T-CREST: A TIME-PREDICTABLE MULTI-CORE PLATFORM FOR AEROSPACE APPLICATIONS

Martin Schoeberl[1], **Cláudio Silva**[2], **and André Rocha**[2]

[1]*Department of Applied Mathematics and Computer Science, Technical University of Denmark*
[2]*GMV, Lisbon, Portugal*

ABSTRACT

Space systems are hard real-time systems, where the worst-case execution time (WCET) of tasks needs to be known to prove absence of deadline misses. For simple processor and memory architectures it is possible to statically derive a safe upper bound of the WCET. However, future requirements in more autonomous missions require more processing power. This increase in processing power is approached by multi-core processors. However, current multi-core processors are not WCET analyzable.

The mission of T-CREST is to develop tools and build a multi-core system that provides high performance, but be WCET analyzable. The T-CREST time-predictable system will simplify the safety argument with respect to the maximum execution time and increase the performance with multi-core technology. Thus the T-CREST system will result in lower costs for safety-relevant applications, reducing system complexity, simultaneously providing faster time-predictable execution. Most of the T-CREST technology is available in open-source.

Key words: Real-Time Systems; CPU Architecture; Worst-Case Execution Time.

1. INTRODUCTION

Multi-core technology emerges as a logical next step to handle the growing requirements for increased performance and increased integration in embedded systems, while at the same time decreasing the power consumption, weight and volume budgets. The space domain can benefit from this increased integration between systems as it becomes feasible to merge different functional chains under the same on-board computer (i.e., data handling, altitude control and payload). Additionally, performance-demanding applications are increasing in spacecraft systems. Space missions are becoming more autonomous, therefore several tasks that used to be delegated to ground-based systems have now to be tackled on-board. Examples of missions requiring higher performance are those which make use of advanced guidance and navigation algorithms. Often, these are based on real-time image processing or model estimation techniques which are very CPU demanding [1].

Moreover, the technology and architectures used to enable fault tolerance and the on-board power availability confine spacecraft on-board computers to low performances (i.e., low clock frequencies and number of instructions per second). Even with state-of-the-art production technologies, the SPARC family of space processors is limited to frequencies in the order of the few hundred MHz [2]. This fact conflicts with the requirements for growing performance and integration and hence will start to constrain the evolution of the space platform. Multi-core architectures appear as a natural solution for this issue [3].

Space systems can be characterized as safety-critical hard real-time systems. In this class of systems, the worst-case execution time (WCET) of the software needs to be bounded and known so that timely delivery of critical responses is guaranteed. This requirement of hard real-time systems becomes problematic when they are deployed on multi-core architectures, mainly because multi-core technologies tend to make it difficult or even impossible to obtain the maximum execution time [4].

This paper presents an innovative multi-processor system-on-chip (SoC) architecture, the T-CREST platform,[1] designed since its inception to be compatible with the predictability requirements of hard real-time systems. The T-CREST time-predictable platform intends to simplify the safety argument with respect to the maximum execution time, and strive to double performance for 4 cores and to be 4 times faster for 16 cores than a standard processor in the same technology (e.g., FPGA).

This paper is organized as follows: in Section 2, we describe the challenges faced by the adoption of multi-core in safety critical systems and in Section 3 the current efforts undertaken to tackle this issue. Section 4 introduces the T-CREST platform. In Section 5, preliminary results are discussed. Finally, Section 6 concludes this paper.

[1] `http://www.t-crest.org/`

2. MULTI-CORE AND SAFETY CRITICAL: A NECESSARY EVIL

Several domains, including space, have already started to address the deployment of multi-core architectures in their systems. The main motivation behind this migration is the ability to increase system integration and performance while at the same time decreasing the power consumption, weight, and volume. A single multi-core processor can handle the software that is currently distributed through multiple modules, hence decreasing the necessary mass and power. Moreover, multi-core processors, due to their raw performance, can unlock several CPU demanding applications, such as image processing or dynamic system reconfiguration, that would be onerous to deploy using today's systems. Even without this rationale, the adoption of multi-core in those domains will be unavoidable, in the long term, as embedded processors hit the same barriers that motivated the desktop processor market to migrate to multi-core (i.e., the power wall and the memory wall). This necessity is exacerbated in space-like domains where the pressure on mass and power budgets is more stringent. As market presses for more functionality and lower costs, multi-core processors will quickly become a requirement.

