T-CREST

Journal of Systems Architecture 61 (2015) 449–471

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

T-CREST: Time-predictable multi-core architecture for embedded systems

Martin Schoeberl [a,*], Sahar Abbaspour [a], Benny Akesson [b], Neil Audsley [c], Raffaele Capasso [d], Jamie Garside [c], Kees Goossens [e], Sven Goossens [e], Scott Hansen [f], Reinhold Heckmann [g], Stefan Hepp [h], Benedikt Huber [h], Alexander Jordan [a], Evangelia Kasapaki [a], Jens Knoop [h], Yonghui Li [e], Daniel Prokesch [h], Wolfgang Puffitsch [a], Peter Puschner [h], André Rocha [i], Cláudio Silva [i], Jens Sparsø [a], Alessandro Tocchi [d]

[a] Technical University of Denmark, Denmark
[b] Czech Technical University in Prague, Czech Republic
[c] University of York, United Kingdom
[d] Intecs S.p.A., Italy
[e] Eindhoven University of Technology, The Netherlands
[f] The Open Group, Belgium
[g] AbsInt Angewandte Informatik GmbH, Germany
[h] Vienna University of Technology, Austria
[i] GMV, Portugal

ARTICLE INFO

Article history:
Received 14 July 2014
Received in revised form 19 January 2015
Accepted 2 April 2015
Available online 9 April 2015

Keywords:
Real-time systems
Time-predictable computer architecture

ABSTRACT

Real-time systems need time-predictable platforms to allow static analysis of the worst-case execution time (WCET). Standard multi-core processors are optimized for the average case and are hardly analyzable. Within the T-CREST project we propose novel solutions for time-predictable multi-core architectures that are optimized for the WCET instead of the average-case execution time. The resulting time-predictable resources (processors, interconnect, memory arbiter, and memory controller) and tools (compiler, WCET analysis) are designed to ease WCET analysis and to optimize WCET performance. Compared to other processors the WCET performance is outstanding.

The T-CREST platform is evaluated with two industrial use cases. An application from the avionic domain demonstrates that tasks executing on different cores do not interfere with respect to their WCET. A signal processing application from the railway domain shows that the WCET can be reduced for computation-intensive tasks when distributing the tasks on several cores and using the network-on-chip for communication. With three cores the WCET is improved by a factor of 1.8 and with 15 cores by a factor of 5.7.

The T-CREST project is the result of a collaborative research and development project executed by eight partners from academia and industry. The European Commission funded T-CREST.

1. Introduction

Safety-critical systems are important parts of our daily life. They have to be reliable, as our lives can depend on them. Examples are controllers in airplanes, braking controllers in cars, or train control systems. Those safety-critical systems need to be certified and the maximum execution time needs to be bounded and known so that response times can be assured when critical situations require a timely reaction. Note that just using a faster processor is not a solution for time predictability. Even with high performance processors in our desktop PCs we notice once in a while that the PC is "frozen" for a few seconds. For word processing we accept this minor inconvenience, but for a safety-critical system such a "pause" can result in a catastrophic failure.

The mission of T-CREST was to develop tools and build a platform that avoids such unexpected pauses. The T-CREST time-predictable platform simplifies the safety argument with respect to maximum execution time and leverages the possible performance improvement of multi-core systems for real-time systems. Thus the T-CREST platform results in lower costs for safety-relevant applications, in reduced system complexity, and at the same time in faster time-predictable code execution.

* Corresponding author.
E-mail address: masca@dtu.dk (M. Schoeberl).

http://dx.doi.org/10.1016/j.sysarc.2015.04.002
1383-7621/

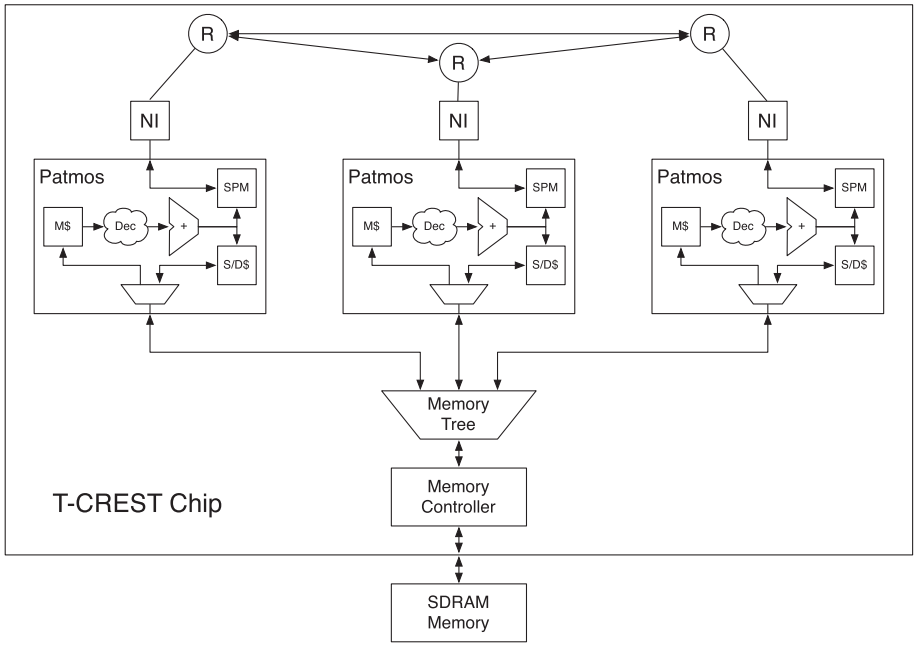


Fig. 1. The T-CREST platform consisting of Patmos processor nodes that are connected via an on-chip network for message passing communication and a memory tree to a memory controller for shared memory access.

Standard computer architecture is driven by the following paradigm: *make the common case fast and the uncommon case correct*. This design approach leads to architectures where computer architects optimize the average-case execution time at the expense of the worst-case execution time (WCET). Modeling the dynamic features of current processors, memories, and interconnects for WCET analysis often results in computationally infeasible problems. The bounds calculated by the analysis are thus overly conservative.

We need a sea change and we shall take the constructive approach by designing computer architectures where predictable timing is a first-order design factor. For real-time systems we thus propose to design architectures with a new paradigm: *make the worst-case fast and the whole system easy to analyze*. Despite the advantages of analyzable system resources, only a few research projects exist in the field of hardware optimized for the WCET.

Within T-CREST we propose novel solutions for time-predictable multi-core architectures. The resulting time-predictable resources (processors, interconnect, memories, etc.) are an easy target for WCET analysis and the WCET performance is outstanding compared to current processors. Time-predictable caching and time-predictable chip-multiprocessing (CMP) provides a solution for the need of increased processing power in the real-time domain.

Besides the hardware, we developed a compiler infrastructure in the project. We developed WCET-aware optimization methods using the known behavior of the hardware. We adapted the WCET analysis tool aiT to support the developed hardware and guide the compilation.

1.1. The T-CREST platform and supported programming models

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 two industry use cases. New languages or new operating system concepts for time-predictable real-time systems are not in the scope of this project. Most of the T-CREST hardware is open-source under the industry friendly, simplified BSD license.[1]

The overall architecture of the T-CREST platform is shown in Fig. 1. The platform consist of a number of processor nodes and two networks-on-chip; one that supports message passing between processor nodes and one that provides access to a shared external memory. A processor node includes a Patmos processor [1], special instruction and data cache memories (a method cache [2] and a stack cache [3]) and local private scratchpad memories (SPMs) for instructions and data. Message passing is implemented using DMA driven writes from the local SPM in a processor node to the SPM in a remote processor node. The message passing NoC [4–6] supports this by offering virtual end-to-end channels. The memory NoC [7] provides access to a shared external SDRAM memory that is controlled by a real-time memory controller [8–10].

The memory NoC and the memory controller do not include any hardware support for cache coherency. The main means of processor-to-processor communication is the message passing NoC. The shared memory is primarily intended for initialization and to extend the memory resources beyond what can fit onto a single chip. Processor-to-processor communication via shared memory is permitted, but coherency mechanisms are implemented in software. This allows for sharing of large data structures protected by locks that implement cache coherency in software.

