is instructed not to introduce any artificial delay. Note that the artificial delay introduced during testing upper-bounds the contention that the processes can suffer during deployment time [21].

The Upper-Bound Delay (UBD) of each request represents the maximum delay a request to NoC and memory resources can suffer due to interferences. The UBD of intra-SWP requests ($UBD_{intra}^{cluster}$) depends on the $WCTT$ internal to the GRP and Mem_{WCRT}. Equation 4 shows the $UBD_{intra}^{cluster}$ for the clustered design.

$$UBD_{intra}^{cluster} = WCTT_{tree} + Mem_{WCRT} \quad (4)$$

Inter-SWP requests instead, are not only affected by intra- and inter-SWP requests belonging to the same application, but also by inter-SWP requests belonging to other applications. Thus, the UBD of inter-SWP requests ($UBD_{inter}^{cluster}$) depends on both, the $WCTT$ internal and external of the GRP and Mem_{WCRT}. Equation 5 shows the $UBD_{inter}^{cluster}$ for the clustered NoC design.

$$UBD_{inter}^{cluster} = WCTT_{tree} + WCTT_{bus} + Mem_{WCRT} \quad (5)$$

5.3.2 Computing Δ_{inter}: NoC and Memory Impact

At system integration time, the WCET estimate of one application computed in isolation ($WCET_{isolation}$) can be affected by inter-SWP communication as expressed in Equation 1. Concretely, inter-SWP requests may delay intra-SWP requests because of their higher priority in NoC and memory. Note that the impact that inter-SWP requests may have on other inter-SWP requests coming from other SWP is already considered in the $WCTT$ used to compute the $WCET_{isolation}$.

As described in Section 2, IMA systems impose that the amount of data transferred in an inter-SWP communication from the source to the destination SWP is known at system integration time, so the application development becomes independent from the system integration. This allows computing the WCET increment (Δ_{inter}) of an application due to interferences that intra-SWP requests may suffer in NoC and memory due to inter-SWP communication requests.

Δ_{inter} is computed using Equation 6 for the clustered architecture designs presented in this paper. In this equation P is the set of SWPs that can simultaneously send inter-SWP communication requests to the GRP in which the destination SWP runs and N_{inter_i} is the number of inter-SWP communication requests of the source SWP i. Note that in the clustered architecture intra-SWP requests do not use the bus(see Figure 4), so we only need to address the interference in the memory ($\Delta_{inter}^{clustered}$). If this were not the case, i.e. intra-SWP request use the bus, we would simply add an additional factor to account for the interference in the bus.

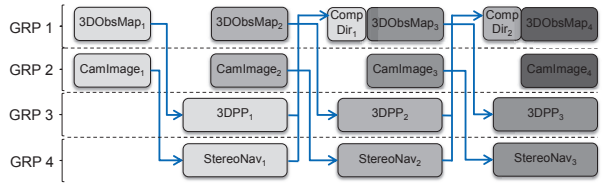


Figure 6: Case study application.

$$\Delta_{inter}^{clustered} = \sum_{i \in P} N_{inter_i} \cdot Mem_{WCRT} \quad (6)$$

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

Hardware Setup. All experiments presented in this section are executed on a cycle-accurate simulator compatible with PowerPC ISA binaries and based on the *SoCLib* simulation infrastructure [5] and the *gNoCSim* cycle-accurate flit-level NoC simulator [2]. We model a 16-core processor clustered architectures presented in Figure 4 implementing 4 GRPs with 4 cores each. We implement a hierarchical NoC, with a tree and a bus as first and second level NoCs, considering $D_{router} = 1$ and $D_{bus} = 2$. The NoC design fulfills the $WCTT$ shown in Section 5.2.1 ($zll_{tree} = 5$; $NoC_{RID}^{tree} = 8$; $zll_{bus} = 8$; $NoC_{RID}^{bus} = 24$). Finally, the simulator also models separated instruction cache and data write-through cache of 64 KB each in each core and four 256MBx16 DDR2 SDRAM 400B memory controllers, one per GRP, implementing two queues each (one per communication type). Higher priority is given to the queue used by inter-SWP requests. We assume that CPU frequency doubles memory frequency. This configuration provides a $Mem_{WCRT} = 42$ processor cycles.

