A Practical Study on WCET Estimation on Multicore Processors for Avionics Applications

E. P. Freitas*, B. B. Cozer*, C. G. Ferreira*, F. R. Wagner* and T. Larsson**.

*Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, RS, 91501-970 Brazil.
(E-mail: *epfreitas@inf.ufrgs.br; bruno.cozer@gmail.com; cristiano.ec@gmail.com; flavio@inf.ufrgs.br*)
** School of Information Technology, Halmstad University, Halmstad, 301 18, Sweden.
(E-mail: tony.larsson@hh.se)

INTRODUCTION

On the early 2000's, avionics systems started to be employed as integrated software modules embedded in a same hardware, as an evolution of previous concept of having a same base hardware with several hardware cards, each one performing a dedicated task. This concept, defined as Integrated Modular Avionics (IMA), was standardized in (RTCA 2005) and brought a consistent improvement on avionics system design.

IMA systems rely mostly on Commercial Off-The-Shelf (COTS) hardware, leaving most of dedicated and customized tasks to be performed by software and programmable hardware applications. During the recent years, the IMA concept had broad acceptance on the market, especially in civil avionics area. The foundations of IMA design rely on the determinism of the combination of each application and the hardware over which it is running. Depending on the criticality of the function being implemented by an IMA embedded software application, it is mandatory to know the expected behavior of such implementation in such a way that this application will not interfere on other ones running over the same processor and sharing the same hardware resources (Lofwenmark et al. 2014).

The best way to assure this independence and non-interference among applications running over the same hardware is assuring the temporal and spatial separation among them (Weilong et al. 2014). Spatial separation means to assure that each application has its own memory area and this area will not be used by any other application except the one intended to use. Temporal isolation means that a given application will seize the hardware resources to execute only during a given pre-established amount of time and

this amount will not be exceeded in order to do not jeopardize the execution of other applications that will run using the same hardware resources.

Spatial separation is a goal that does not present big challenges to be achieved, since real-time operating systems (RTOS) compliant with ARINC653 standard (ARINC 2006) offer a robust tool set to assure the isolation of memory areas between applications. However, temporal separation is a much more delicate issue to be dealt with. This happens because in hard real-time applications each task has hard deadlines to meet and the real-time operating system shall manage the scheduler in order to avoid any unexpected and not deterministic behaviors. In order to assure such real time performance the OS needs to know what is the effort, in terms of time consumption, that each software application takes. Such metric, is named as Worst Case Execution Time (WCET).

In the other hand, although the introduction of multi-core processors integrated in a single chip (MPSoC) brought many improvements in scalability and power efficiency to computer systems, it also posed some challenges compared to single core processors, like the way to perform execution time calculation (Luque et al. 2012) or even proposing new metrics for system performance analysis (Otoom et al. 2015).

This paper aims to discuss and to analyze alternatives to improve the WCET analysis in order to cope with hardware technology evolution towards the usage of multi-core processors in avionics systems. In this context, shared resources, tasks parallelism, memory access latencies and inter-core communication, which are aspects that also increase the analysis difficulty, will be highlighted.

BACKGROUND AND RELATED WORKS

WCET Analysis in Multicore Platforms

WCET analysis basically consists in finding a safe upper bound on the execution time which satisfies the system time constraints. Most of the WCET estimation techniques involve static analysis of the code or measurements and hybrid analysis (Mitra et.al 2007), (Mushtaq et al. 2013). First techniques were based in pure execution measurements that usually reach a rough estimation, leading to a loss of performance. With the technology advance, this kind of estimations (usually too pessimistic) were put aside and new studies start to be develop to reduce the gap between estimation and actual results.

WCET is a fundamental metric for the development and validation process of safety-critical systems. In avionics industry, which requires high reliability, safety and performance, the DO-178B/C is responsible to guide and audit the software development process, including software reviews and analysis, where WCET plays a major role.

WCET calculation is highly dependent of hardware architecture and resources of the system. Nowadays, WCET analysis has achieved a high level precision in the calculation of sequential programs executing on single-core processors (Wilhelm et al. 2013). There are available consolidated COTS software tools, widely used, to provide WCET analysis. These analyses rely heavily in the time determinism of the software execution flow, based on the assumption that only a single processing entity (either a process or a thread) will execute in a given moment.

