Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement

Jan Nowotsch[1], Michael Paulitsch[1], Daniel Bhler[2], Henrik Theiling[3], Simon Wegener[4], Michael Schmidt[4]

[1] EADS Innovation Works, Munich, Germany, {Jan.Nowotsch, Michael.Paulitsch}@eads.net
[2] Cassidian, Friedrichshafen, Germany, Daniel.Buehler@cassidian.com
[3] SYSGO AG, Mainz, Germany, hth@sysgo.com
[4] AbsInt Angewandte Informatik GmbH, Saarbruecken, Germany, {swegener, mschmidt}@absint.com

Report 2013-10 September 2013

Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement

Jan Nowotsch*, Michael Paulitsch*, Daniel Bühler†, Henrik Theiling‡, Simon Wegener§ and Michael Schmidt§
*EADS Innovation Works, Munich, Germany, {Jan.Nowotsch, Michael.Paulitsch}@eads.net
†Cassidian, Friedrichshafen, Germany, Daniel.Buehler@cassidian.com
‡SYSGO AG, Mainz, Germany, hth@sysgo.com
§AbsInt Angewandte Informatik GmbH, Saarbruecken, Germany, {swegener, mschmidt}@absint.com

***Abstract*—The performance and power efficiency of multi-core processors are attractive features for safety-critical applications, as in avionics. But increased integration and average-case performance optimizations pose challenges when deploying them for such domains. In this paper we propose a novel approach to compute a *interference-sensitive Worst-Case Execution Time (isWCET)* considering variable accesses delays due to the concurrent use of shared resources in multi-core processors. Thereby we tackle the problem of temporal partitioning as it is required by safety-critical applications. In particular, we introduce additional phases to state-of-the-art timing analysis techniques to analyse an applications resource usage and compute an interference delay. We further complement the offline analysis with a runtime monitoring concept to enforce resource usage guarantees. The concepts are evaluated on Freescale's P4080 multi-core processor in combination with SYSGO's commercial real-time operating system PikeOS and AbsInt's timing analysis framework aiT. We abstract real applications' behavior using a representative task set of the EEMBC Autobench benchmark suite. Our results show a reduction of up to 75% of the multi-core Worst-Case Execution Time (WCET), while implementing full transparency to the temporal and functional behavior of applications, enabling the seamless integration of legacy applications.**

Keywords*—worst-case execution time, multi-core, safety-critical real-time systems, avionics

I. Introduction

In recent years, the decreasing relative cost of electronics and the pace of electronics development have led to the adoption of modern Commercial Off-The-Shelf (COTS) computing architectures in avionics. Increasing demand for energy efficiency and performance will further introduce multi-core processors, and especially Multi-Processor Systems on Chip (MPSoCs) in safety-critical domains, such as avionics. Besides certification issues with COTS hardware [1], multi-core processors introduce additional problems related to the isolation of functionally disjoint applications.

In avionic systems the Integrated Modular Avionics (IMA) concept is a standard system architecture integrating applications of different criticality on the same hardware platform. The so-called partitioning concept is introduced for management and analysis of safety aspects and to enable the use of incremental certification and development paradigms, cf. [2], [3]. Partitioning ensures spatial and temporal separation of unrelated functions, comprising isolation of address spaces and bounding of temporal interferences respectively, cf. [4], [5]. Spatial separation is considered to be solved since it is not solely required in safety-critical systems. For instance, techniques, such as Memory Management Units (MMUs) and Input/Output Memory Management Units (IOMMUs), are common practice in today's computing platforms. However, multi-core processors introduce challenges for temporal partitioning [6], complicating the exact determination of the timing behavior for shared-resource accesses, such as Network-on-Chip (NoC) and shared caches.

In this paper we tackle the problem of temporal partitioning for multi-core processors, since it is not sufficiently solved but strongly required for safety-critical systems. For this purpose we introduce a Worst-Case Execution Time (WCET) analysis for multi-core processors considering shared resource interferences between applications. Intuitively explained, we split timing analysis into core-local and shared-resource interference delay analysis. We further refer to these bounds as *interference-sensitive Worst-Case Execution Time (isWCET)*. We complement offline analysis with runtime resource usage enforcement to bound the maximum inter-core interference. Using this novel approach, we answer the question of how to efficiently compute a multi-core WCET and guarantee temporal and resource usage behavior for an arbitrary number of hard real-time applications. In contrast to related approaches we are able to guarantee deadlines for arbitrary hard real-time applications, while avoiding resource privatisation and mutual analysis of in-parallel scheduled applications. Hence enabling independent analysis of applications supporting incremental development and certification. We evaluate our approach on a modern COTS MPSoC, Freescale's P4080, using extensions to SYSGO's Real-Time Operating System (RTOS), PikeOS, and AbsInt's timing analysis framework, aiT.

The paper is structured as follows. We define our basic terminology and the resource capacity enforcement in Section II. Thereafter we describe the details of the WCET analysis extensions and how to compute a multi-core timing bound in Section III. In Section IV and V we present our implementation and evaluate the approach. We discuss the results and related research in Sections VI and VII. We conclude the paper with a short summary and future work topics in Section VIII.

II. Resource Capacity Enforcement

Temporal isolation is not a new topic in safety-critical systems. It becomes much more complex when such systems deploy multi-core processors. In single-core systems temporal isolation with respect to shared resources has to be considered only if Direct Memory Access (DMA)-capable peripheral devices or interrupts are enabled [7]. By means of abstraction layers, device drivers and disabling of unpredictable interrupts it is possible to avoid any unintended parallel accesses of processing cores and peripheral devices to memory. In case of multi-core processors, parallel resource accesses are inherent and hardware arbitrated. Firstly, for reasons of maintaining competitive advantage the exact arbitration in COTS processors is unknown in most cases. Secondly, preventing or controlling interfering accesses of low criticality applications may not be possible, due to their loose certification requirements, cf. [5]. Hence the temporal behavior of applications is hard or even impossible to predict and timing analysis becomes increasingly complex.