However, before multi-core processors can be considered a solution for growing market demands and natural system complexity evolution, several problems must be tackled. Safety-critical systems have certain characteristics, like predictability, that can be affected by the deployment in multi-core systems. When developing a safety critical hard real-time application, the developer must be able to determine with accuracy and confidence the WCET of its application. This is necessary because not only the result of a computation matters but also the time at which the result is obtained. For these applications, the verification and validation process is required to include an in-depth characterization of the application timings and a schedulability analysis. Each task in the system needs to be attributed with a time budget that is adequate for its WCET. The WCET is the maximum time a task could take to complete execution on a given hardware configuration.

Commercial general-purpose single-core processors already include features like branch prediction and speculative out-of-order execution that make the process of obtaining a WCET bound difficult. This effect is worsened by the new interferences in a multi-core processor. The main source of interference stems from deep within the architecture of the processor; although each core in the system can execute instructions separately, there are shared physical resources to which access is serialized. The most common shared resources between cores are parts of the memory hierarchy and I/O devices [5].

The memory hierarchy plays a critical role, as it is the most important channel of interference and contention. Even in single-core systems it is the main bottleneck for performance; it is easy for a single-core processor to overwhelm main memory with requests [5]. Given that, in multi-core processors, parts of the memory hierarchy are shared between cores, memory accesses to the shared resources need to be arbitrated. Consequently, the existence or non-existence of memory accesses from the other cores may influence the latency of a memory access of a given task. This influence, or interference, can possibly emerge in every shared resource, including any element of the memory hierarchy (e.g., bus, caches, etc.).

In contrast with single-core, the WCET for a given task in a multi-core system will additionally depend on the profile of memory accesses that other tasks, concurrently executing in other cores, have. Since the execution path of simultaneously executing tasks will vary, so will the execution time. This causes a significant increase in the complexity of WCET estimation, as it is unfeasible to consider every possible interference pattern between the different cores.

The lack of predictability resulting from this interference can only be alleviated by penalizing WCET estimations. To have a safety margin in the WCET bound, the worst possible interference pattern for that processor and memory hierarchy must be considered while computing the WCET for a given task. This typically results in a WCET estimation that is several orders of magnitude greater in a multi-core system than it would be in a single-core. This pessimism in WCET estimation quickly defeats the purpose of multi-core by creating systems with a worse performance than on single-core [6].

3. CURRENT STATUS OF MULTI-CORE IN THE SPACE DOMAIN

The space domain is currently starting with the first steps into the adoption of multi-core technologies. Both ESA and NASA have research and development projects in this area and have developed multi-core processors according to their own requirements [7] [8].

One of these processors is the Next Generation Microprocessor (NGMP) [7]. The NGMP is a SPARC V8(E) quad-core architecture centered around the LEON4FT processor, currently being developed by Aeroflex Gaisler in conjunction with the European Space Agency. It aims to provide significant performance increase compared to earlier generations of European space processors, and aims to be used in the future space missions of the Agency.

As part of the multicore OS benchmark study funded by ESA, the NGMP processor was evaluated with respect to predictability and timing behavior. The study concluded that for memory intensive tasks, with a lot of store instructions, the execution time can be up to 20 times higher than the execution time on a single-core processor [9]. This slowdown verified in the NGMP processor can be considered typical for processors that are optimized for average-case performance. For this kind of processor be-