However, with the introduction of multi-core systems, this assumption is not valid anymore. Although a processing entity can seize in an exclusive way a core in a given moment, the several cores embedded into a MPSoC (Multi-processor System-on-a-Chip) share other processor internal resources as caches, memories and intra- processor communication buses. The usage of such shared resources in a same moment invalidates the previously accepted assumption and poses a new challenge to estimate an accurate WCET for hard real-time software applications (Nowotsch et al. 2014).

Several efforts were made in order to overcome this constraint. However most of them bring significant draw-backs (Nowotsch et al. 2014) as depicted in Table 1.

Table 1. Drawbacks of possible WCET approaches for multicore processors

Approach	Drawback
Shared resources serialization through TDMA schemes	Inefficient resource utilization due to resource privatization
Customizations in processor hardware architecture	Preventing the usage of COTS processors
Shared Resources Joint Analysis	Hard scalability when using multiple cores and difficult usage of incremental development and certification
Response Time Analysis and resource conflicts delays	Does not consider static scheduling and difficult usage of incremental development and certification
Monitoring Mechanisms using processor counters	Only monitor the impact of non-real time tasks while does not guarantee achievement of hard real time tasks deadlines.

Avionics Certification

Most safety-critical embedded software development needs to comply with guidelines and standards in order to produce artifacts with a minimal level of quality, safety and maturity (Lofwenmark et al. 2014).

Since early nineties it is mandatory for avionics that embedded systems need to comply with standards created to be used as a guide to determine if the software will perform reliably in an airborne environment.

The main standard followed by industry for avionics software systems development is DO-178B - Software Considerations in Airborne Systems and Equipment Certification (RTCA 1992), developed by the safety-critical working group RTCA SC-167 of RTCA and WG-12 of EUROCAE. This standard is a guideline dealing with the safety of safety-critical software used in certain airborne systems. The European agencies refer to this document as ED-12B, as registered by EUROCAE.

During its system specification phase, every avionics software needs to be categorized according with a Design Assurance Level (DAL). The DAL is determined based on a safety assessment process and hazard analysis. A hazard is defined (FAA 2012) as a condition that could foreseeably cause or contribute to an accident, an unplanned event or series of events that results in death, injury, or damages to, or loss of, equipment or property.

A failure hazard assessment (FHA) is performed in order to verify the consequences of a failure condition in the system. The failures conditions categorization considers basically their effects on the aircraft, crew, and passengers as depicted in table below defined in (RTCA 1992).

Based on the definitions of Table 2, safety analysis tasks are accomplished in order to determine the software DAL and are required to be documented in system safety assessments (SSA). The relationship between function failure effects, DAL's and failure rate are represented in Table 3.

Table 2. Failure types and consequences

Failure Type	Failure Consequences
Catastrophic	Failure may cause a crash. Error or loss of critical function required to safely fly and land aircraft.
Hazardous	Failure has a large negative impact on safety or performance, or reduces the ability of the crew to operate the aircraft due to physical distress or a higher workload, or causes serious or fatal injuries among the passengers. (Safety-significant)
Major	Failure is significant, but has a lesser impact than a Hazardous failure (for example, leads to passenger discomfort rather than injuries) or significantly increases crew workload (safety related)
Minor	Failure is noticeable, but has a lesser impact than a Major failure (for example, causing passenger inconvenience or a routine flight plan change)
No Effect	Failure has no impact on safety, aircraft operation, or crew workload.

Table 3. Design Assurance Levels per failure types

Failure Condition	DAL	Maximum Failure Rate per Flight Hour
Catastrophic	A	1.0E-9
Hazardous	B	1.0E-7
Major	C	1.0E-5
Minor	D	1.0E-3
No Effect	E	--

Furthermore, after specific safety analysis (modulated), the hazard could be mitigated by system architecture aiming to decrease the DAL level, since development and certification costs greatly increase as the software criticality level is higher (Pop et al. 2013).

DO-178B (RTCA 1992) allows flexibility regard different styles of software life cycles, and because of that, generally, is difficult to implement the first time. The flexibility raises several abstract aspects that depends of how is choose to deal, could increase the complexity, effort and cost. Independent of the aspects, all process must have defined and documented the exit/entry criteria between development phases.