We target an integrated approach of runtime mechanisms and multi-core worst-case timing analysis. The runtime mechanisms are required to enforce defined resource capacities per scheduling entity. The implications of the capacity guarantees are further used to compute additional delays due to concurrent resource accesses - *interference delays*. We calculate an interference-sensitive WCET bound, based on the single-core WCET and the interference delays.

The concepts are designed such that they (1) are functionally and temporally transparent to applications, (2) avoid mutual analyses of in-parallel scheduled tasks and (3) allow to utilise parallel resources. Functional transparency avoids any ability of applications to control the runtime mechanisms, which is inevitable to fulfill safety requirements. Temporal transparency prevents influences on the timing of an application. Avoiding mutual analysis, greatly reduces complexity, since each application can be analysed separately. Furthermore, it is essentially required to enable incremental certification and independent development of applications, cf. [2], [3]. As already mentioned, multi-core processors are, amongst others, interesting due to their performance. Consequently, leveraging built-in parallelism is inevitable for efficient utilisation of the platform.

We further use the terms process to refer to the smallest scheduleable unit, which is used for all concepts. We are aware of different notations for such a unit, depending on application domain and abstraction layer. For instance General-Purpose Operating Systems (GPOSs) use the terms thread and process, where processes are separated from each other and threads share the address space of their parent process. In the avionics domain the ARINC 653 standard [4] is commonly used, its entities are partitions and processes. Partitions can consist of multiple processes and shall be separated from each other. The separation requirements for partitions are more strict than for GPOS since each partition gets guarantees when and how long it is executed.

To partition a resource it is necessary to quantify its capabilities. We abstract a resource, using its *capacity* (C_R), and each requesting process (P_i) is assigned to a certain share (C_{P_i}) of it. In order to provide a safe mechanism, applicable to real-time systems, each share needs to be guaranteed. We further refer to C_{P_i} as processes limit or capacity. For describing the approach we introduce the concepts of *limitation*, *monitoring* and *suspension*.

A. Limitation

Limitation is an offline mechanism used to assign the capacity C_{P_i} per process. This limit is a numerical representative for a specific resource parameter, e.g. bandwidth or number of accesses. The actual parameter depends on the target resource. To provide a safe partitioning the limits are required to bound the resource usage of processes. Furthermore, the resource shall not be over-utilised. Hence the sum over all process capacities must be bounded by the overall resource capacity C_R, cf. Equation 1. The computation of the limits is part of the extended worst-case analysis and discussed in Section III.

$$C_R \geq \sum_{i=0}^{N-1} C_{P_i} \tag{1}$$

B. Monitoring

Runtime monitoring is used to observe the resource usage of a process. Once a process reaches a limit, monitoring is responsible for triggering the suspension mechanism.

Any delays between limit violation and process suspension have to be taken into account, e.g. via a sufficient safety margin in the limits. Monitoring as such has to be transparent, in both, functional and temporal dimension. Functional transparency avoids any control from processes to the monitoring mechanism. Temporal transparency is required to prevent influences on the execution time of processes, which otherwise would additionally complicate timing analysis.

C. Suspension

If a violation of the limit has been detected a suspension action is triggered. The action is responsible to prevent further accesses to the shared resource, avoiding interference to other processes.

Requirements to suspension are functional transparency and deterministic reaction timings. The latter is required to account for resource usages that a process can issue until it is finally suspended. Hence it has to be included into the limits as well.

D. Example: Concurrent NoC Accesses

To illustrate the operation of the mechanism, we describe an example with two processes P_0, P_1 that compete for NoC accesses. Figure 1 shows the processes P_0 to P_{N-1} executed in a periodic manner on $core_0$ and $core_1$. All processes are scheduled within a periodic time frame. Each process has an execution window, during which time and resource usage are guaranteed. This can be understood as a simplistic view of time-triggered execution, as used in avionics. Processes P_0 and P_1 are shown in more details. Each diagram plots the accumulated number of memory accesses (C) over time under different conditions. Continuous lines represent normal execution, dashed lines depict abnormal and dotted lines partitioned conditions. Horizontal dot-dashed lines represent the limits C_{P_i}. The finalisation of a process is marked by an *x*.

Under normal conditions (continuous lines), both processes finish within their process window. For abnormal conditions (dashed lines) P_0 issues significantly more accesses than in

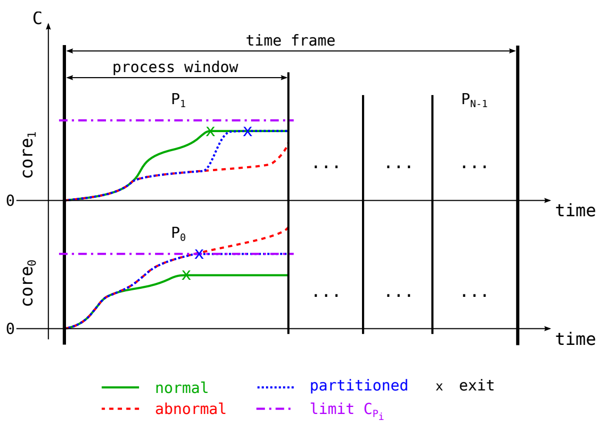

Fig. 1. Example showing the resource usage (C) of processes P_0 and P_1 over time, depicting behavior under normal (continuous), abnormal (dashed), and partitioned (dotted) conditions.

the normal case. This can, for instance, be caused by a software error (unbounded loop) or Single Event Upsets (SEUs). Consequently P_0 executes until it is stopped at the end of the process window. Itsadditional accesses interfere with those of P_1 such that P_1 suffers higher delays, causing a deadline violation. To avoid such unbounded temporal impacts between processes we introduce the described partitioning approach (dotted lines). Once P_0 exhausts its limit (C_{P_0}), it is suspended. Consequently, P_1 experiences some increase in execution time, but since this is bounded by C_{P_0}, P_1 is able to finish within its deadline.