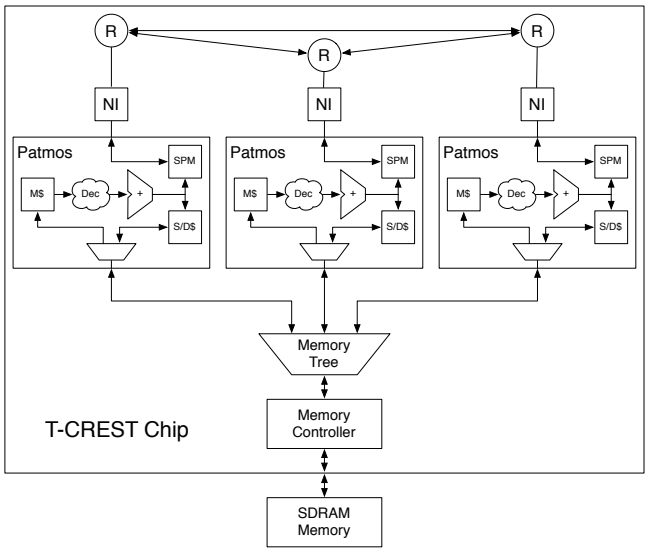

Figure 1. The T-CREST platform consisting of processor nodes that are connected via a network-on-chip for message passing and a memory tree for shared memory access.

havior, there is currently no tool or method support to estimate the WCET [10]. To address this issue, ESA has recently funded a project to research probabilistic WCET estimation techniques [11].

From several research and development activities trying to address the issues of safety-critical multi-core systems, probabilistic WCET (pWCET) appears as a promising trend which has yielded some interesting results as part of the FP7 STREP PROARTIS project [12]. PROARTIS approaches the issue of WCET by removing the dependencies between timing behavior and execution history, and then applying probabilistic analysis techniques [12]. ESA selected this research thread because it is one of the few that can be applied to existing multi-core processors, like the NGMP, with little or no modifications on the hardware side. Additionally, pWCET tools pose few requirements to the user to use them and to the development process [11]. However, there are still several challenges to the technology behind pWCET estimation, partly because it requires the timing behavior of software to be independent from the execution history, which can prove to be a difficult task.

In contrast to the approach of trying to fix the problem of hardly predictable hardware with a probabilistic WCET analysis, we design the hardware from ground up to be time-predictable. T-CREST is a time-predictable multi-core architecture with static WCET analysis support.

4. THE T-CREST PLATFORM

To address the previously described problems with multi-core technology in hard real-time space applications, we propose an innovative approach by designing computer architectures where predictable timing is a first-order design factor [13]. For real-time systems we thus propose to design architectures with a new paradigm [14]:

> Make the worst-case fast and the whole system easy to analyze.

Within T-CREST we propose novel solutions for time-predictable multi-core and many-core system architectures. The resulting time-predictable resources (processor, interconnect, memories, etc.) are intended as a good target for WCET analysis, and the WCET performance is expected to be outstanding compared to current processors. Time-predictable caching and time-predictable chip-multiprocessing will provide a solution for the need of increased processing power in the real-time domain.

T-CREST covers technologies from the chip level (processor, memory, asynchronous network-on-chip), via compiler, single-path code generation, and WCET analysis tools, up to system evaluation with a port of the RTEMS operating system and two industry use cases, one from the avionics domain and one from the railway domain.

Figure 1 shows the T-CREST platform. Several processor cores, called Patmos [15], are connected via a memory tree [16] to a real-time memory controller [17, 18] to the shared, external SDRAM memory. For efficient core-to-core communication each processor is connected to a network-on-chip (NoC). These on-chip communication channels reduce the pressure on the shared memory bandwidth.

4.1. The Processor

The basis of a time-predictable system is a time-predictable processor. Within T-CREST we developed a time-predictable processor, named Patmos [15], as one approach to attack the complexity issue of WCET analysis. Patmos is a statically scheduled, dual-issue RISC processor that is optimized for real-time systems.