The decisions not to use hardware based cache coherency and the inclusion of hardware support for message passing is in line with many current CMPs and multi-processor systems on chip [11–13]. The main reason behind this trend is that the cost of implementing cache coherency in hardware in a multi-processor platform using a packet switched NoC is becoming prohibitive, both in terms of area and in terms of network load.

We have adopted these fundamental architectural decisions and the scope of this paper is to study how message passing and shared memory access can be implemented in a time-predictable manner and optimized with an aim of minimizing the WCET. The platform is further supported by a compiler also developed with

[1] see https://github.com/t-crest.

a focus on WCET, and the aiT WCET-tool has been extended to support the Patmos processor.

1.2. Project contributions

The main contributions of the T-CREST project are:

- The T-CREST platform provides a time-predictable multi-core platform so that we can perform reliable WCET analysis and tighter WCET bounds enable higher processor utilization.
- The T-CREST platform supports a globally-asynchronous locally-synchronous timing organization to enable the implementation of large scale multi-core platforms. An asynchronous network-on-chip connects synchronous processors.
- The T-CREST memory network-on-chip and the memory controller support time-predictable accesses to the shared main memory to allow WCET analysis of such accesses.
- The T-CREST compiler and the WCET analysis tool aiT are tightly integrated to support the T-CREST processor Patmos and to enable WCET driven compiler optimization.
- The T-CREST platform improves the worst-case performance logarithmically in the number of processing cores, under the assumption that the application is a good candidate for parallelization, to provide more processing power for future, more complex embedded real-time systems. For a digital-signal processing application we showed that three cores improve the WCET by a factor of 1.8 and 15 cores by a factor of 5.7.
- Most technology of T-CREST is available in open source and we consider this as a main contribution to the real-time architecture research community. The Patmos reference handbook [14] contains download and detailed build instructions.

1.3. Paper organization

The paper is organized into 8 sections. The following Section 2 compares the T-CREST approach with related work. We organized the rest of the paper according to the different research and development areas of the T-CREST project. Section 3 introduces the time-predictable processor Patmos developed within T-CREST. Section 4 describes T-CREST's core-to-core message passing network-on-chip that supports asynchronous message passing across point-to-point virtual circuits. Section 5 presents the memory hierarchy of the T-CREST platform and explains how to provide time-predictable access to off-chip DRAM memory. Section 6 describes the compilation tool chain for the Patmos architecture and its integration with WCET analysis. Section 7 presents the evaluation of the capability of the T-CREST platform to host real industrial applications with delicate predictability requirements, using domain-specific use case applications from the avionics and railway domains. Finally, Section 8 concludes.

2. Related work

Research on time-predictable architectures is a steadily growing field that is gaining momentum. This section presents related research papers and discusses the relation between T-CREST and other projects.

2.1. Related projects

The projects MERASA, parMERASA, JEOPARD, PREDATOR, ALL-TIMES, and Scalopes are related to the T-CREST project in some regards.

The FP-7 project MERASA (Multi-Core Execution of Hard Real-Time Applications Supporting Analysability) [15] investigated time-predictable execution on a bus-based CMP with multi-threaded version of the TriCore processor. In contrast to MERASA, we developed a network-on-chip based multi-core architecture. Furthermore, we tackled the time predictability challenge by developing a new processor architecture, with a WCET optimized instruction set, and the supporting compiler.

The FP-7 project parMERASA is a followup project of MERASA and tackles the parallelization of applications [16]. Unlike MERASA, parMERASA targets a network-on-chip based multi-core architecture. The parMERASA project focuses on tools and system-level software to support parallelization, WCET analysis and verification. In contrast to T-CREST, parMERASA uses a simulator for their target platform rather than implementing a time-predictable architecture.

The FP-7 project JEOPARD (Java Environment for Parallel Realtime Development) investigated architectures and tools for real-time Java on CMP systems. JEOPARD considered all levels of the architecture: the hardware, the operating system, the JVM, static analysis tools, and evaluation of the architecture with three use cases. While JEOPARD targeted real-time Java, the Java processor JOP [17] provided an inspiration for Patmos, in particular with regard to the cache design. Furthermore, TDMA based memory access arbitration and its WCET analysis was explored within JEOPARD [18].

The FP-7 project PREDATOR studied the interplay between efficiency requirements and critical constraints in embedded system design, aiming at a design for predictability and efficiency. Results on the notion of predictability in general and on the predictability of hardware and software features lead to advice what to avoid when developing a predictable system [19]. This work provided one of the foundations for the work in T-CREST. The processor design in T-CREST follows the design principles of predictable architectures elaborated in the PREDATOR project.

The main goal of the FP-7 project ALL-TIMES [20] was to enhance the timing analysis technology for safety-critical embedded systems. The project aimed at combining available timing tools from SMEs and universities into an integrated tool chain using open tool frameworks and interfaces, and at developing new timing analysis methods and tools where appropriate.

In the context of the Scalopes project, the CompSOC platform [21] and tool flow [22] were developed. The NoC used in the context of Scalopes (Aethereal/aelite) provided an inspiration for the asynchronous NoC developed in T-CREST. However, the T-CREST NoC uses an architecture that reduces system-level cost significantly.

2.2. Time-predictable processors

Within T-CREST we used the results from the memory access arbitration and adapted the cache solutions [23] from the Java processor JOP. JOP is a time-predictable processor that uses Java bytecode as its instruction set. In contrast, the T-CREST processor Patmos uses a RISC instruction set to enable the execution of more traditional languages like C. A key feature of JOP is its cache architecture, with an instruction cache that caches whole methods [24] and separate caches for stack and heap data. Patmos keeps this general cache architecture, but adapts it to fit a RISC instruction set [2,3]. Within the T-CREST project, we also adapted the analysis methodology developed for the object cache of JOP [25] in the analysis tool `platin` [26,27].

The focus of the precision timed (PRET) machine [28] is primarily on repeatable timing and less on predictable timing. A deadline instruction can be used to enforce repeatable timing of a task. A first simulation of their PRET architecture is presented in [29]. The first hardware implementation of PRET implements the ARM instruction set [30,31]. PRET implements a RISC pipeline and

performs chip-level multithreading for four threads to eliminate data forwarding and branch prediction [32]. Scratchpad memories are used instead of instruction and data caches.

A recent version of PRET, FlexPRET [33], extends PRET to support mixed-criticality systems. FlexPRET supports two different thread types, hard and soft real-time threads, directly in the hardware. Both thread types have fixed slots assigned in the fine-grain thread scheduling. However, soft real-time threads can use slots that are not used by a hard real-time thread (e.g., because of stalling or a thread has finished its current release). FlexPRET switched the instruction set to RISC V [34].

The main difference between our proposal and PRET is that we focus on time predictability [35,36] and PRET on repeatable timing. Our approach therefore allows run-time dynamism in scheduling and execution, whereas PRET is essentially static, resulting in a higher implementation cost. In our opinion a time-predictable architecture does not need to provide repeatable timing as long as WCET analysis can deliver tight WCET bounds. Furthermore, PRET implements the standard instruction set, whereas we explore an instruction set that supports WCET based optimization and WCET analysis. In contrast to the PRET we use a dual-issue pipeline for maximum single thread performance. For multi-threaded applications we provide a multi-core version of Patmos.

Fernandez et al. [37] present a detailed study of task interference in a Leon4 based quad-core processor developed by Aeroflex Gaisler and used by the European Space Agency. The processor is a conventional multiprocessor that has been developed with critical real-time systems in mind. The study shows that interference may cause the execution time of a task to increase by a factor of 2–9 times. The results are a strong argument in support of time-predictable multi-core architectures.

2.3. Core-to-core network-on-chip

To achieve time-predictability of the on-chip communication, the NoC needs to provide end-to-end connections that can be analyzed individually. Ways to implement end-to-end connections are circuit-switching (physical or virtual) or by controlling the rate of traffic flows.

SoCBUS [38] and the NoC used in the 4S-platform [39] implement pure circuit switching. In this approach it is straightforward to analyze the circuits provided. However, the 4S platform NoC uses static circuits. This limits flexibility and may potentially waste resources and the approach is best suited for relatively static streaming applications. In this respect SoCBUS is more flexible but its dynamic dial-up mechanism does not in itself guarantee the establishment of a circuit in bounded time. To establish a bound some form of system level analysis is needed. TDM as used in Argo always guarantees the connections established in the schedule and offers more flexibility on the sharing of resources.