In summary, by guaranteeing a resource usage limit per process, the maximum interference with other processes is bounded. Hence, faults of a process do not propagate to other processes, thus temporal isolation and fault containment can be ensured.

III. WCET Analysis

This section is used to describe our novel approach to compute a isWCET bound. This bound includes additional delays due to interferences with in-parallel scheduled processes without requiring mutual analyses of all processes. We will first give a brief overview of state of the art static timing analysis techniques and challenges that arise with modern processor architectures. Based on that, we derive additional analysis blocks necessary to compute a multi-core bound. For this purpose a timing-compositional system with respect to shared resource usage is required.

A. Timing Analysis

Over the last several years, a more or less standard architecture for timing analysis tools has emerged [8], composed of three major building blocks:

- control-flow reconstruction and static control and data flow analyses,
- micro-architectural analysis, computing upper bounds on execution times of basic blocks [9],
- path analysis, computing the longest execution paths through the program [10].

The data flow analysis also detects infeasible paths, i.e. unreachable program points during real execution. This reduces the complexity of the following micro-architectural analysis. Basic block timings are determined using an abstract processor model to analyze the instruction flow through the pipeline accounting for cache hit/miss information. This model defines a cycle-level abstract semantics for each instruction's execution yielding in a certain set of final system states. After the analysis of one instruction has been finished, these states are used as start states in the analysis of the successor instruction(s). Here, the timing model introduces non-determinism that leads to multiple possible execution paths in the analyzed program. The pipeline analysis needs to examine all of these paths. This architecture is implemented by AbsInt's timing analysis tool aiT.

B. Predictability Challenges

Modern embedded processors try to maximize the instruction-level parallelism by specific average-case performance enhancing features. For example pipelines increase performance by overlapping the executions of consecutive instructions. Hence, a timing analysis cannot consider individual instructions in isolation. Instead, they have to be analyzed collectively accounting for their mutual interactions to obtain tight timing bounds. Commonly used performance-enhancing features as caches, static/dynamic branch prediction, speculative execution, out-of-order execution, branch history tables, or branch target instruction caches can cause timing anomalies [11] which render WCET analysis more difficult. Intuitively, a timing anomaly is a situation where the local worst-case does not contribute to the global worst-case. For instance, a cache miss - the local worst-case - may result in a globally shorter execution time than a cache hit because of hardware scheduling effects. In consequence, it is not safe to assume that a memory access causes a cache miss; instead both machine states have to be taken into account. An especially difficult timing anomaly are domino effects [12]: A system exhibits a domino effect if there are two hardware states s, t such that the difference in execution time (of the same program starting in s, t respectively) may be arbitrarily high. The approach presented in the following is based on the idea to separate the analysis of core-local worst-case behavior and the potential worst-case interference delays for accesses to shared resources. This requires timing-compositionality [13]. Consequently, it needs to be proven that the target architecture fulfills the requirements or can be configured in a way that avoids potential sources of timing anomalies.

Concerning processor caches, both precision and efficiency depend on the predictability of the employed replacement policy [14], [15]. Another deciding factor is the write policy (write-through, write-back). The write-back policy is difficult to analyze due to uncertainties in the cache analysis, the precise occurrence of such a write-back operation is not known, increasing the search space. However, a smart system configuration can decrease analysis complexity and increase predictability.

C. Multi-core Analysis

The main contribution of this paper is the computation of the isWCET ($wcet_{is}$) as described in the following. The computation of a timing bound consists of two phases: (1) single-core timing and resource analyses per process and (2) combination of the single-core results of in-parallel scheduled processes to derive the multi-core bounds. The single-core timing analysis includes pipeline and local cache analysis. In addition to this state of the art analysis, a resource analysis has to be performed for each shared resource used by the respective process. Consequently, the single-core analysis of process P_x returns a timing bound, denoted $wcet_{s,P_x}$, and an upper bound for the usage (C_{P_x}) of every utilised shared resource, further denoted Worst-Case number of shared Resource Accesses (WCRA). Both, timing and resource analyses, are implemented using the described standard architecture for timing analysis. The computation of the multi-core timing bound, $wcet_{is}$, is based on the single-core analysis results and the presence of the runtime resource capacity enforcement. For the sake of simplicity we exemplarily focus on only one shared resource, the main memory, including the required interconnect.

The general issue of shared resources are unpredictable access delays that depend on the resource usage of connected devices, e.g. processor cores and peripherals. We argue, arbitration delays to be a crucial, additional factor that influences shared resource access delays. Possible sources for such delays can be the NoC, shared caches, or the memory controller. To compute the additional interference delays it is necessary to determine the worst-case overlap scenario, for concurrent cores. Naturally the worst-case access delay increases disproportionately with a rising number of cores. Hence the worst-case overlap scenario for N cores appears if all requests are issued in parallel. This assumption is true as long as Equation 2 is valid, where d_i denotes the resource access delay if i requests are issued in parallel. In practice Equation 2 can be interpreted such, that no two delays d_i, d_{i+1} exist, where the relative delay d_i is greater than d_{i+1}, normalising to the number of requesters. For architectures where Equation 2 is not valid, the worst-case overlap scenario has to be derived differently. Either it can be derived from the architecture parameters or by analysing all possible permutations, selecting the maximum.

$$\frac{d_i}{i} \leq \frac{d_{i+1}}{(i+1)} \forall i \in \mathbb{N}, 1 \leq i \leq N \tag{2}$$

Figure 2 exemplarily shows how the interference delay is computed for $N = 4$ cores. The x-axis shows the accumulated resource accesses for each process P_i. It can easily be understood that the access delay drops from d_i to d_{i-1} if one process has exhausted its capacity. For instance, all of the accesses of P_0 suffer interference with other processes, since it is the process with the lowest WCRA. Hence, the respective access delay is the maximum delay, d_4 for 4 cores and d_N in the general case. If P_0 exhausts $C_{P_0} = 80$, the accesses of the remaining processes further only suffer from delay d_3. The number of accesses of P_1, that can be accounted with reduced delay can be expressed as the difference of accesses between P_0 and P_1, i.e. $C_{P_1} - C_{P_0} = 30$. The sames applies to the accesses of P_2 and P_3.