A major challenge for the WCET analysis is the memory hierarchy with multiple levels of caches. We attack this issue by using caches that are especially designed for WCET analysis. For instructions we adopt the method cache [19], which operates on whole functions/methods and thus simplifies the modeling for WCET analysis [20]. Furthermore, we propose a split-cache architecture [21, 22] for data, offering dedicated caches for the stack area [23], constants, static data, heap allocated objects, as well as a compiler and program managed scratchpad memory. WCET analysis of the stack cache has been presented in [24].

Patmos contains 5 pipeline stages: (1) instruction fetch (`FE`), (2) decode and register read (`DEC`), (3) execute (`EX`), (4) memory access (`MEM`), and (5) register write back (`WB`). Figure 2 shows an overview of Patmos' pipeline. The pipeline is a dual issue pipeline executing up to two ALU instructions each clock cycle.

Accesses to the different types of data areas are explicitly encoded with the load and store instructions. We call this typed load and store instructions, which direct the loads and stores to the relevant cache. This feature helps the WCET analysis to distinguish between the different data caches.

Patmos also supports predication of all instructions. This feature reduces the number of conditional branches and supports generation of single-path code [25, 26]. The compiler, LLVM, is extended with an optimization path that translates normal code into single-path code [27].

4.2. The Interconnect

In order to build a chip-multiprocessor system out of Patmos processor cores we need a suitable interconnect – a network-on-chip (NoC). The Patmos multi-core processor uses non-coherent large off-chip memory and additional processor local, small memory blocks (scratchpad memories). The NoC supports time-predictable message passing between those local scratchpad memories. The shared off-chip memory is supported by a memory NoC, the memory tree [16].

To enable time-predictable usage of a shared resource the resource arbitration has to be time-predictable. In the case of a NoC, statically scheduled TDM is a time-predictable solution [28, 29]. This static schedule is repeated and the length of the schedule is called the *period*. Like tasks in real-time systems, the communication is also organized in periods. One optimization point of the design is minimizing the period in order to minimize the latency of delivering data and the size of the schedule tables. For regular network architectures and all-to-all communication paths a heuristic is able to find good (almost optimal) solutions for the TDM schedule [30]. The T-CREST NoC uses TDM from end to end, including the network interface. That approach also results in an efficient implementation of the network interface [31].

In the field of embedded systems, multi-processor platforms are typically optimized for a given application or application domain. The NoC communication schedules can be optimized for an application.

Different types of data are transferred on the NoC, e.g., message passing data between cores, cache fills from main memory, synchronization operations such as compare-and-swap. In most architectures, a single NoC serves all those different types of data. However, the requirements of these different data types with respect to e.g., packet size, address ranges, and flow control are different. Therefore, we will further evaluate if several NoCs, for the traffic type optimized, result in a more efficient solution than a single shared NoC.

For the memory access T-CREST provides a TDM based memory NoC [32]. This NoC implements distributed arbitration of the global TDM schedule at the client sides. This architecture also allows pipelining of the merge tree and the return path. That pipeline and the resulting latency is incorporated in the local TDM arbitration.

4.3. Memory Hierarchy

The only memory layer that is under direct control of the compiler is the register file. Other levels of the memory hierarchy are usually not visible – they are not part of the ISA abstraction. The placing of data in the different layers is automatically performed. While caches are managed by the hardware, virtual memory is managed by the operating system. The access time for a word that is in a memory block paged out by the operating system is several orders of magnitude higher than a first level cache hit. Even the difference between a first level cache access and a main memory access is in the order of two magnitudes.

Cache memories for instructions and data are classic examples of the paradigm *Make the common case fast*. Plenty of effort has gone into researching the integration of the instruction cache into the timing analysis [33] and the integration of the cache analysis with the pipeline analysis [34]. The influence of different cache architectures on WCET analysis is described in [35].