Parallel Avionic Applications. pSWPs and GRPs are evaluated using a *real A653-compliant avionic system* provided by Honeywell International and composed of two parallel avionic applications, *each containing 4 processes (threads)*: 3D Path Planning (*3DPP*) and Stereo Navigation (*StereoNav*). Both applications are used for the navigation of Unmanned Aerial Vehicles (UAV). Moreover, the system also includes two applications for data generation: *3DObsMap* and *CamImage*. The former provides the 3D grid obstacle map required by 3DPP; the latter provides the two images (maps) required by StereoNav. Finally, 3DPP and StereoNav outcomes are compared in *CompDir* application. The communication among applications is performed using inter-SWP communication requests. Figure 6 shows how all five applications are executed in a software pipelined manner. Under this scenario, 3DPP is only affected by inter-SWP requests sent by 3DObsMap and StereoNav is only affected by inter-SWP requests sent by CamImage. In both cases, 3DObsMap and CamImage transmit the data that 3DPP and StereoNav will require in the next pipeline iteration.

WCET estimates of applications are derived with measurement-based techniques. Concretely, the architecture introduces the *WCET computation mode* [21], in which at analysis time intra-SWP and inter-SWP requests are artificially delayed by an *upper bound delay* (UBD) as defined in equations 4 and 5 respectively. By doing so, the resultant execution time can be considered as an *Worst-Case Execution Time Bound* or simply *WCB*. At deployment time, the WCET computation mode is deactivated so a request suffers only actual delays, which are bounded by UBD (equations 4 and 5).

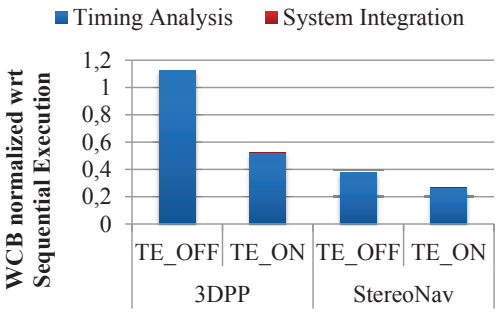


Figure 7: WCB of 3DPP and StereoNav when activating and deactivating transparent execution.

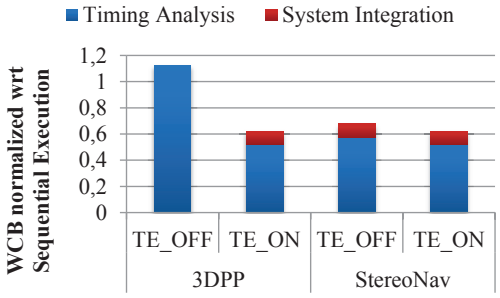


Figure 8: WCB of 3DPP and StereoNav activating and deactivating transparent execution. The number of inter-SWP requests increased by 100x.

6.2 Impact of intra-SWP Communication

GRPs prevent intra-SWP activities from being affected by the intra-SWP activities generated by other SWPs executed on different GRPs. To illustrate this, we run each application in isolation, i.e. no other application is executed in other GRPs, and we measure its execution time. In a second experiment we run all applications simultaneously as shown in Figure 6 with each application mapped into a different GRP. In this second experiment we collect execution times and discount the effect of inter-SWP communications. Our results confirm that in both cases the execution times *were exactly the same*, evidencing that the integration of several SWPs can be done without any impact of their intra-SWP activities on other SWPs.

6.3 Impact of Inter-SWP Communication

One of the central elements of pSWP specification is transparent execution, which allows considering as an additive factor (Δ_{inter}) the impact that inter-SWP requests have on intra-SWP requests at system integration (see equations 1 and 6). This section illustrates that this considerably reduces WCB of applications. We compute the WCB of 3DPP and StereoNav assuming two different strategies:

1. Assuming that no transparent execution mechanism is implemented and so the WCB includes the effect that inter-

SWP communications have on intra-SWP ones and vice-versa [17]. That is, at analysis time, for every intra-SWP communication it is assumed that a conflict may occur with an inter-SWP communication, and vice-versa.