In order to comply with DO-178, the avionic software must follow the development phases described in the standard. Depends of the software DAL, the phase is not required. Each phase generates a certification artifact. Figure 1 illustrates the trace between certification artifacts required by DO-178B/ED-12B.

In 2012, motivated by the advances in hardware and software technology and as an insistent request by the industrial partners, was published the new airworthiness standard, DO-178C/ED-12C (RTCA 2012). Its content is based on the previous standard, but addressing software development methodologies and issues which were not adequately addressed in DO-178B/ED-12B.

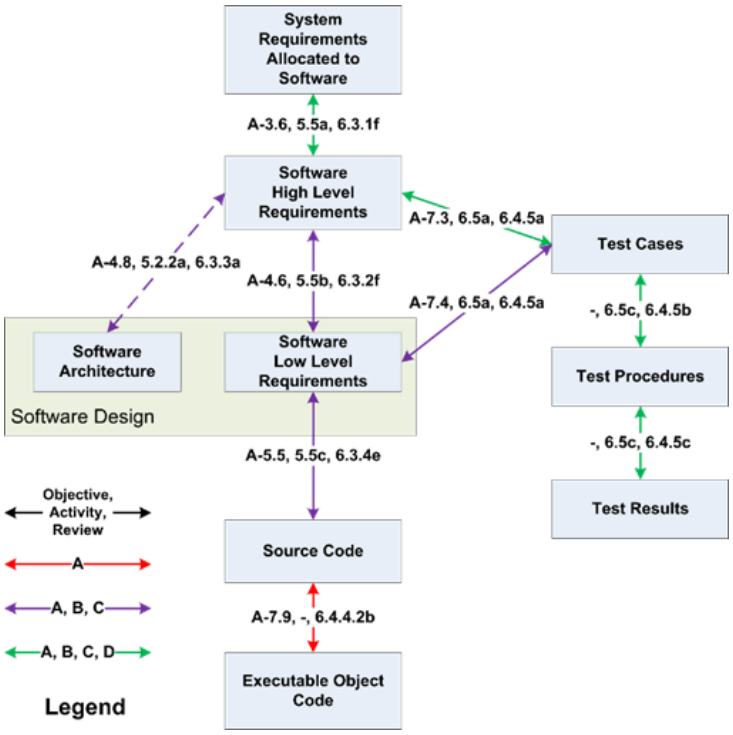

Figure 1: Relationship between DO-178B Certification Artifacts

Focusing in the subject of this paper, the WCET analysis was one of the relevant topics discussed during the meetings to improve the standard. In DO-178B

(RTCA 1992), the WCET analysis was identified as part of reviews and analysis of the source code verification process (section 6.3.4.f - "accuracy and consistency"). Such approach satisfied the goal in the past, when the main programing language was assembly. Today it is impossible to perform this analysis only reviewing the source code, without consider the time constrains, system architecture, memory accessing.

After many discussions regarding this topic, a sentence was added in section 6.3.4.f, requiring that compiler, linker and hardware be assessed for impact on WCET. Also in the introduction to the section on software reviews and analysis (section 6.3), it was remarked that reviews and analysis alone may not completely satisfy some objectives (e.g. WCET, stack analysis) and to achieve that, some tests may be also necessary.

Related Works

Since multicore processors are available since more than a decade, several different approaches already have been discussed on how to take advantage from the increased performance and power efficiency provided by such processor architectures, while still coping with hard real-time requirements.

(Betti et al. 2008) proposed a modified Linux kernel for hard real-time embedded systems. Although this solution succeeds to meet the proposed application deadlines, it relies on resource privatization, what is not a desirable feature since it leads to an inefficient resource utilization as discussed earlier in this section.

(Bastoni et al. 2010) brought concerns from another perspective: scheduler policies for hard and soft real-time applications especially for large multi-core processors, proposing that it may be a good approach to group cores in clusters internally. Additionally, it brings an interesting early concern about execution overhead and the impact of shared resources usage between cores that may impact schedulability and performance of hard real-time applications.