Caches in general, and particularly data caches, are usually hard to statically analyze. Therefore, we introduce caches that are organized to speed-up execution time and provide tight WCET bounds. We propose a split cache architecture consisting of: (1) an instruction cache for

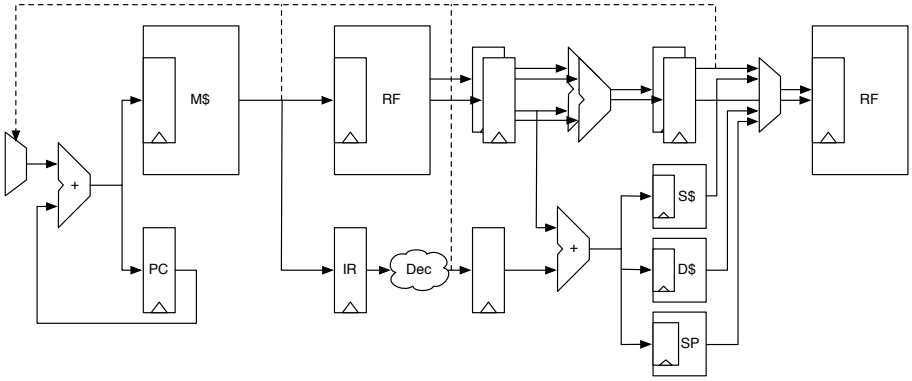

Figure 2. Dual issue pipeline of Patmos with fetch, decode, execute, memory, and write back stages.

full methods, (2) a stack cache, and (3) a cache for static data, constants, and type information. Furthermore, we also support a program- or compiler-managed scratchpad memory for instruction and data storage and inter-processor communication to tighten bounds for hard-to-analyze memory-access patterns.

Even for embedded systems the on-chip available memory is usually too small to hold all code and data. Therefore, off-chip SDRAM serves as shared main memory for the multicore processor. Access time to an SDRAM usually depends on the history of former accesses (e.g., open rows). This optimization improves the average case execution time, but not the WCET. Therefore, within T-CREST memory controllers have been developed that are a better fit for real-time systems [17, 18, 36]

The combination of the TDM based NoC, the memory arbitration tree, and the time-predictable memory controller allows, even on a multi-core system, to provide upper bounds on memory transactions. This upper bound enables WCET analysis of individual tasks executing on a multi-core system.

4.4. Compiler, WCET Analysis, Software Support

The performance of the dual-issue processor depends on statically scheduled instructions. We argue that all architectural features of a processor shall be exposed to the compiler to generate time-predictable code. Within T-CREST the LLVM compiler framework has been adapted to target Patmos. Furthermore, we explore compiler optimizations for the WCET instead of the average case execution time.

The processor is intended as a platform to explore various time-predictable design trade-offs and their interaction with WCET analysis techniques as well as WCET-aware compilation. We propose the co-design of time-predictable processor features with the WCET analysis tool, similar to the work by Huber et al. [37] on caching of heap allocated objects in a Java processor. Only features where we can provide a static program analysis shall be added to the processor.

The WCET analysis tool aiT from AbsInt has been adapted to support the dual-issue processor Patmos. It is also the platform for exploration of time-predictable processor features. The WCET oriented optimizations in the compiler are tightly integrated with the WCET analysis tool [38, 39]. The WCET tool provides information on the worst-case path and basic block timings to guide the optimization process.

Regarding software support, the libraries adapted as part of the compiler port (i.e., newlib) allow for a 'bare-metal' C executive to run on top of Patmos. These libraries were also extended to add support to T-CREST specific features like scratchpad access.

For more complex applications, the RTEMS operating system was ported to the T-CREST platform. RTEMS is a free and open source real-time operating system, used as a baseline for dozens of space missions, that is compatible with open standards such as POSIX or iTRON [40]. The deployment of RTEMS instances in an asymmetric fashion over the T-CREST platform will be experimented as a means to explore multi-core as a hardware partitioning platform inline with the transitioning of the space software reference architecture to Integrated Modular Avionics [40].

5. EVALUATION

Demonstrating the platform's capability to host real applications with delicate predictability requirements provides good evidence for the appropriateness of the platform for its intended purpose in the domain of critical real-time systems. To evaluate the T-CREST platform prototype we make use of industrial use cases derived of real-world applications. These use cases are built upon domain-specific use case applications from the avionics and railways domains.