2. Assuming transparent execution in which intra-SWP requests do not compete with inter-SWP requests when computing the WCB. The impact of inter-SWP interference is accounted later at integration time using Equation 1.

Figure 7 shows the WCB of 3DPP and StereoNav assuming the two strategies presented above, i.e. with and without transparent execution (labeled as TE_ON and TE_OFF respectively), for the modeled processor architecture (shown in Figure 4). All values are normalized w.r.t. the sequential execution time of 3DPP and StereoNav in which no interference among intra-SWP and inter-SWP requests occurs. In this case, the complete system executes sequentially in a single core. Figure 7 also shows the WCB portion of each application coming from the timing analysis, i.e. $WCET_{isolation}$, (in blue) and the portion of the additive factor (Δ_{inter}) coming from the system integration (in red).

Overall, transparent execution considerably reduces the WCB of both applications. Assuming at analysis time that every intra-SWP request is affected by an inter-SWP request leads to pessimistic WCET estimates. In particular 3DPP, for which not using the mechanism makes the WCB of the parallel version worse than the sequential version, increasing the WCB by 13% when executed on the modeled architecture. This is not the case of StereoNav, in which the WCB of the parallel version is better than the sequential one, reducing it by 62%. When we enable the transparent execution mechanism, the WCB of both applications is reduced w.r.t. the sequential execution. In case of 3DPP, the WCB is reduced by 48%; StereoNav further reduces the WCB by 74% on the modeled clustered architecture.

The portion of the WCB coming from the additive factor Δ_{inter} represents less than 1% in both applications. This is due to the fact that the number inter-SWP requests that affects 3DPP and StereoNav (coming from 3DObsMap and CamImage respectively) at system integration is relatively small w.r.t. intra-SWP requests. To illustrate the impact of inter-SWP requests, we repeat the same experiment presented in Figure 7 but artificially increasing the number of inter-SWP requests suffered by 3DPP and StereoNav by 100x. The results are shown in Figure 8.

We observe that the WCB with no transparent execution mechanism remains exactly the same. This is so because in this case the WCB already accounts for the worst interference scenario, i.e. that each intra-SWP request is affected by an inter-SWP request. In case of using the transparent execution mechanism, the portion of the additive factor in the WCB of both applications increases as well, 10% in case of the 3DPP and 6% in case of StereoNav. However, despite the significant increment of inter-SWP request, the WCB is still lower than when not using the mechanism.

Therefore, we can conclude that accounting for inter-SWP request impact at system integration reduces considerably the WCB estimates of applications and allows the application to better exploit performance benefits of many-core processors.

It is important to remark that if the application does not have enough task level parallelism to exploit all cores in the GRP this leads to underutilization of resources. In order to improve GRP occupancy in clustered architectures, it is possible to simultaneously run several SWPs in one GRP with certain considerations, i.e. with the use of hardware techniques to control the load that other tasks (processes) put on the shared resources, at the cost of more pessimistic WCET estimates. In [20] we evaluate the impact of execution of multiple SWP in the same GRP.

7. RELATED WORK

Current COTS multicores have been shown not to have a time composable WCET by default. In this regard, authors in [23][18] quantify the delay suffered due to interferences in shared processor resources of a COTS multicore. Fuchsen [8] performed a similar analysis, but focusing on the hardware and software related interference channels between SWPs in multi-core based IMA platforms.

The literature in the area of analysis of hardware shared resource contention in multicore is vast. At system level, several analysis frameworks have been developed to compute worst-case access time bounds [29, 30]. These frameworks model one off-chip shared resource that can process only one request at a time and in which requests cannot be split. It is assumed that on-chip shared resources (e.g. core-to-cache bus, caches, ...) are replicated or partitioned across tasks. This makes that tasks suffer no contention accessing on-chip resources. Further it is assumed that the accesses to the off-chip shared resource are synchronous (i.e. the accessing task is stalled while the access is performed). The focus is on a specific task model in which tasks are divided into superblocks for which maximum and minimum access bounds and execution time bounds can be derived.