The general case, to compute a multi-core bound for process P_x, can be expressed with Equation 3. The accesses of all processes with higher WCRAs than P_x have to be considered as overlaps. Hence the computation of the interference delay stops after x iterations. In order to apply Equation 3 the WCRAs of the processes are required in ascending order, with C_{P_0} being the smallest.

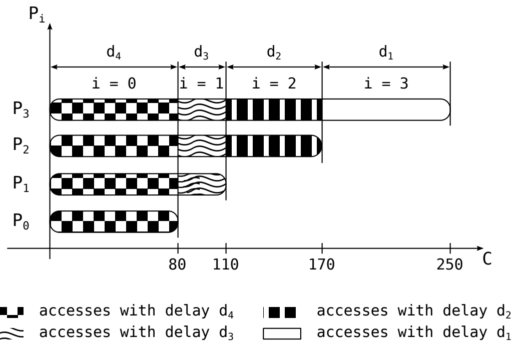

Fig. 2. Computation of the interference delay for process P_3, considering the resource capacities C_{P_0} to C_{P_3} for $N = 4$ processes.

$$wcet_{is}(x) = \overbrace{wcet_{s,P_x}}^{\text{single-core bound}} + \underbrace{d_N \cdot C_{P_0} + \sum_{i=1}^{x} \left(d_{N-i} \cdot (C_{P_i} - C_{P_{i-1}}) \right)}_{\text{interference delay}} \tag{3}$$

Since the runtime mechanisms enforce the process capacities the proposed timing bound computation is safe. Although Equation 3 only covers a single resource it can easily be extended to iterate over multiple resources.

IV. Implementation

In this section we cover the implementation of the described concepts. The WCET analysis extensions are implemented using AbsInt's aiT, the world leading commercial framework for static timing analysis. It has been successfully applied to real avionics applications [16]. The runtime mechanisms have been implemented in SYSGO's PikeOS, a commercial RTOS which has been used in multiple certified projects, like the Airbus A400M [17] and the Airbus A350 [18]. As target computing platform we select Freescale's P4080 as one example for a modern COTS MPSoC, with Freescale having a long tradition of its processors being used in many avionic systems. The eight e500mc PowerPC cores and the CoreNet NoC platform interconnect are from special interest for our work. The NoC of P4080 is considered as black box, since not enough details on design, timing and arbitration are available for in-depth analysis.

A. Limitation

Process capacities can either be assigned using static analysis, measurement-based approaches or manually. We implemented the described multi-core timing analysis in AbsInt's aiT analysis framework. The existing single-core analysis has been extended to calculate the WCRA for memory accesses. The calculation is based on the knowledge whether an access is a cache hit/miss or unknown. Consequently it can be decided if an access is a local access (cache hit) or a memory/shared resource access (cache miss or unknown). In order to provide safe boundaries, accesses marked as unknown are treated as shared resource accesses. Since the absence of timing anomalies in case of memory accesses is assumed, cf. Section III-B, this local decision is reasonable. The WCRA is determined for each basic block. We extended path analysis to solve an Integer Linear Program (ILP) optimising for WCRA. Besides this extension, an appropriate architecture model for the e500mc cores is required. In the current version a prototype, based on an early e600 model is used. Architectural differences have been derived from the processor manuals. The current model does either represent L1 caches or Data Line Fill Buffer (DLFB) and Instruction Line Fill Buffer (ILFB). This results in large overestimations of the resulting WCETs and WCRAs. This is however no limitation to our approach. To apply the tool for the verification of safety-critical systems model refinement and additional tool validation is required.

B. Monitoring and Suspension

Monitoring and suspension are implemented in SYSGO's real-time Operating System (OS), PikeOS. A similar implementation is available for evaluation purposes, using a bare metal OS layer. Both implementations use the built-in processor core performance counter. They are configured to record accesses to the bus interface and trigger an exception once the limit of the process is reached. The corresponding Interrupt Service Routine (ISR) further executes the suspension action, i.e. the corresponding process is stopped and the core is idle until the end of the actual process window. The implementations are encapsulated in a separate driver module. The current implementation does yet ignore outstanding write-backs. In productive implementations, such outstanding requests either have to be handled during analysis, or avoided by invalidating the caches during suspension.

C. Processor Core Configuration

The caches, Translation Lookaside Buffers (TLBs), Branch Target Buffers (BTBs) and Branch History Tables (BHTs) of the e500mc processor cores typically use Pseudo Least Recently Used (PLRU) and First In First Out (FIFO) replacement schemes. This makes them a non-timing-compositional architecture according to [19]. In order to still be able to assume timing compositionality we use the cores in a very deterministic configuration, avoiding any known domino effects. Hence, the branch prediction is switched off, the data cache is used in write-through mode, 2nd-level caches are exclusively used as scratchpad memories, partial cache locking is used to reach Least Recently Used (LRU) replacement, and all TLBs are preloaded to avoid any miss. Avoiding all these sources of domino effects still is no formal proof of timing compositionality but is sufficient to use the platform for evaluation. To cover for possible remaining timing effects, it might be necessary to add some safety margin when using the platform for real systems. Nevertheless, from our limited set of practical experiments, we were not able to observe any remaining effects and we will ignore them in the following. For the use in safety-critical systems and for certification, further analytical and experimental validation is required. Furthermore it shall be noted that the validity of presented approach is independent from the evaluation platform.

TABLE I. P4080 MEMORY ACCESSES LATENCIES FOR INCREASING NUMBER OF CONCURRENT CORES, LATENCIES USED FOR EVALUATION ARE MARKED BOLD

	latency [cycles]							
cores	1	2	3	4	5	6	7	8
read	**41**	75	170	268	296	438	459	603
write	38	**164**	**244**	**463**	**516**	**736**	**782**	**1007**

V. Evaluation

In this section we evaluate the described approach with respect to the reduction of the isWCET. In lack of results for related approaches we fall back to the naive solution as reference. This also requires the evaluation of the core-local analysis phases for timing and resource usage. We conclude the section by demonstrating the runtime effect of the partitioning, by integrating all of the selected benchmarks. We use a set of benchmarks from the EEMBC Autobench benchmark suite [20]. After careful analysis of real avionics applications the selected benchmarks have been identified to adequately represent different applications' behavior.