The avionics use cases consist of a set of avionics applications that are hosted on one computing platform as it is common practice in on-board systems integrated according to the principles of Integrated Modular Avionics

(IMA). The avionics applications considered in the evaluation are the following:

- Airlines Operational Centre (AOC) - The AOC is the on-board part of an Air Traffic Management (ATM) system that enables digital text communication between aircrew and ground ATM units.
- Crew Alerting System (CAS) - The CAS system receives signals from on-board subsystems, such as doors, engines or the environment control system, and displays relevant aircraft information such as engine parameters (e.g., temperature values, fuel flow and quantity).
- I/O Partition (IOP) - The I/O Partition is a dedicated partition in an IMA for Space system that allows all other partitions to access I/O devices.

These applications were originally designed and implemented for specific run-time platforms (e.g. ARINC 653 APEX or RTEMS). To successfully evaluate the platform, these demonstrators must make use of the T-CREST platform specific features such as the inter-core NoC, local memory and WCET-aware compilation. As such, their adaptation to the T-CREST platform requires significant effort, being a rather complex task that involves several (re)design decisions. Our preferred option to address this issue was to envision evolving use cases that are incrementally more complex and optimized for the platform.

All demonstrators were originally IMA applications that communicated with other applications and external systems through queuing and sampling ports. These communication interfaces are usually based on buffers in main memory and, hence, are subject to heavy contention in a multi-core processor. As part of the optimization to the T-CREST platform, the port interfaces used by the demonstrators will be mapped to inter-core communication using the configurable NoC. This optimization removes some burden from the main memory and should reduce the WCET of the demonstrators. The original port specifications in terms of size and number of messages can be used as an input to configure the TDM schedule in the NoC.

Additionally, the demonstrators are all based on real-time operating systems. A possible optimization is to remove the operating system and turn the demonstrators into bare cyclic applications. With this modification, it will be probably possible to fit the applications without OS into the local memory of each core. The applications with OS are forced to remain in main memory which is subject to heavy contention from concurrent accesses. This optimization, if feasible, will result in lower WCET since local memory is not subject to contention. This optimization will be clearer in use cases making use of several cores. Nonetheless, it shall be possible to determine a WCET bound even if the applications are running from main memory.

One of the final objectives of the evaluation will be to demonstrate that, given a configuration of the T-CREST platform, it is possible to independently obtain the WCET of any application. This independence between applications is a cornerstone in the Integrated Modular Avionics concept and it is extremely difficult to obtain in multi-core systems. In order to validate application independence, and given the high numbers of cores available, each core will host a different application that would, in a typical IMA system, be a standalone partition. This distribution of applications through cores in an asymmetric fashion will validate that the timing of each application does not depend on the software executed on other cores.

6. CONCLUSION

Future space applications need more processing power for more autonomous mission control. Multi-core systems can deliver this processing power. However, space applications are also real-time systems where the worst-case execution time of tasks needs to be known. Current multi-processor systems are hardly analyzable.

In this paper we presented T-CREST, a multi-core platform that was designed for real-time systems from ground up. It is worst-case execution time analyzable and therefore a good candidate for answering the requirements of future space missions.

ACKNOWLEDGEMENTS

This work was partially funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

Source Access

The sources of the implementation are available in open source: `https://github.com/t-crest`. Further information can be found at the project web site: `http://www.t-crest.org` and the web site for the processor and the compiler: `http://patmos.compute.dtu.dk/`

REFERENCES

1. Ortega, G. Gnc application cases needing multi-core processors. In *Proceedings of the Workshop in Avionics Data, Control and Software Systems (ADCSS)*. European Space Agency, 2011.
2. Magistrati, G. Multi-core processors for space applications. In *Proceedings of the Workshop in Avionics Data, Control and Software Systems (ADCSS)*. European Space Agency, 2011.

3. Terraillon, J.-L. Multicore in space. Keynote - 17th International Conference on Reliable Software Technologies (Ada-Europe 2012), June 2012.

