Design of a time-predictable multicore processor: the T-CREST project

Schoeberl, Martin

Published in:
Proceedings of 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)

Link to article, DOI:
10.23919/DATE.2018.8342138

Publication date:
2018

Document Version
Peer reviewed version

Design of a Time-predictable Multicore Processor: The T-CREST Project

Martin Schoeberl

Department of Applied Mathematics and Computer Science
Technical University of Denmark
Email: masca@dtu.dk

***Abstract*—Real-time systems need to deliver results in time and often this timely production of a result needs to be guaranteed. Static timing analysis can be used to bound the worst-case execution time of tasks. However, this timing analysis is only possible if the processor architecture is analysis friendly.**

This paper presents the T-CREST processor, a real-time multicore processor developed to be time-predictable and an easy target for static worst-case execution time analysis. We present how to achieve time-predictability at all levels of the architecture, from the processor pipeline, via a network-on-chip, up to the memory controller. The main architectural feature to provide time predictability is to use static arbitration of shared resources in a time-division multiplexing way.

I. Introduction

This paper presents a technique for achieving time predictability with a multicore processor architecture. The main design idea is to *optimize* all hardware and software components *for a short worst-case execution time* (WCET) instead of improving average-case performance [1]. We achieve this time predictability by consistently using static arbitration for shared resources. In most cases, we use time-division multiplexing with a pre-computed schedule.

The multicore processor presented here was developed within a 3-year STREP project funded under the European Union's 7th Framework Programme under grant agreement no. 288008: Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST).

T-CREST is a multicore architecture that consists of several time-predictable processor cores, called Patmos. Those processor cores are connected via a network-on-chip (NoC) to support time-predictable message passing between core local memories. For larger data structures and programs T-CREST connects to an external memory via a memory tree and a real-time memory controller. On the software side, a compiler has been adapted to support Patmos and perform WCET oriented optimizations. WCET analysis is supported by the aiT WCET analyzer.

T-CREST has now been in use for research and teaching for several years. As a platform, it offers: (1) a ready to use processor for real-time systems, (2) a platform for research on time-predictable multicore architectures, and (3) a platform for teaching time-predictable computer architecture.

T-CREST has been further developed with several research projects. Within the RTEMP project, funded by The Danish Council for Independent Research – Technology and Production Sciences (FTP), we explored, together with Danfoss Power Electronics, the usage of T-CREST for real-time motor control. Within the FTP funded project PREDICT (Time-predictable Control Systems) we are bringing T-CREST into the air by controlling drones. This project is in cooperation with the Drone Research Laboratory at Aalborg University. T-CREST will provide the platform of an open-source Fog computing node for the FORA (Fog Computing for Robotics and Industrial Automation) European Training Network.

An overview of T-CREST has been presented in [2]. In [3] we have discussed some lessons learned within the project related to open-source development and how to organize the final integration work for hardware developed at different places. The contribution of this paper is a detailed description of how to support time-predictable execution of real-time tasks in a multicore processor.

The paper is organized in 5 sections: The following section presents related work. Section III gives an overview of the T-CREST platform. Section IV presents the design decisions for a time-predictable multicore architecture. Section V concludes.

II. Related Work

In the FP-7 project MERASA (Multi-Core Execution of Hard Real-Time Applications Supporting Analysability) [4], the real-time processor CarCore was developed. CarCore is a simultaneous multi-threading version of the TriCore processor. We share the same vision as the MERASA project: building hardware to support real-time systems and WCET analysis. In contrast to the MERASA processor core, Patmos focuses on single-thread real-time performance. To benefit from thread-level parallelism, we replicate the simple pipeline to build a multicore system.

The CoMPSoC platform [5] removes all application interference by resource reservation. CoMPSoC combines the AEtherreal style NoC with customized processor cores from Silicon Hive and a composable memory controller. In contrast to the T-CREST platform, no caches are supported and all code needs to fit into on-chip memory. The memory controller developed during the T-CREST project is now in use in the CoMPSoC project.