Virtual circuit switching is an alternative approach and TDM is a way to implement virtual circuit in a time-multiplexing way. NoCs that follow the TDM approach are Æthereal [40], aelite [41], Nostrum [42] and TTNoC [43,44], and Argo. Argo differs in its novel timing organization. Argo employs an asynchronous implementation of routers and a novel network interface design for a reduced system-level cost.

An alternative approach for time predictability is to apply rate control in the traffic flows. MANGO [45] and the Kalray NoC [46] follow this approach. MANGO implements an arbitration mechanism for virtual channels on links and Kalray NoC enforces a throughput limit on traffic flows. Schedulability analysis [47,48] and network calculus [49,50] are analytical approaches that aim at solving the contention problem and offering time-guarantees. However, the above techniques lead to high cost in the router implementation. Argo avoids this implementation complexity.

2.4. External memory access

External memory access in the T-CREST platform is provided by connecting the cores via a tree-shaped memory interconnect to a memory controller back-end. Using a tree-shaped interconnect is motivated by two criteria: (1) a task's memory communication and inter-task communications requirements are not necessarily the same and (2) conventional arbiters do not scale to today's multi-core requirements [51].

Separating the requirements of memory access and task communication is nothing new, for example the MEDEA architecture [52] provides separate arbitration for both communication and memory requests. The Tilera Tile processor [53] incorporates entirely separate networks for tile-to-tile accesses, I/O accesses, and shared memory accesses. While these works separate the inter-core and memory communication into distinct networks, no work currently separates them into entirely different network topologies.

Tree-based interconnects have been used in other work to connect processors to memory. Balkan and Uzi [54] and Rahimi et al. [55] use a "mesh-of-trees" approach in order to connect multiple processors to multiple memory banks. The approach presented here uses a similar, but simpler interconnect, since only one memory "bank" is used.

The tree approach also allows for scalable arbitration within interconnects, which are distributed across the tree, rather than being realized as a single large arbiter. Small TDM arbiters at the leaves of the tree can control the access to the memory tree in a scalable distributed way [56].

Another tree-based approach is presented by Gomony et al. [51], but encodes a priority within each packet and uses a generic arbitration framework. Each multiplexer then admits the packet with the highest priority. This has the advantage of being able to issue requests in work-conserving mode by adding a fixed offset to the static priority if the requestor is exceeding its given bounds. Both show that using a distributed approach allows the memory interconnect to run at a much higher frequency, although both approaches require global clock synchronization.

The T-CREST memory controller back-end differentiates from related memory controller back-ends by considering the following two challenges: (1) memory clients in complex systems comprising multiple types of processing elements that may have different request sizes and (2) different systems require different trade-offs between (worst-case) execution time, bandwidth, and power consumption.

Existing memory controller back-ends do not fully satisfy these diverse requirements as most current controllers are not designed with real-time applications in mind and do not provide bounds on WCET of transactions [57–59]. On the other hand, existing real-time memory controllers are using either (semi-)static command scheduling and cannot provide low average execution time to memory traffic [60–62] or dynamic scheduling, but are limited in architecture or analysis to a single transaction size and memory map configuration [63–65].

Looking more specifically at work from related projects, the PRET memory controller [62] aims at achieving repeatable and predictable timing by mapping memory clients to privatized memory banks that are then accessed using time-division multiplexing. The command scheduling of each memory request is done statically and takes a fixed amount of time. The benefit of this scheme is that memory clients cannot interfere with each other's bank state by e.g., closing rows opened by another thread. The drawback of this

approach is that the number of threads is limited by the number of available memory banks, making this solution unsuitable for the T-CREST project.

The MERASA memory controller [66] is more similar to the T-CREST memory controller in the sense that it does not privatize memory banks and uses dynamic command scheduling. The command-scheduling algorithm is a modified version of the DRAMSim memory simulator [67], although the modifications to the original scheduling algorithm are not specified and there is no hardware implementation. In addition, the analysis is limited to a fixed transaction size and a single memory map configuration.

2.5. Compiler

The WCET-aware C compiler (WCC) [68,69] is a custom developed C compiler that focuses on WCET optimization, targeting Infineon TriCore microcontrollers. The AbsInt aiT tool performs WCET analysis. Analysis results such as basic block execution times, encountered instruction cache misses, or the found worst-case path are back-annotated to the intermediate representation of WCC to be used by high-level optimizations. For this, all optimization and transformation passes in WCC need to maintain a mapping between the blocks of the intermediate and low-level representations. The work on the WCC compiler done in the PREDATOR project represents an important first step towards developing WCET-aware compilation techniques for single-core architectures by selecting between alternative code sequences.

In contrast to the WCC compiler, the T-CREST compiler is based upon an existing open-source, industrial-strength compiler, the LLVM [70] framework. General and target-independent compiler passes and tools such as the `platin` toolkit are complemented by dedicated support for architecture-specific features of Patmos, which has been co-designed together with the hardware platform itself.

Kirner et al. transform flow information in parallel to high-level optimizations [71]. Their transformation technique requires the compiler to generate control flow update rules for optimizations that modify the control-flow graph or change loop bounds or other flow constraints. These update rules specify the relation between edge-execution frequencies before and after the optimization, and are used to consistently transform all flow constraints affected by the optimization.

In contrast to this work and the WCC compiler, in the T-CREST compiler framework existing compiler passes are not modified to maintain any mappings or to record all performed transformations. Instead, the compiler uses novel techniques to transform meta-information among different levels of program representation. A combination of source code markers, compiler analyses, and relation graphs is used to transform flow facts and to back-annotate analysis results. This approach enables a simple integration of WCET analyses and the LLVM framework, and allows existing high-level compiler analyses to improve the precision of the WCET-analysis result.

Few code transformation techniques other than the single-path approach have been proposed that target the time-predictability of programs. Negi et al. [72] presented a transformation to reduce the number of paths required to be analyzed by making infeasible paths explicit or by factoring out code blocks with constant execution time. Both transformations seem hard to be performed automatically in a compiler and no general solution has been supplied so far.

The implementation of single-path code generation in a compiler backend has been proposed long ago [73] but has never been actually implemented before. Yan and Zhang [74] study the application of the single-path approach at the assembly level by a compiler in the context of a compilation framework for VLIW processors. They limit their investigation of the single-path approach to basic blocks of innermost loop regions, omitting loop transformation and interprocedural support.

2.6. WCET analysis

A comprehensive survey of methods and tools for determining the WCET is given in [75]. The most successful formal method for WCET computation is static program analysis based on abstract interpretation. Static program analyzers compute information about the software under analysis without actually executing it. Most interesting program properties—including the WCET—are undecidable in the concrete semantics. The theory of abstract interpretation [76] provides a formal methodology for semantics-based static analysis of dynamic program properties where the concrete semantics is mapped to a simpler abstract semantics. The static analysis is computed with respect to that abstract semantics, enabling a trade-off between efficiency and precision. A static analyzer is called sound if the computed results hold for any possible program execution. Applied to WCET analysis, soundness means that the WCET bounds will never be exceeded by any possible program execution. Abstract interpretation supports formal soundness proofs for the specified program analysis.

In addition to soundness, further essential requirements for static WCET analyzers are efficiency and precision. These properties are related to the predictability of the hardware [77,78] and software [79] that is being analyzed.

Over the last few years, a more or less standard architecture for timing analysis tools has emerged [80,81] which is composed of three major building blocks: (1) control flow reconstruction and static analyses for control and data flow, (2) micro-architectural analysis, computing upper bounds on execution times of basic blocks, and (3) path analysis, computing the longest execution paths through the whole program.

The commercially available tool aiT[2] [82,83] implements the architecture outlined above. The tool has been successfully employed in the avionics [84,85] and automotive [86] industries to determine precise bounds on execution times of safety-critical software.

Heptane[3] is an open-source static WCET analysis tool developed at IRISA. While it initially used a tree-based WCET computation approach [87], more recent versions use—like most other WCET analysis tools—an integer linear programming approach.

OTAWA[4] is an open-source static WCET analysis framework developed at the University of Toulouse [88]. It is designed to enable the combination of analysis algorithms and to simplify the implementation of new approaches.