4. Kinnan, L. M. Use of multicore processors in avionics systems and its potential impact on implementation and certification. In *Proceedings of 28th Digital Avionics Systems Conference (DASC)*, pages 4.1 – 4.6, 2009.

5. Fuchsen, R. How to address certification for multicore based ima platforms: Current status and potential solutions. In *Proceedings of the Digital Avionics Systems Conference (DASC)*. IEEE, 2010.

6. Nowotsch, J. and Paulitsch, M. Leveraging multicore computing architectures in avionics. In *Proceedings of the European Workshop on Dependable Computing*, 2012.

7. Andersson, J., Hjorth, M., Habinc, S., and Gaisler, J. Development of a functional prototype of the quad core NGMP space processor. In *Proceedings of Data Systems In Aerospace Conference*, 2012.

8. Cho, S. and Demetriades, S. Maestro: Orchestrating predictive resource management in future multicore system. In *NASA/ESA Conference on Adaptive Hardware Systems*, 2001.

9. Cazorla, F. J., Gioiosa, R., Fernandez, M., and nones, E. Q. Multicore os benchmark - final report.

10. Patte, M. and Lefftz, V. System impact of distributed multi core systems. Final Report. ESTEC Contract 4200023100.

11. Agency, E. S. Schedulability analysis techniques and tools for cached and multicore processors. Statement of Work, ESA ITT 1-7646/13/NL/JK, July 2013.

12. Cazorla, F. J., nones, E. Q., Vardanega, T., Cucu, L., Triquet, B., Bernat, G., Berger, E., Abella, J., Wartel, F., Houston, M., Santinelli, L., Kosmidis1, L., Lo, C., and Maxim, D. Proartis: Probabilistically analysable real-time systems. In *ACM Transactions on Embedded Computing Systems*, 2012.

13. Schoeberl, M. Is time predictability quantifiable? In *International Conference on Embedded Computer Systems (SAMOS 2012)*, Samos, Greece, July 2012. IEEE.

14. Schoeberl, M. Time-predictable computer architecture. *EURASIP Journal on Embedded Systems*, vol. 2009, Article ID 758480:17 pages, 2009.

15. Schoeberl, M., Schleuniger, P., Puffitsch, W., Brandner, F., Probst, C. W., Karlsson, S., and Thorn, T. Towards a time-predictable dual-issue microprocessor: The Patmos approach. In *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, pages 11–20, Grenoble, France, March 2011.

16. Garside, J. and Audsley, N. C. Investigating shared memory tree prefetching within multimedia noc architectures. In *Memory Architecture and Organisation Workshop*, 2013.

17. Akesson, B., Goossens, K., and Ringhofer, M. Predator: a predictable sdram memory controller. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 251–256, New York, NY, USA, 2007. ACM.

18. Lakis, E. and Schoeberl, M. An SDRAM controller for real-time systems. In *Proceedings of the 9th Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2013.

19. Schoeberl, M. A time predictable instruction cache for a Java processor. In *On the Move to Meaningful Internet Systems 2004: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2004)*, volume 3292 of *LNCS*, pages 371–382, Agia Napa, Cyprus, October 2004. Springer.

20. Degasperi, P., Hepp, S., Puffitsch, W., and Schoeberl, M. A method cache for Patmos. In *Proceedings of the 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2014)*, Reno, Nevada, USA, June 2014. IEEE.

21. Schoeberl, M. Time-predictable cache organization. In *Proceedings of the First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009)*, pages 11–16, Tokyo, Japan, March 2009. IEEE Computer Society.

22. Schoeberl, M., Huber, B., and Puffitsch, W. Data cache organization for accurate timing analysis. *Real-Time Systems*, 49(1):1–28, 2013.

23. Abbaspour, S., Brandner, F., and Schoeberl, M. A time-predictable stack cache. In *Proceedings of the 9th Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2013.

24. Jordan, A., Brandner, F., and Schoeberl, M. Static analysis of worst-case stack cache behavior. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS 2013)*, pages 55–64, New York, NY, USA, 2013. ACM.