A. Worst-case Analysis

We use the naive approach as baseline comparison as there are no data for other multi-core WCET approaches available. With the naive approach, each memory access is accounted with the maximum delay ($d_8 = 1007$ for $N = 8$ cores). This is a valid approach, as long as no assumptions on in-parallel applications can be made. The timing bound is computed according to Equation 4.

$$wcet_{\text{naive}}(x) = wcet_{s,P_x} + d_N \cdot C_{P_x} \qquad (4)$$

Table I shows the memory access latencies for read and write operations with increasing number of interfering cores. The data is acquired using the approach described in [21], while mapping all eight cores of the P4080 to the same memory controller. Since the path from processor cores to main memory is very complex it is not possible to obtain real worst-case latencies without in-depth knowledge of the chip design. Hence, the latencies in Table I shall only be understood as indicators. As described, the architecture model, used for the WCET analysis, is not able to represent both, L1 caches and DLFB, ILFB. In order to have a comparable setup for analysis and measurements, we disable the caches at runtime. However, this does not restrict the presented concept or this evaluation, only the absolute values are increased. This will also not affect the general statements of the paper. For evaluation we always use the higher latency, as marked in Table I, since they are the safe decision as long as the exact distribution of read and write accesses is not known.

We compare the analysed timing and resource usage bounds ($wcet_{s,P_x}$, $wcra$) and respective, observed maximum values in Table II. The appropriate bound deviation is relative to the observed values. The results indicate relatively tight bounds for execution and resource usage. Only the *matrix* and *aifftr* benchmarks suffer huge overestimations.

Table III compares the computed multi-core bounds of the naive ($wcet_{\text{naive}}$) and our interference-sensitive approach ($wcet_{is}$). The reduction is relative to the naive approach.

TABLE II. Comparison of statically analysed timing ($wcet_{s,P_x}$) and resource usage bounds ($wcra$), and respective maximally observed execution times (ET) and resource accesses (RA).

benchmark	analysed $wcet_{s,P_x}$ [ms]	analysed $wcra$ [10^6]	observed ET [ms]	observed RA [10^6]	deviation ET [%]	deviation RA [%]
a2time	151	3.2	71	1.7	113.4	88.4
cacheb	389	9.5	371	8.3	5.0	14.2
iirflt	516	13.5	369	8.1	40.0	67.2
rspeed	862	19.3	445	9.3	93.7	108.3
bitmnp	2393	53.8	1106	21.8	116.4	147.2
tblook	2371	56.8	1131	23.1	109.6	145.5
matrix	4707	99.9	230	4.3	1944.5	2249.3
aifftr	7193	190.0	87	2.2	8191.8	8681.8

TABLE III. Comparison of interference-sensitive WCET ($wcet_{is}$) and the naive approach ($wcet_{\text{NAIVE}}$).

benchmark	$wcet_{\text{naive}}$ [ms]	$wcet_{is}$ [ms]	reduction [%]
a2time	2804	2804	0.0
cacheb	8362	7178	14.2
iirflt	11812	9735	17.6
rspeed	17095	12610	26.3
bitmnp	47560	27444	42.3
tblook	50014	28022	44.0
matrix	88524	36250	59.1
aifftr	166604	41813	75.0

As can be seen, the implications of the runtime resource enforcement can reduce the multi-core timing bound up to 75%. Which, even if the absolute timings are huge, is a great reduction, proofing the validity of our approach.

B. Functional Behavior

To demonstrate the functional behavior of monitoring and suspension we use three scenarios:

(1) isolated: where core 0 executes the reference benchmark, while the remaining cores are inactive,
(2) interfered: with additional load on interconnect and memory by interfering benchmarks on cores 1 to 7 without limiting their memory accesses,
(3) partitioned: with the same setup as for (2), but with enabled limitation of cores 1 to 7, according to the analysed WCRAs from Table II.

The reference benchmark, *bitmnp*, is executed on core 0. The benchmarks on cores 1 to 7 are used to introduce interference for scenarios (2) and (3), while our partitioning approach is only enabled for scenario (3). All benchmarks are parametrised with the analysed WCRAs from Table II. To intense the effect of interference, the benchmarks on cores 1 to 7 are executed iteratively, until the reference benchmark is finished. Hence we increase the probability for the benchmarks to reach their resource limit, which otherwise is very unlikely or even impossible due to static analysis. The results are shown in Figures 3, 4 and 5, respectively. Each figure shows a single diagram per core, plotting the resource usage (C) over time. Triangles indicate a suspension due to a limit violation. For evaluation only one process window is used, but of course in practice each core can accommodate multiple processes, cf. Figure 1.

A comparison of the results illustrates the impact of interference and partitioning. While *bitmnp* in isolation finishes after $1071ms$, its execution time is increased to $2876ms$ by

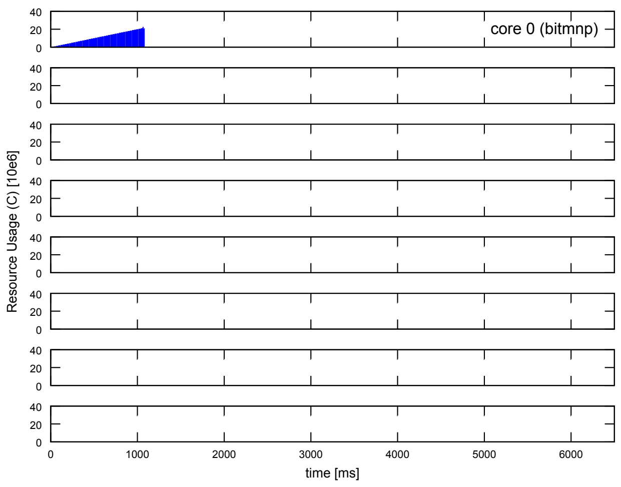

Fig. 3. Observed resource usage (C), executing scenario (1), running *bitmnp* on core 0 without any interference by other cores.