WCA is the WCET analysis tool for the Java processor JOP [89]. While the former presented WCET analysis tools target several general-purpose embedded processors, WCA was specifically designed for the time-predictable processor JOP. This tight coupling of the hardware design and WCET analysis enabled co-design of architectural features and their analysis, e.g., an object cache and its analysis [25]. Within T-CREST we approached the design of the architecture in a similar way by getting feedback from the WCET analysis tool aiT on Patmos features.

An alternative to static WCET analysis is measurement-based WCET analysis. On the one hand, measurement-based WCET analysis does not require a formal model of the processor and therefore promises easy adaption to new architectures. On the other hand, it faces the challenge that it is difficult to provoke a worst-case scenario during the measurements and guarantee the soundness of

[2] http://www.absint.com/ait/.
[3] http://www.irisa.fr/alf/index.php?view = article&id = 29.
[4] http://www.otawa.fr/.

the WCET bound. An example for a commercial measurement-based WCET analysis tool is RapiTime[5] by Rapita Systems [90]. The MERASA project used both OTAWA and RapiTime to evaluate their hardware design.

3. The processor

The basis of a time-predictable system is a time-predictable processor. Within T-CREST we developed the time-predictable processor Patmos [1]. Patmos is a 32-bit, RISC-style microprocessor optimized for time-predictable execution. The user can configure Patmos to include a two-way pipeline for high performance or a single-way pipeline to save hardware resources.

Patmos is statically scheduled, in the sense that all instruction delays are explicitly visible at the instruction set architectural (ISA) level and the programmer or compiler need to respect the pipeline delays to guarantee correct execution. This approach simplifies the processor model for WCET analysis and helps to improve the latter's accuracy.

The modeling of the memory hierarchy is critical for WCET analysis. Patmos simplifies this task by offering caches that are especially designed for WCET analysis. Accesses to different data areas are different with respect to WCET analysis. Static program analysis can easily track addresses of static data, constants, and stack allocated data. Heap allocated data on the other hand demands for different caching techniques to be analyzable [91]. Therefore, Patmos contains two data caches, one for the stack and one for other data. Access to non-analyzable addresses (e.g., heap allocated structures) bypass the data cache with the non-cached load and store instructions.

3.1. Fully predicated instruction set

The Patmos instruction set is similar to a RISC style load/store instruction set. Instructions take at most three register operands. Only the first pipeline executes control-flow instructions and instructions that access memory, while both pipelines can execute arithmetic and logic instructions.

To reduce the number of conditional branches and to support the single-path programming paradigm [92,93], the compiler can predicate all instructions. Compare instructions, which the compiler can predicate as well, set predicates. Patmos has 8 predicate registers; logic operations like AND and OR operate directly on these predicate registers.

Apart from the usual encoding of constants in the 32-bit instruction words, Patmos also supports operations with 32-bit constants, where the second slot of the instruction bundle contains the constant. This feature is also present if Patmos contains only a single pipeline.

The type of the accessed data area is explicitly encoded with the load and store instructions. This feature helps the WCET analysis to distinguish between the different data caches. Furthermore, an earlier stage in the pipeline can detect which cache a load or store instruction will access.

3.2. Dual-issue pipeline

Patmos contains 5 pipeline stages: (1) instruction fetch, (2) decode and register read, (3) execute, (4) memory access, and (5) register write back. Fig. 2 shows an overview of Patmos' pipeline.

The two pipelines share the register file with 32 registers. Patmos supports full forwarding between the two pipelines. The basic features are similar to a standard RISC pipeline. Patmos splits the data cache for different cache areas. Typed load and store instructions distinguish between the different caches.

To simplify the diagram in Fig. 2, we omitted forwarding and external memory access data paths. We implemented the method cache (M$), the register file (RF), the stack cache (S$), the data cache (D$), and the scratchpad memories (SP) in on-chip memories of an FPGA. All on-chip memories of Patmos use registered input ports. As we cannot access the memory-internal input registers, we duplicate the program counter (PC) with an explicit register. The instruction register (IR) stores the instruction fetched from the method cache. The register file is using that instruction also to fetch the two (four in a dual issue configuration) register values during the decode stage.

As Patmos provides full forwarding from both pipelines, this forwarding network consumes a lot of resources. If the full power of dual issue is not needed, the user can configure Patmos as single-issue pipeline.

3.3. Local memories

Patmos contains three caches (method, data, and stack cache). The size of all caches can be configured before the hardware generation. Patmos also contains (optional) scratchpad memories (SPMs) for instructions and data. A program can use these SPMs in addition to caches or instead of caches, when code and/or data caching shall be under program control.

To distinguish between the different caches and SPMs, Patmos implements typed load and store instructions. The compiler (for the stack cache) or the programmer (for the data SPM and IO devices) assigns the type information.

3.3.1. Method cache

Patmos contains a method cache [2] that stores whole functions. We use the term *method cache* as this form of caching has been originally introduced for a Java processor [24]. Caching whole functions means that the processor may load full functions on a call or a return. All other instructions of Patmos hit in the method cache. Our assumption is that those possible miss points allow for an easier and more precise WCET analysis. We developed a scope based WCET analysis for this method cache [27].

The assumption of a method cache is that the cache is larger than all individual functions in a program. However, not all programs guarantee this assumption. Furthermore, an optimizing compiler inlines functions to avoid the call and return overhead, leading to even bigger functions. To mitigate this issue we have implemented a function splitter that splits too large functions into smaller (sub) functions that fit into the cache (see Section 6).

3.3.2. Stack cache

The latency bounds of memory accesses are crucial for time-predictability. For the calculation of the WCET bound, cache analysis tries to statically predict hits and misses [94]. With a known address of a load or store instruction we can classify the operation as a hit or miss. Addresses of heap-allocated data are only known at runtime and therefore not statically predictable. Since accessing an unknown address can destroy the cache state during analysis, we use a separate cache for stack allocated data, which improves the time-predictability of the design. This design separates predictable and unpredictable load and store operations to different caches (or uncached accesses).

Without dynamically sized stack objects, we can statically determine the addresses of data allocated on the stack. Moreover, a dedicated stack cache reduces the number of loads from the data cache and decreases the number of slow main memory accesses. Furthermore, it eliminates some long latency stores

[5] http://www.rapitasystems.com/products/rapitime.

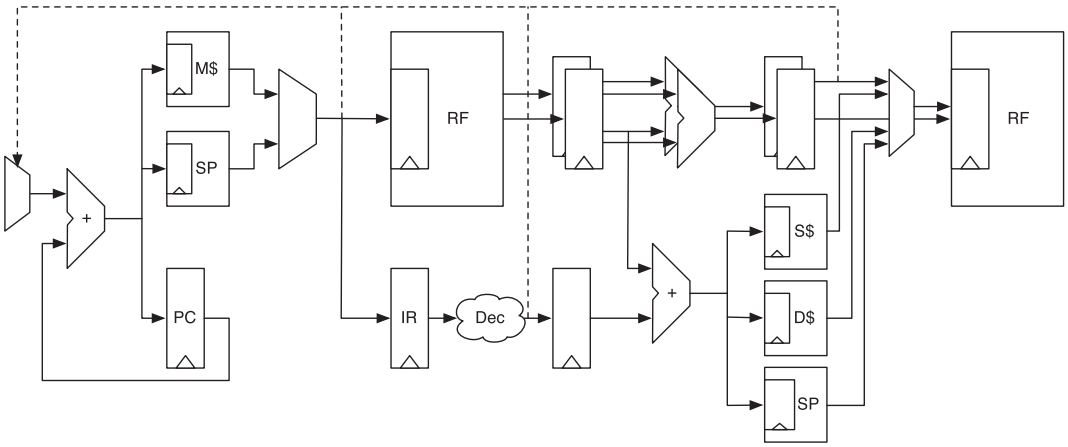


Fig. 2. Pipeline of Patmos with fetch, decode, execute, memory, and write back stages.

to the write-through data cache, thereby reducing the WCET of the design.

The stack area in a program contains the return address, registers saved by the current function, and local variables and data structures. Particular uses of the stack in C-like languages (e.g., dynamic allocation, passing pointers to addresses on the stack, stack objects exceeding the stack cache size) are not directly supported by the predictable stack cache design. For this reason, the compiler manages a *shadow stack* outside the stack cache, where objects not appropriate for the stack cache are allocated.