25. Puschner, P. and Burns, A. Writing temporally predictable code. In *Proceedings of the The Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, pages 85–94, Washington, DC, USA, 2002. IEEE Computer Society.

26. Puschner, P. Transforming execution-time boundable code into temporally predictable code. In Kleinjohann, B., Kim, K. K., Kleinjohann, L., and Rettberg, A., editors, *Design and Analysis of Distributed*

Embedded Systems, pages 163–172. Kluwer Academic Publishers, 2002. IFIP 17th World Computer Congress - TC10 Stream on Distributed and Parallel Embedded Systems (DIPES 2002).

27. Puschner, P., Kirner, R., Huber, B., and Prokesch, D. Compiling for time predictability. In Ortmeier, F. and Daniel, P., editors, *Computer Safety, Reliability, and Security*, volume 7613 of *Lecture Notes in Computer Science*, pages 382–391. Springer Berlin / Heidelberg, 2012.

28. Schoeberl, M., Brandner, F., Sparsø, J., and Kasapaki, E. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proceedings of the 6th International Symposium on Networks-on-Chip (NOCS)*, pages 152–160, Lyngby, Denmark, May 2012. IEEE.

29. Sørensen, R. B., Schoeberl, M., and Sparsø, J. A light-weight statically scheduled network-on-chip. In *Proceedings of the 29th Norchip Conference*, Copenhagen, November 2012. IEEE.

30. Brandner, F. and Schoeberl, M. Static routing in symmetric real-time network-on-chips. In *Proceedings of the 20th International Conference on Real-Time and Network Systems (RTNS 2012)*, pages 61–70, Pont a Mousson, France, November 2012.

31. Sparsø, J., Kasapaki, E., and Schoeberl, M. An area-efficient network interface for a TDM-based network-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1044–1047, San Jose, CA, USA, 2013. EDA Consortium.

32. Schoeberl, M., Chong, D. V., Puffitsch, W., and Sparsø, J. A time-predictable memory network-on-chip. In *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, 2014.

33. Arnold, R., Mueller, F., Whalley, D., and Harmon, M. Bounding worst-case instruction cache performance. In *IEEE Real-Time Systems Symposium*, pages 172–181, 1994.

34. Healy, C. A., Arnold, R. D., Mueller, F., Whalley, D. B., and Harmon, M. G. Bounding pipeline and instruction cache performance. *IEEE Trans. Computers*, 48(1):53–70, 1999.

35. Heckmann, R., Langenbach, M., Thesing, S., and Wilhelm, R. The influence of processor architecture on the design and results of WCET tools. *Proceedings of the IEEE*, 91(7):1038–1054, Jul. 2003.

36. Gomony, M. D., Akesson, B., and Goossens, K. Architecture and optimal configuration of a real-time multi-channel memory controller. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1307–1312, 2013.

37. Huber, B., Puffitsch, W., and Schoeberl, M. WCET driven design space exploration of an object cache. In *Proceedings of the 8th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2010)*, pages 26–35, New York, NY, USA, 2010. ACM.

38. Puschner, P., Prokesch, D., Huber, B., Knoop, J., Hepp, S., and Gebhard, G. The T-CREST approach of compiler and WCET-analysis integration. In *9th Workshop on Software Technologies for Future Embedded and Ubiquitious Systems (SEUS 2013)*, pages 33–40, 2013.

39. Huber, B., Prokesch, D., and Puschner, P. Combined WCET analysis of bitcode and machine code using control-flow relation graphs. In *Proceedings of the 14th ACM SIGPLAN/SIGBED conference on Languages, compilers and tools for embedded systems*, pages 163–172. The Association for Computing Machinery, 2013. talk: Conference on Languages, Compilers and Tools for Embedded Systems (LCTES 2013), Seattle, WA, USA; 2013-06-20 – 2013-06-21.

40. Silva, C. Integrated modular avionics for space applications: Input/output module. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, 2012.