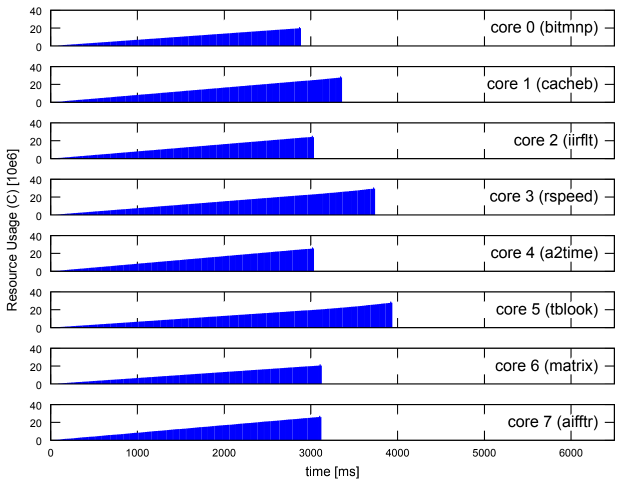

Fig. 4. Observed resource usage (C), executing scenario (2), running *bitmnp* on core 0 with interference by benchmarks on cores 1 to 7.

unlimited interfering cores. Enabling the partitioning causes suspensions on cores 1 to 4, reducing the execution time of *bitmnp* to $2230ms$. The measured WCRAs for cores 1, 2, 3 and 4 further show a deviation of 8 to 12 accesses compared to the configured limits of Table II. This stems from the overhead of the suspension routine as described in Section II. The static analysis of the corresponding routine returns a timing bound of $25.0\mu s$ and a WCRA of 567 accesses.

VI. Discussion

The evaluation has shown the validity and applicability of the presented approach. The results of the single-core timing and resource analysis are considerably high but reasonable having in mind the prototype status of the architecture model. The overestimation for both, timing and resource analysis, is in the same order of magnitude. Comparing the different benchmarks, the results for *matrix* and *aifftr* are significantly higher than for others. This can be explained by the code

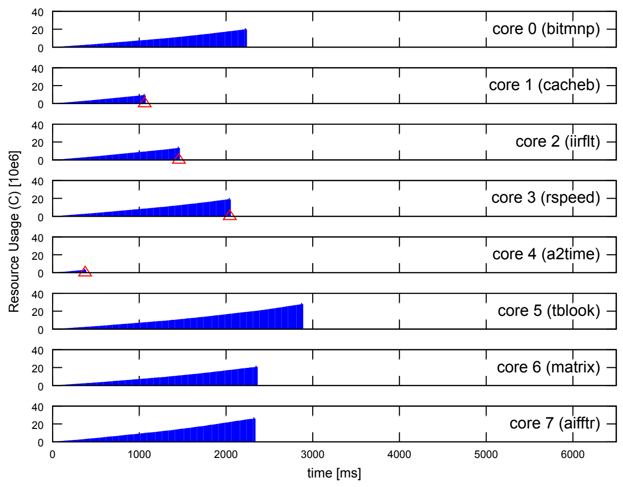

Fig. 5. Observed resource usage (C), executing scenario (3), running *bitmnp* on core 0 with interference by benchmarks on cores 1 to 7 and enabled resource limitation according to the analysed WCRAs of Table II.

structure, containing triangular loops that are hard to analyse. Also, the missing consideration of input data does effect those benchmarks more than others, causing the analysis to examine many paths that are infeasible in practice.

The evaluation of the resulting isWCETs shows a significant reduction of up to 75% over the naive approach. Comparing the reduction for different benchmarks, it can be seen that the resulting effect clearly depends on the individual benchmark characteristics. Precisely, the larger the difference of WCRAs, the higher the WCET improvement. This can be used to optimise system scheduling, for instance, schedule applications such that one process with a lower and one with a higher WCRA are scheduled in parallel. Applying this effect to legacy applications, one can expect applications with only a few core-local accesses to suffer a much higher WCET increase, compared to applications with a relatively small number of shared resource accesses.

The evaluation of the functional behavior proves the runtime effect as described in Section II, the correctness of the implementation and also shows the predicted overhead for suspension. We have further shown how to bound the suspension overhead, analysing the respective routine. In addition, the interrupt latency and outstanding instructions have to be accounted. The interrupt latency for the e500mc cores is limited to $\leq 10cycles$, unless a guarded load or a cache inhibited *stwcx.* instruction is in the last completion queue entry [22]. For the latter cases the latency is determined by the operations' targeted memory location. In general the outstanding operations depend on the application, but in terms of memory accesses the worst-case occurs if every pending instruction is a load or store. Any runtime overhead due to the use of the performance counters is impossible, according to Freescale. Hence an active process does not suffer temporal delays by enabled monitoring.

Besides the multi-core WCET improvements, the described partitioning further provides a safety net [23], which isolates applications in cases of miss-behavior and faults. The so ensured temporal fault containment guarantee is novel for multi-core processors.

In summary, the architecture model is sufficient in order to show the validity of the presented approach. For the judgement of the results it shall be understood that the development of an accurate architecture model is extremely time-consuming. That is why we used a prototype that does not exactly represent the real hardware. However, this does not limit the presented approach and the general statement of the paper, since it only influences the absolute results, but not the order of magnitude for the WCET reduction. Moreover, it is possible to replace the static analysis with a different technique, for instance a measurement-based approach. The applicability only depends on the target application and their certification requirements. In particular, the resulting assurance has to be sufficient for an application's criticality.

VII. Related Work

How to use multi-core processors in real-time systems is an active field of research. Hence, over the past years several very different approaches have been presented.