Paukovits and Kopetz use a time-triggered NoC for the time-triggered system-on-chip (TTSoC) architecture [6]. The main

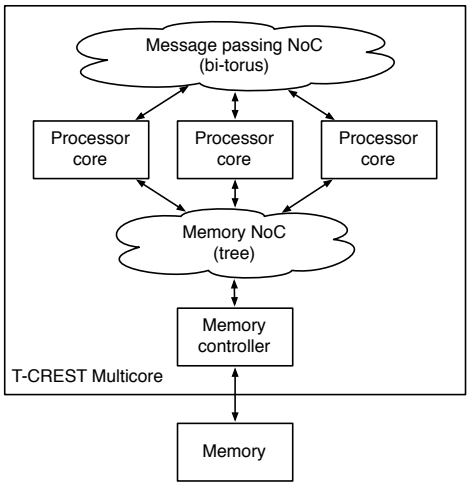

Fig. 1. The T-CREST multicore architecture with several processor cores connected to two NoCs: one for core-to-core message passing and one for access to the shared, external memory

difference to other NoC designs is use of time is derived from world time (via GPS) that forms the basis for the TDM slotting. The TTSoC architecture has been used in the GENESYS multiprocessor system-on-a-chip [7]. In contrast to the TTSoC our TDM schedule uses a time base established by mesochronous clocking of the network interfaces. Therefore, we can schedule communication at clock cycle granularity, even without the need for a global system clock.

III. The T-CREST Platform

The T-CREST Platform consists of Patmos processor cores, two on-chip networks, a memory controller, a compiler, a port of the C standard library, and two WCET analysis tools.

A. The Hardware

T-CREST is a multicore processor where the individual processor cores are connected by two NoCs: (1) a message passing NoC for communication between individual cores and (2) a memory NoC for communication between the cores and a memory controller for shared, external main memory.

Fig. 1 shows the T-CREST multicore processor [2]. Each processor core is a Patmos processor [8], which is a RISC pipeline optimized for low WCET bounds. Patmos contains a statically scheduled dual-issue pipeline. The Patmos processor contains scratchpad memories for time-predictable and low latency access to local data and instructions. The access to main memory is backed up by three different caches: (1) a method cache caches the instructions of full methods [9], (2) a stack cache caches data allocated on the stack [10], and (3) a data cache for the other data. For data where a data cache analysis cannot predict hits and misses, Patmos contains load and store instructions that bypass the cache. The compiler can emit those instructions when needed.[1]

The processor cores are connected by the Argo NoC [11]. The Argo NoC provides time-predictable movement of data between local scratchpad memories of the processors (message passing). When data is to be moved, the sender sets up the message, which is then transferred by the NoC from the sender's local scratchpad memory to the receiver's local scratchpad memory. To support many cores on a chip, Argo is available as asynchronous NoC to implement a globally asynchronous locally synchronous system.

To support larger programs and data structures we use external, shared memory. For the access to shared memory T-CREST provides two solutions: (1) the Bluetree memory tree with prefetching [12] and (2) the TDM based memory arbiter [13]. the Bluetree memory tree is optimized for the Predator memory controller [14], while the TDM arbiter is a general purpose burst based TDM arbiter.

B. The Software

T-CREST contains an adaption of the LLVM compiler that targets the Patmos instruction set and optimizes for the WCET [15]. As a C standard library, we have ported `newlib` for Patmos.

Patmos support has also been integrated into the WCET analysis tool aiT from AbsInt [16]. The compiler and the WCET analysis tool are integrated in two ways: (1) the compiler delivers information about the program, such as loop bounds and possible branch targets of indirect branches to the WCET analysis tool; (2) the WCET analysis tool delivers the WCET path in the control flow to guide the compiler to optimize that path. The T-CREST project also includes the open-source, academic WCET tool called platin [17].

C. Usage of T-CREST

T-CREST has now been in use for research and teaching for several years. As a platform, it offers: (1) a ready to use processor for real-time systems, (2) a platform for research on time-predictable multicore architectures and WCET analysis, and (3) a platform for teaching time-predictable computer architecture.