The Patmos stack cache [3] provides a window into the main memory, implemented using only two processor-internal registers. Three instructions manipulate the stack cache: (1) *reserve* reserves space in the stack cache, (2) *free* frees space on the stack cache, and (3) *ensure* forces data to be available in the stack cache. Only the *reserve* and *ensure* instructions may trigger a possible exchange with the main memory (spill and fill). All stack cache load and store instructions hit in the stack cache and thus we model them as single cycle operation for the WCET analysis. The extent to which stack manipulation instructions actually cause memory transactions can be statically predicted [95]. A specialized analysis bounds the stack cache utilization throughout the program and remains scalable by only introducing context-sensitivity when needed.

Call stacks are usually shallow and standard optimizations available in modern compilers (e.g., LLVM) are effective at reducing stack frame sizes; therefore, even a small stack cache provides good hit rates.

3.3.3. Data cache

We have two implementations of the data cache: (1) a direct-mapped cache and (2) a two-way set-associative cache with a least recently used (LRU) replacement policy. Set-associative caches, as long as they use the LRU replacement strategy, are very predictable and fully supported by the aiT WCET analysis tool. The data caches use write-through and no allocation on a write. We chose write-through as it is hard to predict statically when a write-back operation happens, and thus many state-of-the-art WCET analysis tools do not support write-back caches. The design decision to use a write-through policy is an excellent example how WCET analyzability influences the hardware design for a time-predictable processor—we optimize for the WCET. A write-back cache would actually increase the WCET bound, as each cache miss penalty includes a possible cache write-back. To mitigate the performance effects of the write-through policy, we implemented a small buffer that combines writes into bursts.

For statically unknown load and store addresses, Patmos has load and store instructions that bypass all caches.

3.3.4. Miss detection and pipeline stalling

The cache configuration of Patmos is special with respect to miss detection: for all three caches, the memory stage detects all misses and stalls the pipeline. This is normal for a data cache, but a standard instruction cache misses in the fetch stage. However, the method cache performs miss detection only on call and return. Therefore, these instructions can stall as well in the memory stage.

The consequence of a single stalling pipeline stage is twofold: (1) the hardware implementation of stalling is simpler and (2) cache analysis becomes simpler. No two instructions can trigger a cache miss in the same clock cycle for two caches. We consider this feature to contribute to a timing-composable architecture, as we can analyze different caches independently.

4. The core-to-core message passing network-on-chip

The core-to-core communication NoC is packet switched and it uses source routing. The topology is a bi-torus. The NoC supports asynchronous message passing across point-to-point virtual circuits, and it implements these using DMA-driven block-transfers from the local SPM in one processor node into an SPM in a remote node. We named the T-CREST NoC Argo.

We can implement virtual circuits using (statically scheduled) time division multiplexing (TDM) of the resources (routers and links) in the NoC or by using non-blocking routers in combination with a rate-controlled service discipline [96]. We decided for a TDM-based NoC for two reasons: (i) A TDM router avoids dynamic arbitration and virtual channel buffers and its hardware implementation is correspondingly simple. (ii) TDM avoids interference of traffic and timing analysis is straightforward. In a TDM-based NoC, the network interface (NI) injects packets into the network of routers and links according to a predetermined periodic and static schedule.

The schedule determines when the NI injects a packet for a given destination into the packet-switched NoC, and once injected the packet travels along a pipelined path from source to destination. The schedule guarantees absence of deadlocks, and the fact that it avoids arbitration, buffering, and flow control results in small and efficient circuit implementations. For symmetric networks and an all-to-all communication pattern we found a simple heuristics to generate the TDM schedule [97]. For application specific schedules we use a metaheuristic scheduler [98].

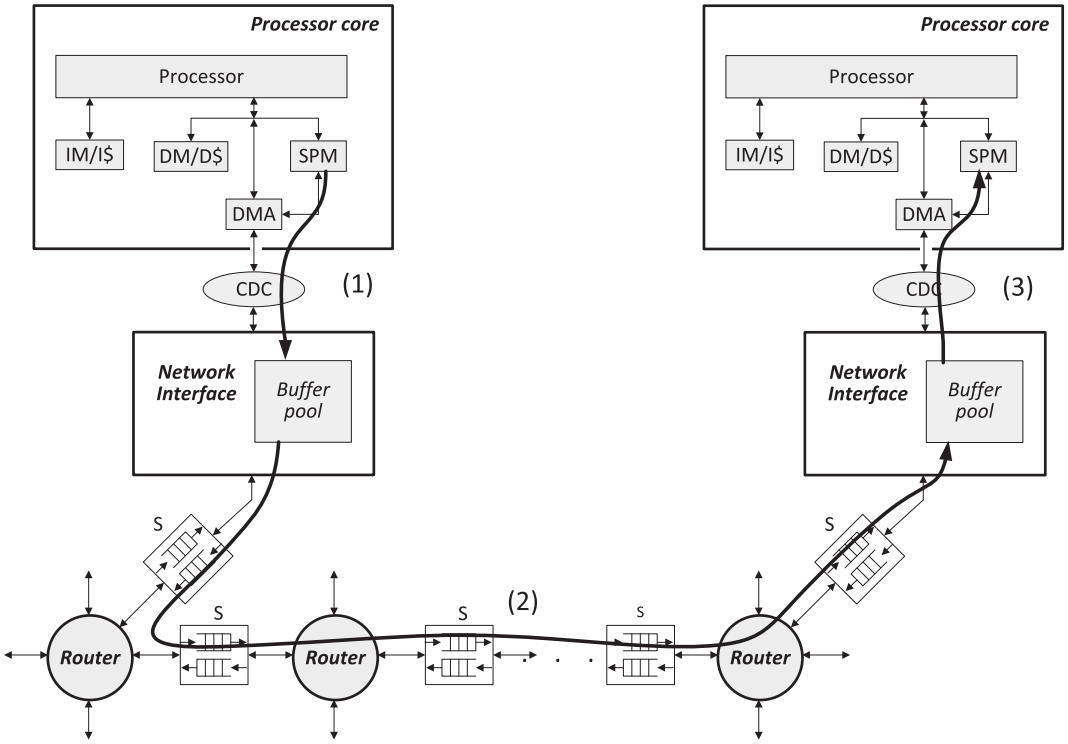


Fig. 3. Data transfer on a traditional TDM-based NoC.

The functionality of the T-CREST core-to-core NoC is similar to the aelite NoC [99]. The key contribution of the T-CREST project is the novel hardware architecture that results in a more efficient and smaller circuit implementation that we present below.

A traditional implementation, illustrated in Fig. 3, implements a transmit/receive buffer and a DMA controller in both ends of every single virtual circuit. The DMA controllers transmit and receive packets to/from buffers in the NI. Data moves across the NoC from the NI in the source to the NI in the destination using a static TDM schedule. The result is that data moves from the sending processor node to the receiving processor node in three steps: (i) from the sending processor into the NI, (ii) from this NI across the network of routers and links and into the destination NI, and (iii) from this NI into the receiving processor node. And because these steps are independent/autonomous, flow control has to be introduced to avoid overflow and underflow of buffers in the NIs. In this way the NoC as a whole does not enjoy the benefits of TDM—that it eliminates the need for arbitration, buffering, and flow control.

By rethinking the architecture and keeping the essence of TDM in mind we have developed a new architecture [100] where we have moved the DMA controllers into the NIs and integrated them with the TDM scheduling, see Fig. 4. As the TDM schedule interleaves the DMA transfers out of a processor node, only one DMA controller can be active at a time. This allows a single table-based implementation of all the pointers and counters of all the DMA controllers. In combination with a single shared SPM in every processor this allows TDM driven data transfer from the source SPM to the destination SPM without any buffering or flow control, as illustrated in [100]. The DMA moves data out of a processor node in an interleaved fashion across a sequence of TDM schedule periods. Therefore, the size of a TDM slot (and the resulting packet size) can be small, which helps to keep the latency short. This new architecture results in substantial area reductions in the NIs.

Another key feature of the architecture is its efficient support of a globally-asynchronous locally-synchronous (GALS) timing organization of the entire platform. The new architecture avoids all clock domain crossings (and all the associated synchronization and metastability issues) except for the interface used by the processor to set up the DMA controller. Here a clock domain crossing module (CDC) must be used as illustrated in Fig. 4. The individual processors may be independently clocked possibly exploiting voltage-frequency scaling. The dual ported SPM supports clock-domain crossing between the processor node and the NI. The NIs are driven from a single clock source. However, skew in the reset and clock nets on the chip between the NIs do not guarantee in phase operation. More precisely, the TDM slot counters in the NIs that control the scheduling can be off by one or more clock ticks. The term mesochronous denotes this mode of operation characterized by the use of a single oscillator and bounded skew [101, Ch.10].