In [24], [25], Bellosa introduces the idea to leverage built-in counters to acquire additional task runtime information. These data are further used to extend task scheduling to consider memory accesses. Even though we developed our approach independently, this work can be seen as a conceptual foundation. Recently, Yun et al. [26] presented an approach that also uses processor performance counters to determine the number of memory accesses of threads, separating real-time and non-real-time threads.

However, they focus on the response times of non-real-time threads, while our target is to give hard real-time guarantees. They further assume the WCETs and number of cache misses to be given beforehand. Hence we complement their work.

Schranzhofer et al. [27], Pellizzoni et al. [28], and Boniol et al. [29] propose deterministic execution models to control the access to shared resources. The basic concept is to divide program execution into multiple phases and restrict their capabilities. In [27], Schranzhofer et al. compare different resource access schemes based on splitting the execution in acquisition, execution, and replication phases. Each phase gets an execution time and a maximum number of accesses to shared resources assigned. With the different schemes the phases in which communication to shared resources is permitted, are distinguished and evaluated. Pellizzoni et al. [28] propose the PRedictable Execution Model (PREM) architecture for single-core COTS processors, introducing a co-scheduling for shared resources. A program is split into sequences of what they call predictable and compatible intervals. Predictable intervals are restricted such that all data and instructions are preloaded into caches, and system calls and interrupt preemptions are prohibited. Traffic from peripheral devices is only permitted during the execution phase of a predictable interval, resulting in an architecture with very few contention for accesses to shared resources. A comparable approach is presented by Boniol et al. [29]. They also split execution in communication and local execution phases. In contrast to Pellizzoni, they target a multi-core processor and study several tools to realise such an architecture.

It would be interesting to compare the utilisation of such resource privatising approach to ours, since we leave fine grained arbitration to the hardware, allowing utilisation of parallel resources. Furthermore, our approach does not require any changes to the applications.

Another popular approach is joint analysis. To address sharing of resources those approaches analyse the program flows on all cores using the considered shared resource. Therefore detailed knowledge on the state of execution is required. Yan and Zhang [30] apply WCET analysis to multi-core processors with shared L2 caches by analysing inter-thread dependencies. The analysis is based on the program control flow and accounts for all possible conflicts on the shared cache. Li et al. [31] and Hardy et al. [32] extend this analysis by identifying possibly overlapping threads and reducing the number of possible conflicts between overlapping threads, respectively. Li et al. use Message Sequence Charts (MSCs) and Message Sequence Graphs (MSGs) to describe concurrent programs. Hardy et al. use compiler techniques to identify single-usage blocks, consequently only caching blocks that are statically known as reused. Chattopadhyay et al. integrate shared cache and bus analysis in various publications, cf. [33], [34], [35]. The bus analysis is based on a Time Division Multiple Access (TDMA) arbitration. In [35], the authors combine cache and bus analysis with other architectural features such as pipelines and branch prediction.

Our approach is considerably less complex, since it avoids mutual analysis of applications. Instead abstracting an application, using its time and resource requirements. We state this to be essential for future platforms, containing increasing numbers of cores.

Rosen et al. [36] and Paolieri et al. [37] are two examples of approaches which propose changes to the hardware to address the resource sharing problem and its consequences for WCET analysis. Since the focus of this paper is on COTS processors, further details on custom hardware based solutions will not be discussed.

VIII. Summary and Future Work

In this paper we addressed the problem of computing WCET bounds for multi-core processors to enforce temporal partitioning. Intuitively explained, we split the timing analysis into core-local analyses and computation of the worst-case interference delay caused by the use of shared resources. We further extend core-local analysis with additional phases to account for the usage of shared resources. The proposed approach uses runtime resource capacity enforcements to bound the interference between in-parallel executed processes. Using this approach we are able to independently analyse multiple applications, enabling the seamless integration of legacy applications and the use of incremental development and certification.

The validity of the approach has been proven by integrating application blocks, that have been analysed to adequately represent different behavior of real avionics applications. The WCET analysis extensions and runtime mechanisms have been implemented in AbsInt's framework for static timing analysis and SYSGO's RTOS, PikeOS. Both are commercial products, successfully applied to certified projects. We evaluated the approach on Freescale's P4080 MPSoC, showing a reduction of up to 67% of applications' WCET. Great benefits, compared to other approaches, are the true parallel usage of shared resources while avoiding mutual analysis of applications. This enables the utilisation of multi-core benefits while still reducing analysis complexity. Mixed criticality workloads can even be used to adjust the WCET bounds, optimising overall system execution time.

Even though the absolute multi-core bounds are considerably high, this is no limitation to the approach, but can rather be explained by the prototype architecture model. For later use during production, the architecture model would be refined to estimate the WCET much more precisely. To reduce the analysis results in short-term, it would be possible to replace static analysis by an alternative approach without requiring any changes to the presented approach. Assumptions and configurations made to increase the analysability of the architecture are also no limitation to the approach but necessary to cope with the complexity of the processor. We conclude that the requirements of composability are a common issue for timing analysis, already required by nowadays single-core analyses. Hence, the proposed approach does not pose any new assumptions or requirements to computing architecture, applications and overall system compared to existing approaches.

Future work will address the problem of overestimated timing bounds due to variations in resource access delays, targeting increased average system utilisation. Furthermore, we will investigate how to control the effects of DMA-capable I/O devices, extending the presented approach. Based on the approach, scheduling solution will be developed to select applications that should be scheduled in parallel to optimise system resource utilisation.

IX. Acknowledgement

This work was funded within the project ARAMiS by the German Federal Ministry for Education and Research with the funding ID 01IS11035.

References

[1] EASA, "Certification memorandum - development assurance of airborne electronic hardware (chapter 9)," Software & Complex Electronic Hardware section, European Aviation Safety Agency, CM EASA CM - SWCEH - 001 Issue 01, 11th Aug. 2011.

[2] RTCA, *DO-297 - Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*, Std., 2005.

[3] A. Wilson, W. River, T. Preyssler, and W. River, "Incremental certification and integrated modular avionics," *Proceedings of the 27th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, pp. 1–8, 2008.