Under this scenario, the access to the shared resource is assumed to be arbitrated by either a TDMA bus [29], a dynamic arbitration bus [28] or an adaptive bus arbiter [30]. For those cases in which the arbiter is dynamic, the load that a task puts on the shared resource affects other tasks access time. Other authors [28] propose different approaches to derive per-task bounds to the number of accesses in a period of time. While the number of accesses that a task generates to the resource can be considered intrinsic to the task (i.e independent of the co-runners) as long as caches are partitioned, its frequency of access depends on how often the co-runners delay the task requests, dependence that is captured by the presented models. With dynamic arbiters time-composability cannot be guaranteed since the WCET derived for a task depends on its co-runner tasks. Time-composability is of paramount importance to enable incremental qualification as required by ARINC653, and hence it is the objective of our designs, preventing us from using dynamic arbiters. Further, we focus on shared on-chip resources and the main memory, which handle multiple requests and naturally split cache misses (the requests) into several memory commands that are parallelized across requests, preventing us from using the analysis frameworks presented above. Finally, we use real unmodified avionics applications, which do not follow the superblock model.

Several efforts coming from the WCET community have focused on providing combined (i.e. multitask) WCET estimates for tasks sharing a bus and a cache [16, 14]. This can be used at the process level (i.e. among the processes of a given application) as shown in Figure 2(b). Applying it across applications would break time composability and hence incremental qualification.

At hardware level, we identified two main approaches to deal with contention [32]. The first approach relies on de-

signing a custom platform targeting a specific application with timing constraints, as it is the case for the time-triggered [15], the PRET [3], and the CompSOC [11] architectures. The second approach, although it still requires hardware changes, focuses on adapting general-purpose platforms to allow the execution of those applications with timing constraints. This is the case of the MERASA [17] approach. We note that some special features included in the application-specific architectures like scratchpads require modifications in the application's code, challenging portability of legacy applications. Further, the growing cost of developing and manufacturing chips makes the use of application-specific architectures only relevant for high volume products [32], which is not typically the case for the avionics domain. Therefore, in this paper we have used as baseline the MERASA architecture, on top of which we implemented the novel concept of transparent execution.

8. CONCLUSIONS

Time partitioning, when deployed on many-cores does not time isolate SWPs: the presence of a SWP in the many-core affects the timing behavior of other SWPs due to uncontrolled simultaneous access to shared hardware resources.

Parallel software partitions or pSWP, specify how inter-SWP interactions are controlled to isolate SWPs. In particular, pSWPs prevent local intra-SWP activities from affecting (or being affected by) remote intra-SWP activity. The impact of inter-SWP activities, is made *additive* such that at integration time different independently developed and time analyzed SWPs can be brought together to form a system with minimum effort. pSWPs rely on *guaranteed resource partition* or GRP. GRP defines an execution environment composed of a cluster of processor resources in which SWPs run, providing the desirable timing isolation properties among intra-SWP activities and making inter-SWP activities to have an additive nature. This is done by providing mechanisms to freeze intra-SWP and local GRP requests to let inter-SWP requests proceed, allowing to consider the impact of inter-SWP communication at system integration time as required by pSWP (Δ_{inter} addend).

Our proposal is evaluated in a real avionic system composed of two parallel applications provided by Honeywell, 3D path planning and stereo navigation, on a 16-core processor with hierarchical NoC. Inter-SWP communications overhead is reduced to less than 1% for both applications.

Acknowledgments

The research leading to these results has been funded by the European Union Seventh Framework Programme under grant agreement no. 287519 (parMERASA) and by the Ministry of Science and Technology of Spain under contract TIN2012-34557. Miloš Panić is funded by the Spanish Ministry of Education under the FPU grant FPU12/05966.