To cope with the skew between the NIs the network of routers and links that connect the NIs must offer some timing elasticity. This is traditionally provided by adding dual-clock FIFOs in every input port of every router as illustrated in Fig. 3. The addition of these FIFOs roughly doubles the area and power consumption of a clocked synchronous router [5]. Our NoC uses asynchronous routers instead and as asynchronous pipelines inherently behave as self-timed ripple FIFOs we avoid these explicit synchronizer FIFOs. The asynchronous router uses the same three stage pipelined design as the synchronous router, the only difference is the use of asynchronous handshake latches instead of clocked registers, as Fig. 5 shows. The three pipeline stages of the router are: (i) link traversal, (ii) header-parsing unit (HPU), and (iii) the crossbar switch (Xbar). The handshake latch used in our design consists of a normal enable latch and a handshake controller that implements a two-phase (NRZ) bundled-data protocol [102]. We present the asynchronous router design in more detail in [5].

Fig. 6 shows the packet format used in our architecture. A packet consists of three phits, a header phit and two payload phits. A phit is the smallest atom that is transmitted in a (clock) cycle. Each phit contains a 32-bit word and three additional control bits

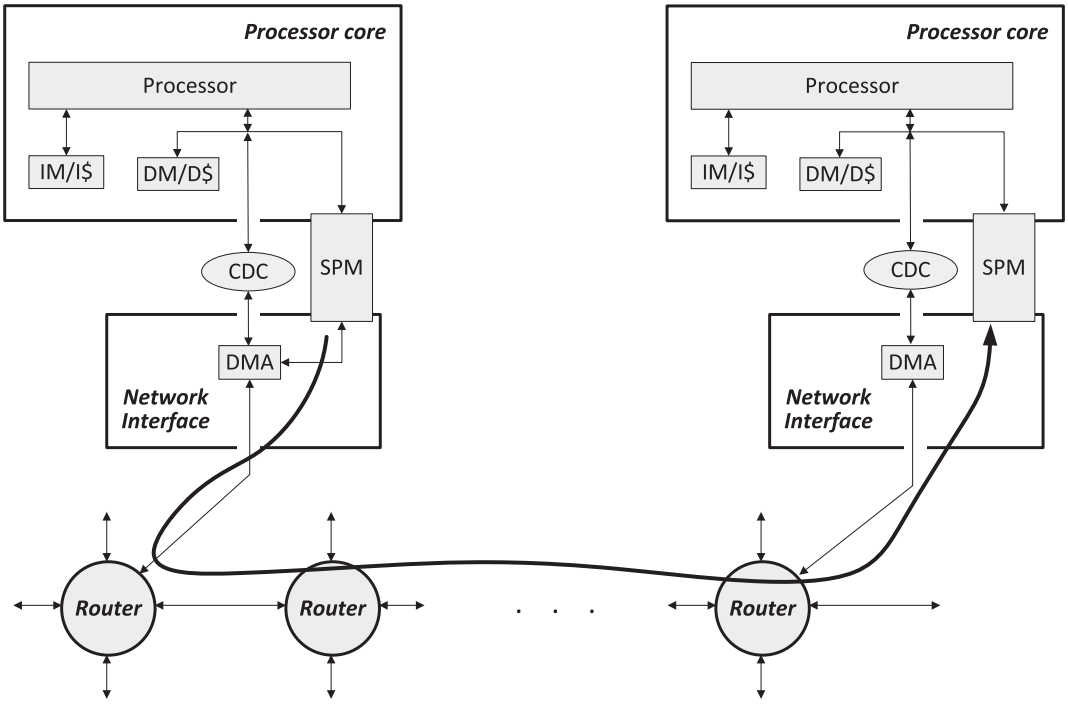


Fig. 4. Data transfer in the T-CREST core-to-core message passing NoC.

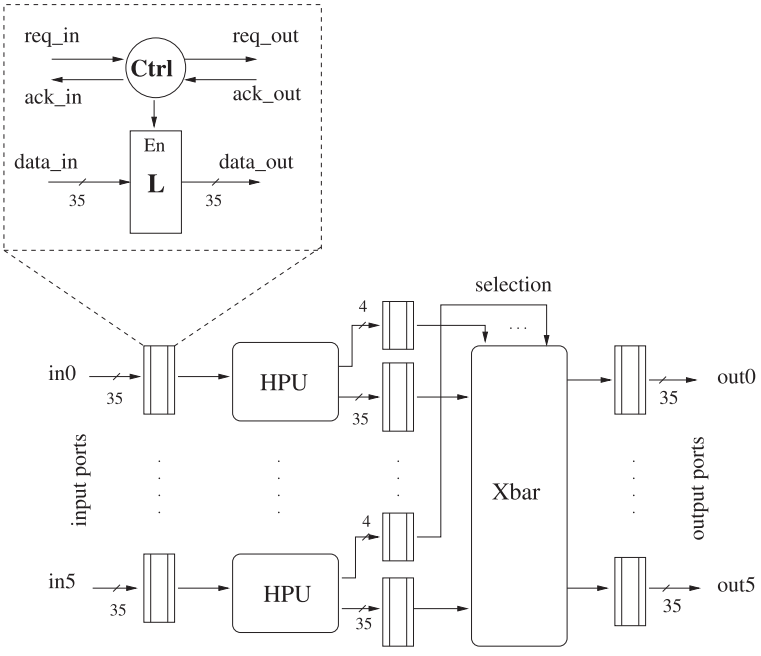


Fig. 5. Block diagram showing the micro-architecture of the asynchronous router.

that define whether the phit is a header phit (start) of a packet (sop) or the end phit of a packet (eop) and whether it is a valid phit or not (vld). The header phit contains the route and the destination write address (wp). The HPU stage in the router decodes the route field of the header phit and provides control signals to the crossbar to direct the packet to the correct output port.

The fact that the NoC allow some skew between the NIs does not affect the WCET analysis of the time it takes to transmit a message across the NoC. As all the NIs operate using a single clock source, analyzing the WCET is basically a matter of counting cycles, and in a statically scheduled TDM-based NoC this is straightforward: The time to transmit a packet is the worst case wait for the first slot reserved, plus the time it takes to transmit the data in a sequence of reserved slots, plus finally the time it takes the last package of the message to traverse the NoC. This is all a matter of counting NI clock cycles. The skew between the source and destination NI adds just a few clock cycles of uncertainty/variation and this is negligible.

Instances of Argo were implemented in ASIC and FPGA technologies. The implementation technology for the ASIC flow is a

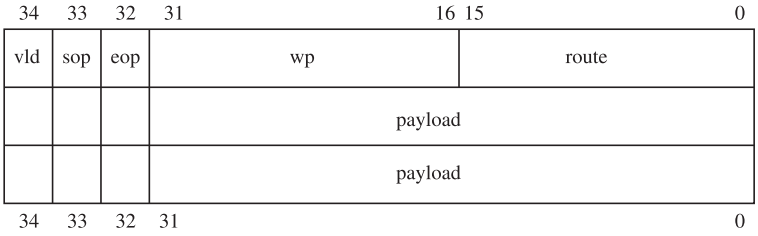


Fig. 6. Packet format of the Argo NoC.

Table 1
ASIC and FPGA area results for a router and an NI of Argo NoC.

	ASIC	FPGA		
		#LUT	#FF	BRAM bits
Router	7965 μm^2	538	580	–
NI	33587 μm^2	457	250	1024

65 nm CMOS standard cell STMicroelectronics library and the target FPGA board is a Xilinx ML605 board. Table 1 shows area results for the components of the NoC; a router and a NI. Detailed results of the implementation of the components appear in [5,100].

The asynchronous design of the routers provides time elasticity. In [6] we analyze a network of such routers and show that it can tolerate several cycles of skew between the NIs. In conclusion the use of asynchronous routers offers more timing elasticity than clocked mesochronous routers and at the same time the hardware area of an asynchronous router is approximately half of the area of a mesochronous clocked router [5,6].