(Luque et al. 2012) discusses a key factor that affects directly the way to design hard real-time embedded systems, especially avionics. The new approach on how to account CPU time in multicore processors is directly related with WCET calculation, that is a fundamental metric for avionics verification and validation, especially regarding applications that are intended to be ported from legacy single-core platforms to multicore platforms.

Intra-processor shared resources pose a challenge to multicore performance for hard real-time embedded applications. (Nowotsch et al. 2014) discussed and proposed a new method of calculating WCET considering the shared resources inside an MPSoC, naming it interface-sensitive WCET (isWCET). Indeed, the interface between multiple cores through the shared resources is found to be a major player in WCET calculations for multicore processors and will be the main focus of this paper analysis.

Besides the actual shared resources that may develop into a bottleneck for the concurrent execution of hard real-time processes, multi-core processors also bring a challenge regarding Execution Overhead time, since a considerable amount of inter-core communications and protocols need to be conducted inside the processor, and the time spent for these communications is added on top of the actual process execution time (Saranya et al. 2014).

METHODS

Since the availability of open-source and academic WCET calculation tools for multi-core platforms is still restricted, in order to assess the impact of different types and amounts of shared resources, it was used a processor simulator tool. CHRONOS (Xianfeng et al. 2007) is a static analysis tool that generates WCET estimations based on SimpleScalar simulator architecture, a widely popular cycle-accurate architectural simulator that allows the user to model a variety of processor platforms in software (Burger et al. 1997). This allows the processor architecture to be tailored via input parameters like timing models of different micro-architectural features present in modern processors. In particular, it models in-order and out-of-order pipelines, instruction caches, dynamic branch prediction and their interactions, in such a way that the tool is able to define time bounds for each basic block execution under certain execution contexts. Hence, CHRONOS allows simulating the impact of changing each of these features over the WCET of known embedded software.

CHRONOS uses a series of step that go from C source file compiled with a dedicated GCC in order to build the program control flow graph and detect flow and user constraints. Afterwards, based on the configured processor model, the micro-architectural model is built (at this point this paper will focus the experiments described in the next section). Next, through Integer Linear Programming solver, the estimated WCET is calculated, however considering the micro-architectural modeling. Finally, an observed WCET can also be calculated using SimpleScalar toolset. The observed WCET is guaranteed to be lower than estimated WCET (Xianfeng et al. 2007). The summarized process is depicted in the flowchart below (Xianfeng et al. 2007).

Using a set of benchmark applications that are distributed with the tool, the relevant parameters linked with intra-processor shared resources were modified and the impact of each one of them over WCET was assessed. Shared caches and buses are potential important players in this analysis (Chattopadhyay et al. 2010) and were considered in the experiments described later in this work.