9. REFERENCES

[1] *Kalray MPPA 256 Many-Core Processor, http://www.kalray.eu/ products/mppa-manycore,.*
[2] *NanoC: http://www.nanoc-project.eu.*
[3] Precision Timed (PRET) Machines. `http://chess.eecs.berkeley.edu/pret`.
[4] *ARINC Specification 653: Avionics Application Software Standard Standard Interface, Part 1 and 4*, 2012.
[5] Soclib, http://www.soclib.fr/trac/dev, 2012.
[6] B. Akesson, et. al. Predator: a predictable sdram memory controller. In *CODES+ISSS*, 2007.
[7] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Elsevier, May 2004.
[8] R. Fuchsen. How to address certification for multi-core based IMA platforms: Current status and potential solutions. In *DACS*, 2010.
[9] M. Gerdes, et. al. The split-phase synchronisation technique: Reducing the pessimism in the WCET analysis of parallelised hard real-time programs. In *RTCSA*, 2012.
[10] M. Gerdes, et. al. Time analysable synchronisation techniques for parallelised hard real-time applications. In *DATE*, 2012.
[11] K. Goossens, et. al. Virtual execution platforms for mixed-time-criticality systems: The compsoc architecture and design flow. *SIGBED Rev.*, 10(3):23–34, October 2013.
[12] B. Jacob, et. al. *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., 2007.
[13] J. Jalle, et. al. Deconstructing bus access control policies for real-time multicores. In *SIES*, 2013.
[14] T. Kelter, et. al. Static analysis of multi-core tdma resource arbitration delays. *Real-Time Systems*, 2013.
[15] H. Kopetz and G. Bauer. The time-triggered architecture. *Proc. of the IEEE*, 91(1):112–126, 2003.
[16] Yan Li, et. al. Timing analysis of concurrent programs running on shared cache multi-cores. In *RTSS*, 2009.
[17] MERASA. *EU-FP7 Project: www.merasa.org*.
[18] Jan Nowotsch and Michael Paulitsch. Leveraging multi-core computing architectures in avionics. In *EDCC*, 2012.
[19] H. Ozaktas, et. al. Automatic wcet analysis of real-time parallel applications. In *WCET workshop*, 2013.
[20] M. Panic, et. al. Parallel many-core avionics systems. Technical Report UPC-DAC-RR-CAP-2014-6, UPC, 2014.
[21] M. Paolieri, et. al. Hardware support for wcet analysis of hard real-time multicore systems. In *ISCA*, 2009.
[22] M. Paolieri, et. al. Timing effects of the memory system in real-time multicore integrated architectures: Problems and solutions. In *TECS*, 2012.
[23] P. Radojkovic, et. al. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. In *HiPEAC*, 2012.
[24] D. Rahmati, et. al. Computing accurate performance bounds for best effort networks-on-chip. *IEEE Trans. on Computers*, 62(3), 2013.
[25] J. Rattner. Single-chip cloud computer: An experimental many-core processor from Intel Labs.
[26] A. Roca, et. al. Enabling high-performance crossbars through a floorplan-aware design. In *ICPP*, 2012.
[27] C. Rochange, et. al. WCET analysis of a parallel 3D multigrid solver executed on the MERASA multi-core. In *WCET workshop*, 2010.
[28] S. Schliecker, et. al. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *DATE*, 2010.
[29] A. Schranzhofer, et. al. Timing analysis for TDMA arbitration in resource sharing systems. In *RTAS*, 2010.
[30] A. Schranzhofer, et. al. Timing analysis for resource access interference on adaptive resource arbiters. In *RTAS*, 2011.
[31] Zheng Shi, et. al. Schedulability analysis for real time on-chip communication with wormhole switching. In *IJERTCS*, volume 1, 2010.
[32] J. Sparsoe. Design of networks-on-chip for real-time multi-processor systems-on-chip. In *ACSD*, 2012.
[33] Y. Tamir and G. L. Frazier. High-performance multiqueue buffers for VLSI communication switches. In *ISCA*, 1988.
[34] Tilera Corporation. *Tile Processor, User Architecture Manual, release 2.4, DOC.NO. UG101*, 2011.
[35] R. Wilhelm, et. al. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 28(7), 2009.