An avionic use case has been successfully ported to T-CREST [18]. In this evaluation use case it has been shown that the T-CREST multicore scales better with respect to the WCET than the LEON multicore processor. As a second use case, T-CREST was used for a signal processing application [19]. A musical effect device uses multiple cores for the individual effects and the NoC to move sample data between cores in the processing pipeline. This application shows that the bandwidth of the Argo NoC, even with an all-to-all schedule, is high enough to easily support this multicore application.

T-CREST serves as a platform for real-time architecture research. The TiCOS operating system has been ported to Patmos [20]. TiCOS is a time-predictable operating system that supports the ARINC 653 standard. The multicore operating system MOSSCA has been ported to T-CREST [21].

Patmos support also starts to appear in WCET analysis tools. For example, Heptane [22] has an experimental support for Patmos.[2] Furthermore, platin, one of the WCET analysis

[1] Private communication with Daniel Wiltsche-Prokesch.

[2] Private communication with Benjamin Rouxel and Isabelle Puaut

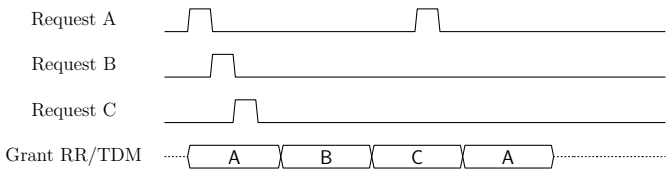

Fig. 2. Arbitration schemes round-robin (RR) and time-division multiplexing (TDM) for a request scenario that results in the same grants with RR and TDM.

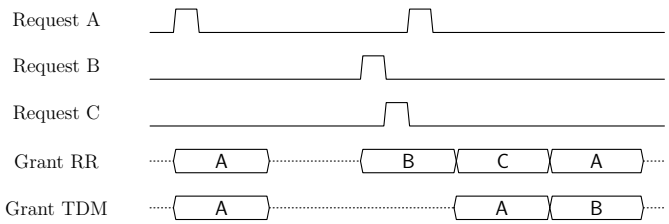

Fig. 3. A request scenario that results in a longer latency for requester A in the RR scheme than when using TDM.

tools from T-CREST, has been adapted to perform whole-system response-time analysis for fixed-priority real-time systems [23].

T-CREST has also been used in two TACLe summer schools on WCET analysis and time-predictable compilation and processor architecture.[3] Furthermore, at DTU we use T-CREST in a course of advanced computer architecture and TU Vienna bases its course on WCET analysis on the T-CREST platform.

IV. Achieving Time Predictability

The main technique used to achieve time predictability in T-CREST is time-division multiplexing (TDM) as the arbitration mechanism for shared resources. With TDM, also called time-triggered architecture [24], time is used for arbitration. A TDM round is divided into TDM time slots, which are often of equal length to offer equal bandwidth for all requesters. Each requester owns its time slot and is only served in that time slot. The beauty of TDM is that there is no interference at all between requesters. The access time for a request only depends on the its timing relative to the requester's allotted time slot. In the worst case, this time slot has just been missed and the request needs to wait for one full TDM round.

The main criticism of TDM arbitration is that it is not work conserving. This means that unused slots are not reused by a different core. However, this is a misconception in the context of real-time systems. For real-time systems, we need to provision resources for the worst case. But the worst case with, say, a round-robin arbiter is the same as for a TDM arbiter. Furthermore, the distance between consecutive TDM slots for a given requester is always constant. This information can be used to hide some of the latency when work is performed between TDM slots. This cannot be statically explored for WCET analysis with a round-robin arbiter.

Moreover, round-robin arbitration can result in a longer waiting time than with a TDM arbiter [25]. Consider three requesters A, B and C, competing for a resource. They are accessing it in following order: A, B, C, A. In TDM each access can only happen in a requesters own slot. If it misses a slot, one must wait for the next round. If all requesters are ready to access the resource, round-robin will look the same as TDM. This is considered the worst case for round-robin arbitration.