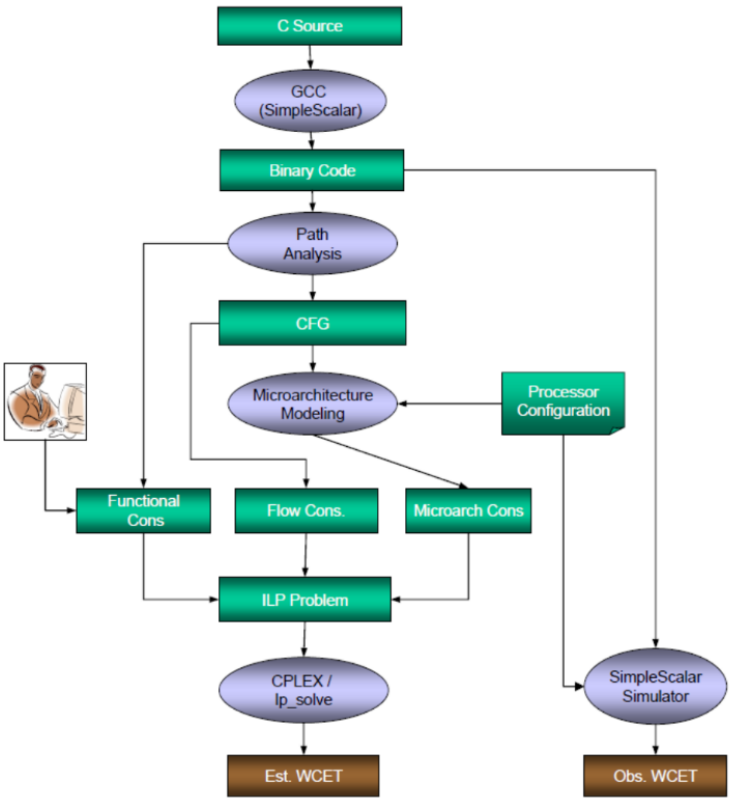

Figure 2: Chronos computation analysis flowchart

RESULTS AND DISCUSSION

In order to demonstrate the impact of architectural resources in the WCET calculation, this paper prepared an analysis focusing on executing WCET benchmarks in different processor model. Using a standardized set of benchmarks provided by Chronos distribution, processor configuration scenarios were created combining the task allocation, different L2 cache sizes and cache architectures to demonstrate the influence of those parameters on execution time.

Experiments

The cache analysis was divided in two experiments according to the architecture of L2 cache: private or shared. In architectures in which the L2 cache is private, each core accesses an exclusive cache, without interference of other cores. Opposing that case, shared architectures allow sharing of L2 cache between two or more cores.

The experiments were designed using processor architecture with two cores, fixed size of L1 cache and analyzed using the same set of scenarios for both experiments. Figure 3 illustrates the experiments design.

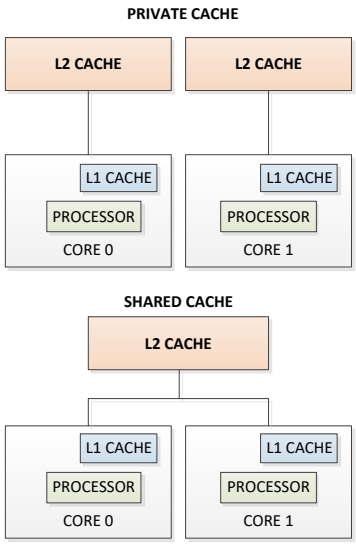

Figure 3: L2 cache architecture

As mentioned above, several scenarios were developed to stress the benchmarks, varying the modelling of processor. It is made changing parameters used as input for Chronos Tool during the microarchitecture modeling phase. In the next paragraphs, there is a description of the purpose of each benchmark and what is the similarity and differences among the processor configurations.

Processor Configuration Scenarios

As described in the previous section, the Chronos Tool Analyzer allows the processor's modelling varying parameters in the context of single or multicore microarchitecture simulation. In this paper all scenarios were developed using 2 cores, with one benchmark running in each core. Since the focus of the paper is the cache memory analysis, all parameters related with internal processing and buses were maintained the same for all scenarios, in order to avoid interference of other components of the process.

In the experiments, it was decided to fix some parameters, like L1 cache size. The size chosen to L1 cache was 128 bytes, since increasing its value would lead to an impaired stress of L2 cache resource.

Therefore, the most significant modification in the processor modelling was the size of L2 cache. The scenarios were created increasing the size in a range from 512 Bytes to 64 Kbytes. Additionally, the set of tests were duplicated changing the cache architecture between private and shared. Thus, the scenarios summarized in the Table 4, were developed for both experiments.

Table 4. Processor configuration scenarios

Scenario	L1 Size	L2 Size
1	128 B	512 B
2	128 B	1 KB
3	128 B	2 KB
4	128 B	4 KB
5	128 B	8 KB
6	128 B	16 KB
7	128 B	32 KB
8	128 B	64 KB

Benchmarks

Each experiment analysis was made grouping the benchmarks in pairs, one in each core, and executed using Chronos Analyzer for each configuration processor scenario. This allocation can be observed in the Figure 4.

The benchmarks were developed following particular restrictions to simplify the static analysis and make possible the WCET calculation. All implementations are completely structured (no unconditional jumps, exit from loops), with no 'switch' and 'do/while' statements. Additionally, operators like 'and/or' for multiple expressions are not used, as well as no library calls.