Overall, the entire NoC was implemented in 2x2 and 4x4 instances, which were verified to operate under skew. A 4x4 instance of Argo, with all-to-all communication (240 uni-directional channels) and schedule period of 23 time-slots, implemented in ASIC technology consumes an area of 0.72 mm^2. A similar instance of aelite with 11 NIs, 6 routers and 45 bi-directional channels consumes an area of 2.5 mm^2 [41]. The two instances are not directly comparable but the numbers indicate that Argo is at least 3.5 times smaller than alternative TDM NoCs, since aelite is already a very small NoC.

5. The memory hierarchy

This section presents the memory hierarchy of the T-CREST platform and explains how to provide time-predictable access to off-chip DRAM. The two key requirements for the DRAM sub-system are: (1) it must be a time-predictable architecture and have an analysis that is able to tightly bound the response time of a memory transaction, and (2) it has to scale to a large number of processing elements to fit in a multi-core environment.

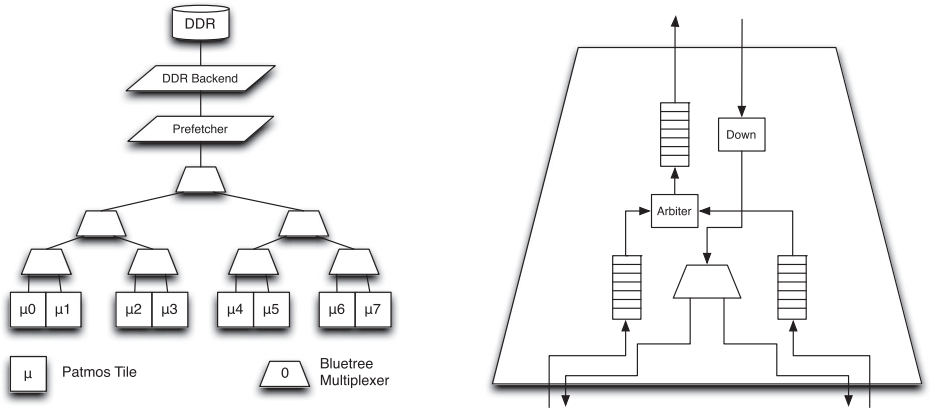
The DRAM sub-system comprises three components, the Bluetree memory tree, a prefetcher, and a dynamically scheduled SDRAM back-end, as shown in Fig. 7a. These components are described in the following sections, starting with Bluetree in Section 5.1, followed by the prefetcher in Section 5.2 and the back-end in Section 5.3. Section 5.4 then presents the WCET of a memory request accessing the DRAM through the memory tree for different numbers of contending processors and compares to the case of a TDM-based memory tree.

5.1. The bluetree memory tree

The Bluetree memory tree [103] is a memory interconnect motivated by the growing bandwidth requirements of modern embedded systems, along with the need to decouple the memory requirements of a task from its communication requirements; a mapping on a Manhattan grid NoC, which allows a set of tasks to meet their communication deadlines, may not allow all high-bandwidth memory clients to meet their requirements, and vice versa.

In addition, conventional monolithic arbiters cannot scale to the requirements of modern systems. Typical arbiters demultiplex the input stream into a number of virtual channels, perform accounting on these channels, and then multiplex those virtual channels back onto the output stream. As the number of virtual channels increases, the complexity and size of the multiplexer/demultiplexer increase, which in turn results in a slower maximum clock speed [104].

We designed the Bluetree memory interconnect to support the memory requirements of modern systems, leaving the TDM-based



(a) Overview of the memory hierarchy. (b) Internals of a memory tree multiplexer.

Fig. 7. The Bluetree memory network-on-chip.

NoC for core-to-core communication only. This tree does not allow for communication between processing nodes, only from processing nodes to main memory. The design distributes memory arbitration across the routers, rather than using a large monolithic arbiter next to memory. This allows the tree to fulfill the scalability requirements of the system, enabling a larger number of requesters at a higher clock frequency than would be available with a single monolithic arbiter.

This tree consists of a set of 2-into-1 full-duplex multiplexers, each with a small arbiter. Fig. 7b shows a block diagram of the internals of one of these multiplexers. Each of these multiplexers contains two input FIFOs, which are then multiplexed onto an output FIFO through a simple arbiter. The downward path contains a single register and is, per definition, non-blocking. The content of this register moves each cycle towards the correct client. We combine these multiplexers into a system such as the one shown in Fig. 7a.

In order to prevent a single core dominating the tree, and to be able to satisfy the requirement that the memory subsystem can be time-predictable, each multiplexer contains a *blocking counter*. This encodes the number of times that a high-priority packet (i.e., a packet from the left) has blocked a low-priority packet (i.e., a packet from the right). When this counter becomes equal to a fixed value m, the counter is reset and a single low-priority packet is given service. This then allows providing an upper bound of the WCET for a memory transaction.

5.2. Prefetcher

Another issue with shared memory is the potentially large worst-case response time for memory transactions (as Table 2 shows). The prefetcher can, to some extent, hide this latency. This is a hardware unit that attempts to speculatively issue memory requests for data, which the processor may require in the near future.

Such a technique is typically not employed within real-time systems due to the unpredictability that such a unit can introduce; a useless prefetch will tie up the memory controller for a period of time and introduce additional blocking for other tasks. If the prefetcher dispatches useless prefetches, and does so without any consideration for any other tasks, it is possible that the prefetcher may actually *harm* the worst-case response time of the system. This is undesirable since the worst-case analysis of the system is then invalidated by the inclusion of the prefetcher.

Instead, we propose a prefetching approach that we can use alongside standard worst-case analysis, to improve the average case while we maintain the required time predictability of the system [7,105]. We make the observation that bandwidth provisioning is typically static, and may not be fully utilized by a task during its whole life-cycle. Typically in these cases, arbiters allow other tasks that have fully utilized their bandwidth bound to request in work-conserving mode. In the context of Bluetree, if a low-priority packet is given service without being blocked by a high-priority packet, or m high-priority packets are given service without any being blocked by a low-priority packet, the arbiter is operating in work-conserving mode.

In these cases, rather than relaying a request in work-conserving mode, the prefetcher can use this slot. The prefetcher fills an empty packet and transmits it to memory. Since, in the context of system analysis, this slot would have been a memory request normally, this approach requires no modification to the system analysis, and allows for prefetching without harming the worst-case execution time.

Another inherent problem with prefetching is that the prefetcher inserts the data into a processor's cache that may displace useful information, thus effectively invalidating any cache analysis that has already taken place to ascertain a worst-case execution time. In order to alleviate this issue, we add a small "prefetch cache" in-between the processor and the first multiplexer. This is a small direct-mapped cache that stores incoming prefetches, and then relays them to the processor when requested. This has the same line size as a Bluetree request (i.e., 16 bytes), and logically requires as many indices as the number of possible prefetch streams, which is eight in this case, thus needs to be of size 128 bytes. In reality, this is slightly larger (i.e., 512 bytes) in order to alleviate any cache locality issues.

Table 2
WCET (cycles @ 300 MHz) of a 32-byte transaction for multiple processors.

Cores	TDM	L6	L5	L4	L3	L2	L1	No blocking
2	108	–	–	–	–	–	90	62
4	164	–	–	–	–	183	99	71
8	276	–	–	–	388	192	108	80
16	500	–	–	817	397	201	117	89
32	948	–	1694	826	406	210	126	98
64	1844	3467	1703	835	415	219	135	107

After a prefetch has taken place and it was useful to the processor, the prefetch cache relays a "prefetch hit" message back to the prefetcher on the same cycle that the processor accesses the data. This is then used to generate another prefetch on behalf of that processor. Since this packet would have been a memory request for the aforementioned data, a prefetch can also be safely transmitted in this case, since the prefetch replaces the memory request that would have taken place had the prefetcher not been operating, and hence the "prefetch hit" packet can also use the slot.

Given this analysis, we use a simple stream prefetcher within this framework. Stream prefetching [106] makes the assumption that if the data at addresses $A, A+1$, and $A+2$ has been requested, it is likely that the data at address $A+3$ will be required in the near future. We can also use other approaches such as stride prefetching [107], Markov prefetching [108], or global history buffer based approaches [109] with this framework without harming the WCET of the system.