[4] Aeronautical Radio Inc., "ARINC 653: Avionics application software standard interface," Oct. 2003, 2551 Riva Road, Annapolis, MD 21401, U.S.A.

[5] RTCA, *DO-178C/ED-12C - Software considerations in airborne systems and equipment certification*, RTCA, Inc Std., 2012.

[6] C. Rochange and P. Sainrat, "Multicores and critical systems: Challenges for temporal analysability," *SAE AeroTech Congress & Exhibition*, pp. 1–15, 2011.

[7] J. Littlefield-Lawwill and L. Kinnan, "System considerations for robust time and space partitioning in integrated modular avionics," *Proceedings of the 27th Digital Avionics Systems Conference*, 2008.

[8] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm, "Reliable and precise WCET determination for a real-life processor," in *Proceedings of EMSOFT 2001, First Workshop on Embedded Software*, ser. Lecture Notes in Computer Science, vol. 2211. Springer-Verlag, 2001, pp. 469–485.

[9] C. Ferdinand, "Cache Behavior Prediction for Real-Time Systems," Ph.D. dissertation, Saarland University, 1997.

[10] H. Theiling and C. Ferdinand, "Combining abstract interpretation and ILP for microarchitecture modelling and program path analysis," in *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, Dec. 1998, pp. 144–153.

[11] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker, "A Definition and Classification of Timing Anomalies," in *International Workshop on Worst-Case Execution Time Analysis (WCET)*, F. Mueller, Ed., July 2006.

[12] T. Lundqvist and P. Stenström, "Timing anomalies in dynamically scheduled microprocessors," in *Real-Time Systems Symposium (RTSS)*, December 1999.

[13] R. Kirner and P. Puschner, "Obstacles in worst-case execution time analysis," in *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 333–339.

[14] J. Reineke, D. Grund, C. Berg, and R. Wilhelm, "Timing predictability of cache replacement policies," *Real-Time Systems*, vol. 37, no. 2, pp. 99–122, 2007.

[15] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, and R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," in *Proceedings of Embedded Real Time Software and Systems*, May 2010, pp. 36–42.

[16] S. Thesing, R. Heckmann, J. Souyris, F. Randimbivololona, M. Langenbach, R. Wilhelm, and C. Ferdinand, "An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software," *Proceedings of 33th International Conference on Dependable Systems and Networks (DSN)*, 2003.

[17] SYSGO AG, "Rheinmetall selects DO-178B certifiable PikeOS from SYSGO for A400M project," http://www.sysgo.com/en/news-events/press/press/details/article/rheinmetall-selects-do-178b-certifiable-pikeos-from-sysgo-for-a400m-project/.

[18] ——, "Airbus selects SYSGOs PikeOS as DO-178B reference platform for the A350 XWB," http://www.sysgo.com/en/news-events/press/press/details/article/airbus-selects-sysgos-pikeos-as-do-178b-reference-platform-for-the-a350-xwb/.

[19] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *IEEE TCAD*, 2009.

[20] The Embedded Microprocessor Benchmark Consortium, "EEMBC AutoBench 1.1 benchmark software," http://www.eembc.org/benchmark/automotive_sl.php.

[21] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," *Proceedings of the 9th European Dependable Computing Conference*, 2012.

[22] Freescale Semiconductor, "e500mc core reference manual, all revisions, chapter 4.13," 2012.

[23] B. Green, J. Marotta, B. Petre, K. Lillestolen, R. Spencer, N. Gupta, D. O'Leary, J. Lee, J. Strasburger, A. Nordsieck, B. Manners, and R. Mahapatra, "DOT/FAA/AR-11/2 - handbook of the selection and evaluation of microprocessors for airborne systems," FAA, Tech. Rep., 2011.

[24] F. Bellosa, "Process cruise control: Throttling memory access in a soft real-time environment," University of Erlangen, Tech. Rep., 1997.

[25] ——, "Memory access - the third dimension of scheduling," University of Erlangen, Tech. Rep., 1997.

[26] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha, "Memory access control in multiprocessor for real-time systems with mixed criticality," *Proceedings of 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 299 – 308, 2012.

[27] A. Schranzhofer, J.-j. Chen, and L. Thiele, "Timing predictability on multi-processor systems with shared resources," *Embedded Systems Week - Workshop on Reconciling Performance with Predictability*, 2009.

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

[29] F. Boniol, H. Cassé, E. Noulard, and C. Pagetti, "Deterministic execution model on COTS hardware," *Proceedings of the 25st international conference on Architecture of computing systems (ARCS)*, 2012.

[30] J. Yan and W. Zhang, "WCET analysis for multi-core processors with shared L2 instruction caches," *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 80–89, 2008.

[31] Y. Li, V. Suhendra, Y. Liang, T. Mitra, and A. Roychoudhury, "Timing analysis of concurrent programs running on shared cache multi-cores," *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pp. 57 – 67, 2009.

[32] D. Hardy, T. Piquet, and I. Puaut, "Using bypass to tighten WCET estimates for multi-core processors with shared instruction caches," *Proceedings of the 30th IEEE Real-Time Systems Symposium*, 2009.

[33] S. Chattopadhyay, A. Roychoudhury, and T. Mitra, "Modeling shared cache and bus in multi-cores for timing analysis," *Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems*, 2010.

[34] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Bus-aware multicore WCET analysis through TDMA offset bounds," *Proceedings of the 23rd Euromicro Conference on Real-Time Systems*, 2011.

[35] S. Chattopadhyay, C. L. Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk, "A unified WCET analysis framework for multi-core platforms," *Proceedings of the 18th Real Time and Embedded Technology and Applications Symposium*, 2012.

[36] J. Rosen, A. Andrei, P. Eles, and Z. Peng, "Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip," *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, pp. 49 – 60, 2007.

[37] M. Paolieri, F. J. Cazorla, M. Valero, and G. Bernat, "Hardware support for WCET analysis of hard real-time multicore systems," *Proceedings of the 36th annual international symposium on Computer architecture*, 2009.