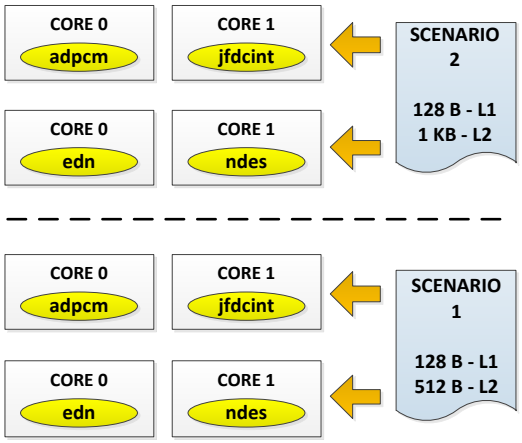

Figure 4: Processor Configuration Scenarios

Furthermore, the benchmarks listed in Table 5 were selected because they present a heavy memory allocation, since all of them perform operations involving vector, arrays and calculations over these elements.

Table 5. Benchmarks

Benchmarks	Meaning	Description
edn	Vector Multiplication	Compilation of several cases which implements vector multiplications and array handling.
jfdcint	JPEG slow-but-accurate integer implementation of the forward Discrete Cosine Transform	Long calculation sequences (i.e., long basic blocks), single-nested loops.
adpcm	Adaptive Differential Pulse Code Modulation algorithm	16Khz sample rate data is used as input data and after calculation the result and compressed array are generated.
ndes	Complex embedded code.	A lot of bit manipulation, shifts, array and matrix calculations.

Results and Discussion

The results analysis presented below, in form of graphs, summarizes all WCET measurements extracted from the experiments execution. The graphs represent the comparison between the benchmark execution using private and shared L2 cache architecture.

There is more information that could be extracted from the graphs, like what is the best WCET of a given benchmark in such scenario. Another possible information that can be extract is how much L2 memory is necessary to achieve the best WCET. All those points will be analyzed in the next paragraphs.

As a result of the experiments addressed by the bench-marks described in Table 5, and executing all scenarios of Table 4, the graphs on Figures 5 – 8 were obtained. Solid

lines in the graphs represent private L2 caches while dashed lines represent shared L2 cache.