[3]See http://www.mrtc.mdh.se/projects/TACLe/tacle.eu/index.php/activities/summer-schools-forums/2014-venice.html and https://www.cister.isep.ipp.pt/news/451/tacle_summer_school_2016/

However, from a single processor's point of view this is not the worst case! Assume that the access of B and C is delayed by almost two slots. In that case B and C also delay A's second access to the resource. That means that less global traffic can result in longer latency for an access. In the timing analysis community, this is called a timing anomaly [26].

Fig. 2 and Fig. 3 show the two different request scenarios to a single shared resource with two arbitration mechanisms: round-robin (RR) and TDM. Fig. 3 depicts the timing anomaly that a globally lower request rate can result in a locally longer latency with round-robin arbitration.

In the NoC we use TDM in two components: the router and the network interface. The NoC router is organized as a 3-stage pipeline without any additional buffering. Arbitration to the 5 ports is based on TDM and is statically scheduled. No two packets compete on an output port. Packets are read from a local memory in the network interface before being injected into the NoC. This reading is implemented by a DMA machinery. As the packet injection into the NoC is itself scheduled in a TDM fashion, the DMA machinery itself is time sliced in a TDM fashion. Packets in the NoC are transmitted from a core local memory, through a network of routers, and into the receiver's core local memory in the allocated TDM slots without any buffering or flow control.

For larger data structures and the program code T-CREST uses external memory. This access to shared memory is also TDM arbitrated. Each core receives a time slot large enough for one burst read or write to fill or evict one cache line.

Another important aspect of TDM is a low-cost arbitration mechanism, which needs just counters and tables. Furthermore, TDM arbitration can be distributed to the clients, when all components have a common notion of time. This means that there is no central arbitration point (as it would be with round-robin or priority based arbitration). A central arbitration point results in limits of scalability, while TDM scales better. A common notion of time is trivial in clocked systems, but can also be achieved with mesochronous clocked sources of tokens flowing through an asynchronous NoC [11].

The Patmos processor is organized in a 5-stage pipeline, like many other RISC processors. Although usually not seen in this way, pipelining also implements TDM for processor resources such as the ALU. However, in standard architectures different pipeline stages often share a resource. One example is the access to memory on a cache miss. An instruction in the fetch

stage can miss in the instruction cache in the very same cycle as a load or store can miss in the memory stage. Although these two instructions may be completely independent, the access to the shared memory resource results in a timing dependency between those two instructions. In Patmos, we even time-slice the access to the memory from the processor caches. All cache loading in Patmos is performed in the same pipeline stage, the memory stage. Thus, cache misses can only happen in that single pipeline stage, which effectively provides TDM based arbitration between the caches. No instruction timing depends on any other instruction, resulting in a time composable processor architecture without any timing anomaly.

V. Conclusion

This paper presents the time-predictable multicore architecture T-CREST. Time predictability can be achieved by using static, predefined arbitration to shared resources. In most cases time-division multiplexing is the arbitration scheme used in T-CREST. With this static arbitration, we are able to completely avoid any timing interference between tasks executing on different processing cores on the multicore processor. This isolation enables static worst-case execution time analysis of real-time tasks, executing on a multicore processor.

Acknowledgment

I would like to thank Jens Sparsø for the discussion on TDM arbitration and for feedback on an early version of the paper.

The work presented in this paper was partially funded by the Danish Council for Independent Research | Technology and Production Sciences under the project PREDICT (http://predict.compute.dtu.dk/), contract no. 4184-00127A.

References

[1] M. Schoeberl, "Time-predictable computer architecture," *EURASIP Journal on Embedded Systems*, vol. vol. 2009, Article ID 758480, p. 17 pages, 2009.

[2] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, S. Hepp, B. Huber, A. Jordan, E. Kasapaki, J. Knoop, Y. Li, D. Prokesch, W. Puffitsch, P. Puschner, A. Rocha, C. Silva, J. Sparsø, and A. Tocchi, "T-CREST: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[3] M. Schoeberl, "Lessons learned from the EU project T-CREST," in *Design, Automation Test in Europe Conference Exhibition (DATE 2016)*. IEEE, March 2016, pp. 870–875.