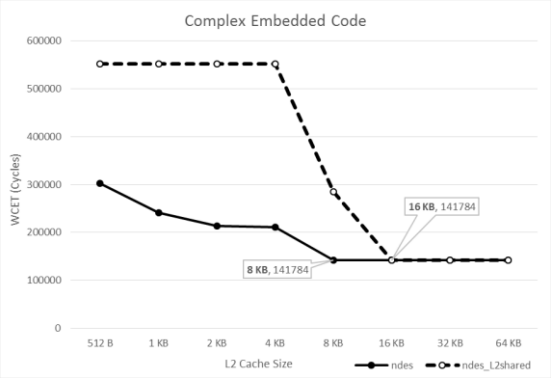

Figure 5: ndes benchmark comparison

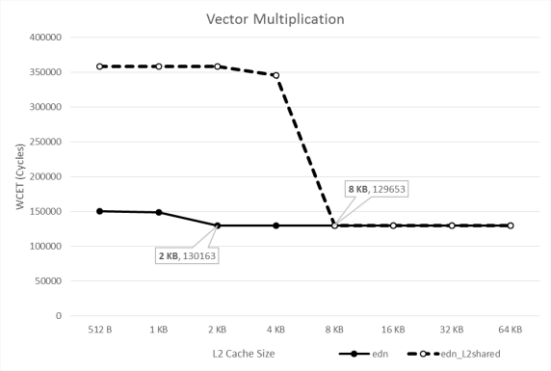

Figure 6: edn benchmark comparison

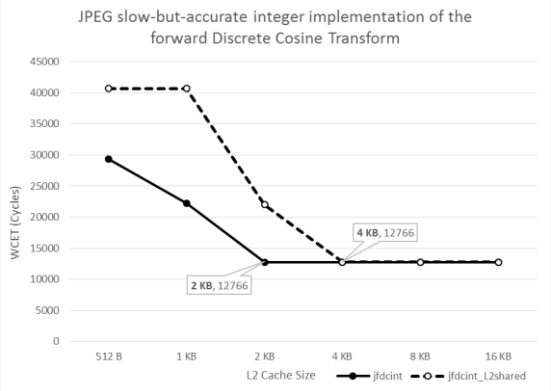

Figure 7: jfdcint benchmark comparison

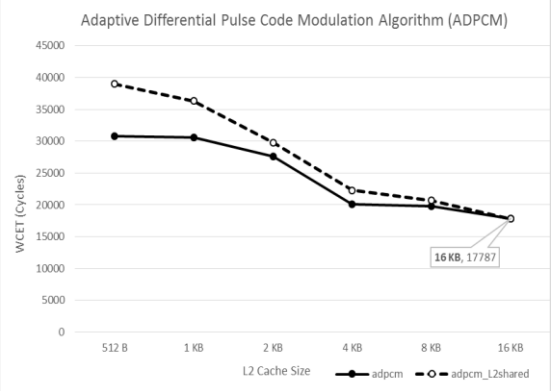

Figure 8: adpcm benchmark comparison

Analyzing the presented WCET graphs, it is possible to verify that if the modeled processor does not have enough L2 cache to allocate all necessary memory for the benchmark under execution, the WCET presents a number larger than its best estimation. The estimated WCET decreased when the L2 cache increases its size, approaching its best estimation. This can be observed by the difference between the solid and dashed lines of the Figures 5 – 8.

Despite the similar behavior presented by the four tested benchmarks, it is possible to observe in the figures that due to particularities of the computation performed by each one of them, the approximation to the private L2 cache results present different shapes. In Figure 5 and Figure 6 it is possible to observe sharper drops, while in Figure 7 and Figure 8 the drops are softened. This is due to the implementation and purpose of each benchmark. The sharper drops are consequence of a large amount of memory used by both benchmarks. It increases the number of cache miss when the size of L2 cache is not enough to store the necessary data. By the other side, in Figure 7 and Figure 8, the drops are softened because the benchmarks are implemented focusing in performance. Several calculations and operations are performed under the data allocated in the

memory, resulting in a special behavior in the graphs. The solid and dashed curves are always following each other, evidencing for those cases that the bottleneck is not the cache size but the computation power.

Additionally, all benchmarks WCET were larger when L2 cache is shared. A major consequence of this finding is that in order to achieve the best WCET estimation it would be necessary to increase L2 cache memory size. However, since COTS processors have already pre-defined cache sizes, probably this WCET best estimation number will not be reached. In this experiment, all benchmarks reach the best estimation, either in private or shared cache, only when L2 cache was significantly increased. This was more remarkable in the results shown in Figure 5, in which the results of the shared L2 cache achieved the same results as the private ones only when the 16 KB size was reached.

CONCLUSIONS

The experiments using a simulated dual core processor modeled on CHRONOS tool with shared and private L2 cache, running pre-defined standardized benchmarks showed that indeed, shared resources between cores pose a significant impact over WCET analysis. This analysis was described in the related works (Nowotsch et al. 2014). The consequence of that is the impact in system determinism, bringing major challenges on avionics certification for multicore systems.

WCET estimation for multicore platforms is still an open topic for avionics industry (Lofwenmark et al. 2014). However, given that the usage of multicore processors is a point of no return, the problem still needs to be tackled. The contribution of this paper towards this goal is to demonstrate the influence of L2 cache architecture in the WCET estimation, highlighting the difficult to achieve the best WCET when the resource is shared. The experimental results evidenced that increasing the L2 cache size allows to reach better WCETs in tasks where there is a heavy memory access, increasing system performance, especially in tasks with vector and matrix manipulations.

As future work it is possible to wide up the analysis on simulated environment considering processors with more than 2 cores and also using more complex software as input, thus verifying the impact of other shared resources. As a step ahead on the same subject, it is intended to perform resources monitoring and WCET calculation on actual real multicore hardware, aiming to minimize the impact of shared resources verified on simulations performed previously.

ACKNOWLEGMENT

The authors thank to the Swedish-Brazilian Research and Innovation Centre – CISB for the provided support to develop this research.

REFERENCES

RTCA, Inc. 2005. "RTCA/DO-297, integrated modular avionics (IMA) development, guidance and certification considerations",.

Lofwenmark, A. & Nadjm-Tehrani S. 2014 "Challenges in Future Avionic Systems on Multi-core Platforms", *2014 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp 115-119, (Conference proceedings)

Weilong, R. & Zhengjun Z. 2014 "Kernel-level Design to Support Partitioning and Hierarchical Real-time Scheduling of ARINC 653 for VxWorks". *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*, pp 388-393.