[4] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, and J. Wolf, "Merasa: Multi-core execution of hard real-time applications supporting analysability," *Micro, IEEE*, vol. 30, no. 5, pp. 66–75, 2010.

[5] A. Hansson, K. Goossens, M. Bekooij, and J. Huisken, "CoMPSoC: A template for composable and predictable multi-processor system on chips," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 2:1–2:24, Jan. 2009.

[6] C. Paukovits and H. Kopetz, "Concepts of switching in the time-triggered network-on-chip," in *Proceedings of the 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2008)*, August 2008, pp. 120 –129.

[7] R. Obermaisser, H. Kopetz, and C. Paukovits, "A cross-domain multi-processor system-on-a-chip for embedded real-time systems," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 548 –567, nov. 2010.

[8] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S. Karlsson, and T. Thorn, "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)*, Grenoble, France, March 2011, pp. 11–20.

[9] P. Degasperi, S. Hepp, W. Puffitsch, and M. Schoeberl, "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: IEEE, June 2014, pp. 100–108.

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

[11] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. T. Müller, K. Goossens, and J. Sparsø, "Argo: A real-time network-on-chip architecture with an efficient GALS implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 479–492, 2016.

[12] J. Garside and N. C. Audsley, "Prefetching across a shared memory tree within a network-on-chip architecture," in *System on Chip (SoC), 2013 International Symposium on*, Oct 2013, pp. 1–4.

[13] M. Schoeberl, D. V. Chong, W. Puffitsch, and J. Sparsø, "A time-predictable memory network-on-chip," in *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, Madrid, Spain, July 2014, pp. 53–62.

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

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

[16] R. Heckmann and C. Ferdinand, "Worst-case execution time prediction by static program analysis," AbsInt Angewandte Informatik GmbH, Tech. Rep., [Online, last accessed November 2013].

[17] S. Hepp, B. Huber, J. Knoop, D. Prokesch, and P. P. Puschner, "The platin tool kit - the T-CREST approach for compiler and WCET integration," in *Proceedings 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach, Austria, October 5-7, 2015*, 2015.

[18] A. Rocha, C. Silva, R. B. Sørensen, J. Sparsø, and M. Schoeberl, "Avionics applications on a time-predictable chip-multiprocessor," in *24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2016)*. IEEE Computer Society, Feb 2016, pp. 777–785.

[19] D. S. Ausin, L. Pezzarossa, and M. Schoeberl, "Real-time audio processing on the T-CREST multicore platform," in *MCSoC 2017*.

[20] M. Ziccardi, M. Schoeberl, and T. Vardanega, "A time-composable operating system for the Patmos processor," in *The 30th ACM/SIGAPP Symposium On Applied Computing, Embedded Systems Track*. Salamanca, Spain.: ACM Press, April 13–17 2015.

[21] F. Kluge, M. Schoeberl, and T. Ungerer, "Support for the logical execution time model on a time-predictable multicore processor," in *14th International Workshop on Real-Time Networks*. Toulouse, France: ACM SIGBED Review, July 2016.

[22] D. Hardy, B. Rouxel, and I. Puaut, "The Heptane Static Worst-Case Execution Time Estimation Tool," in *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*, ser. OpenAccess Series in Informatics (OASIcs), J. Reineke, Ed., vol. 57. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 1–12.

[23] C. Dietrich, P. Wägemann, P. Ulbrich, and D. Lohmann, "Syswcet: Whole-system response-time analysis for fixed-priority real-time systems (outstanding paper)," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017, pp. 37–48.

[24] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[25] W. Puffitsch and M. Schoeberl, "On the scalability of time-predictable chip-multiprocessing," in *Proceedings of the 10th International Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2012)*. Copenhagen, DK: ACM, October 2012, pp. 98–104.

[26] T. Lundqvist and P. Stenström, "Timing anomalies in dynamically scheduled microprocessors," in *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS 1999)*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 12–21.