ARINC 2006. *ARINC Specification 653: Part 1, Avionics Application Software Standard Interface, Required Services* (March 7, 2006) available from ARINC, 2551 Riva Road, Annapolis, MD 21401 Available at: https://www.arinc.com/cf/store/index.cfm Accessed in: 2015-11-20.

Luque, C., Moreto M., Cazorla, F. J., Gioiosa, R., Buyuktosunoglu A. & Valero M. 2012 "CPU Accounting for Multicore Processors", *IEEE Transactions on Computers, vol. 61, no. 2*.

Otoom, M. & Paul, J. M. 2015 "Multiprocessor Capacity Metric and Analysis". *IEEE Transactions on Computers, vol. 64, no. 11*.

RTCA 1992, "DO-178B/ED-12B - software considerations in airborne systems and equipment certification".

Federal Aviation Administration 2012. "*FAA Order 8040.4A Safety Risk Management Policy Document Information*". https://www.faa.gov/regulations_policies/orders_notices/index.cfm/go/document.current/documentNumber/8040.4 .

Pop, P., Tsiopoulos, L., Voss, S., Slotosch, O., Ficek, C., Nyman, U. M. & Lopez, A. R. 2013 "Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms". *WICERT 2013 Conference Proceedings*.

RTCA, "RTCA/DO-178C, software considerations in airborne systems and equipment certification", 2012.

Mitra, T. & Roychoudhury, A. 2007 "Worst Case Execution Time and Energy Analysis", in: Y. Srikant, P. Shankar (eds.), *The Compiler Design Handbook*, CRC Press.

Mushtaq H., Al-Ars, Z. & Bertels, K. 2013 "Accurate and Efficient Identification of Worst-Case Execution Time for Multicore Processors: A Survey", *2013 8th International Design and Test Symposium (IDT)*, pp 1-6.

Wilhelm, R. 2008 "The worst-case execution-time problem overview of methods and survey of tools". *ACM Transactions on Embedded Computing Systems (TECS)*, Volume 7 Issue 3, Article No. 36, April 2008.

Nowotsch, J., Paulitsch, M., Henrichseny, A., Pongratzy, W. & Schachty A. 2014 "Monitoring and WCET Analysis in COTS Multi-core-SoC-based Mixed-Criticality Systems", *2014 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp 1-6, 2014.

Betti, E., Bovet, D. P., Cesati, M. & Gioiosa R. 2008 "Hard Real-Time Performances in Multiprocessor-Embedded Systems Using ASMP-Linux". *EURASIP Journal on Embedded Systems Volume 2008*, Article ID 582648, 16 pages doi:10.1155/2008/582648.

Bastoni, A., Brandenburg, B. B. & Anderson, J. H. 2010 "An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers". *31st IEEE Real-Time Systems Symposium*, pp 14-24.

Nowotsch, J., Paulitsch, M., Buhler, D., Theiling, H., Wegener, S. & Schmidt, M. 2014 "Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement". *26th Euromicro Conference on Real-Time Systems*, pp 109-118, 2014.

Saranya, N. & Hansdah, R. C. 2014 "An Implementation of Partitioned Scheduling Scheme for Hard Real-Time Tasks in Multicore Linux with Fair Share for Linux Tasks", *IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp 1-9, 2014.

Xianfeng, L., Yun L., Mitra, T. & Roychoudhury, A. 2007 "Chronos: a Timing Analyzer for Embedded Software", *Science of Computer Programming*, Volume 69, Issues 1–3, 1 December 2007, pp 56–67.

Burger, D. & Austin, T. 1997 "*The SimpleScalar Tool Set, Version 2.0, Technical Report CS-TR-1997-1342*", Report of University of Wisconsin, Madison, http://www.simplescalar.com/docs/users_guide_v2.pdf.

Chattopadhyay S., Roychoudhury A. & Mitra, T. 2010 "Modeling Shared Cache and Bus in Multi-cores for Timing Analysis", Proceeding *SCOPES '10 Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems*, Article No. 6, 2010.