5.3. Time-predictable SDRAM back-end

This section presents our time-predictable dynamically scheduled memory controller back-end. The two main contributions of the T-CREST memory controller back-end are: (1) the architecture and the dynamic command scheduling algorithm are time-predictable, (2) the analysis supports different request sizes and memory map configurations, enabling the designer to choose the number of parallel banks to serve a transaction. This allows trade-offs between bandwidth, (worst-case) execution time, and power consumption [110].

The off-chip memory on the Xilinx ML-605 development board, used in the T-CREST project, is a 64-bit DDR3–1066 SDRAM, although the Xilinx PHY only offers a 32-bit interface. The SDRAM back-end interfaces with this memory and is responsible for generating and scheduling commands to access the memory according to the incoming requests in a time-predictable manner, while satisfying the minimum timing constraints between the commands.

We implemented the back-end architecture in VHDL and it has five pipeline stages, as shown in Fig. 8. After a transaction arrives at the interface of the back-end, Stage 1 obtains the relevant information, including the transaction type (read or write), logical address, and the required number of bursts for the given transaction size. Stage 2 splits the transaction into the required number of bursts, and translates the target address to the corresponding physical address (bank, row and column) in the memory. Thereafter, the

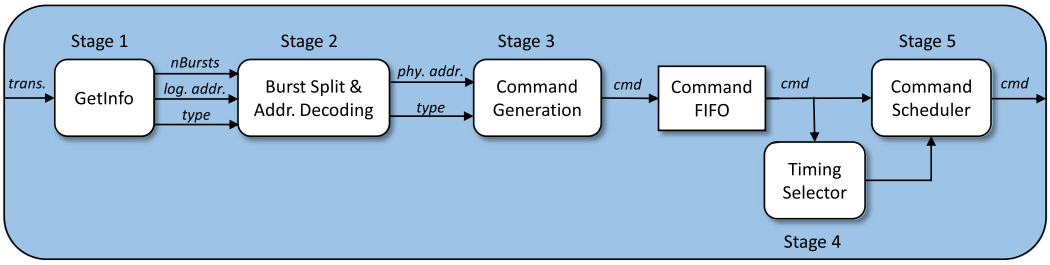


Fig. 8. Control path of the dynamically scheduled back-end.

command generator in Stage 3 produces the required *activate*, *read*, *write*, and *precharge* commands for each burst. To prevent the memory from losing data, *refresh* commands are also generated periodically every 7.8 µs. The generated commands are then inserted into the *command FIFO*. In Stage 4, the *timing selector* is responsible for checking the associated timing constraints for scheduling the command at the head of the command FIFO. The *command scheduler* issues the command in Stage 5 if the timing constraints are satisfied. The action in each stage consumes one clock cycle, except burst splitting in Stage 2, which requires one cycle per burst, and command generation in Stage 3, which takes one cycle per command.

The back-end runs at a frequency of 150 MHz, while the memory itself runs at 300 MHz. This means that the data path of the memory controller works with 128 bit words provided by the memory tree internally, but reads or writes 4×32-bit words per clock cycle to the memory to compensate for the double frequency (150–300 MHz) and the double data rate (two data transfers per 300 MHz clock cycle).

5.4. WCET of a memory transaction

Having presented the concepts and architectures of the memory tree and the memory controller back-end, we now consider the WCET of a single outstanding memory request in the T-CREST platform for a varying number of processors, and hence for different tree depths, and compare to the case of a TDM-based system. Note that it is assumed that application software has been mapped to cores within the architecture prior to WCET calculation.

We report the WCET of a memory transaction in clock cycles at 300 MHz, the frequency of the memory. The results assume a fully balanced tree and a scaling factor of 3 for Bluetree to account for it running at a lower frequency of 100 MHz.

The analysis of the memory controller back-end uses the analytical framework, presented in [111], developed as a part of T-CREST, and adjusted slightly to account for pipeline delays in the FPGA implementation. This analysis framework is general and explicitly models all dependencies between DRAM commands. This enables it to derive the WCET for memory transactions of different sizes in a parameterized way depending on the number of banks used in parallel to serve each transaction, which is determined at design time. Note that refresh is not included in the WCET of memory transactions, but modeled as a high-priority periodic task during schedulability analysis of the system. DDR3 specification gives the period of the refresh as 7.8 µs and our analysis of the timing selector and command scheduler bounds the refresh time.

Given a fixed request size of 32 bytes, the memory controller can accept a new request every 28 cycles. However, the first request in a sequence incurs a pipeline delay of 25 cycles in the worst case, making the WCET of an isolated memory transaction 53 cycles at 300 MHz. Given a blocking factor, we can calculate and bound the number of times the multiplexer can block a request while transiting up the tree. For a fully congested tree (i.e., one in which every client issues a request in every cycle which it can), we can use Eqs. (1) and (2) to calculate the number of times multiplexers may block a packet for a given blocking factor m, when transmitted from a multiplexer at level i. Here, the high priority side is the left-hand side of the multiplexer, and the low priority side is the right-hand side. Additionally, the multiplexer at the root of the tree is level 1, and level n at the leaves of the tree.

$$B_i = \sum_{n=1}^{i} (B'_i - 1) \tag{1}$$

$$B'_i = \begin{cases} 2 & i = 1 \wedge \text{High Priority} \\ m & i = 1 \wedge \text{Low Priority} \\ 2 \times B'_{i-1} & i \neq 1 \wedge \text{High Priority} \\ m \times B'_{i-1} & i \neq 1 \wedge \text{Low Priority} \end{cases} \tag{2}$$

B'_i calculates the *periodicity*, in the worst case, that an input to the multiplexer at level i gains service, and is slightly pessimistic in order to cater to the case where $m = 2$. In this case, the multiplexer at level 1 will gain service once in every two requests to the multiplexer (as there is contention with only one other requestor). This is multiplicative as the blocking also depends on the multiplexers further up the tree (towards memory). When an up-stream multiplexer blocks, it will block an entire *subtree*. In this case, any multiplexers in this subtree will not be able to relay any packets, hence blocking counters will not be updated, and the entire subtree will be stalled. Because of this stalling, the periodicity of lower multiplexers also depends on those further up the tree. We can calculate the blocking occurring at each level of the tree by subtracting one from this periodicity.

B_i then sums the amount of blocking experienced at each level of the tree to ascertain the final blocking figure. We multiply this figure by the worst-case memory delay in order to ascertain the number of cycles that a request can be blocked.

Given these equations, we can use Eq. (3) to calculate the total round trip time for a memory request from the multiplexer at level i. Here, t_{up} is the number of cycles required to transit a multiplexer when traveling to memory, and is six cycles (one to transit the arbiter and be enqueued into the output FIFO, one to leave the output FIFO, then multiplied by the scaling factor). t_{down} is the time taken to transit a multiplexer when traveling downwards, and is three cycles (one cycle to cross each multiplexer, then scaled). t_{mem} is the worst case time for a memory transaction (28 cycles); 25 cycles are the pipeline fill delay; and finally t_{mem} is the execution time of the requested memory transaction.

$$t^i_{mux} = (t_{up} \times i) + (t_{down} \times i) + (B_i \times t_{mem}) + 25 + t_{mem} \tag{3}$$

With this equation we calculate the worst-case execution time of a tree with a varying number of clients (from 2 to 64), and a blocking factor of $m = 2$ and show the result in Table 2. These figures also

detail the worst-case timings given when only varying levels of the tree are fully congested. As an example, the tree for two clients has a single multiplexer. In this case, the figure for *L1* shows the WCET when the multiplexer at level 1 is congested, whereas *No blocking* shows the case where the multiplexer is not congested and thus a request can be immediately transmitted. Similarly, for four processors (with two levels of multiplexers), the figure for *L1* shows the case where only the root level is congested, and the figure for *L2* shows the case where the multiplexer at level 2 and the root multiplexer are fully congested.

The *TDM* case concerns itself with the worst-case response time for a TDM-based arbiter, assuming each TDM slot equals the WCET of a 32 byte request in the back-end (28 clock cycles). This also assumes that each processor has an identical time slice and identical periodicity. This worst case will occur when the client in question issues a request just after its slot has become available. In this case, it will have to wait until its current slot has expired and all other slots have been serviced. It will then have to wait for its own memory request to be serviced. The pipeline delay (25 clock cycles) is factored in also. This implies a total delay therefore of $(28 \times nProcs) - 1 + 28 + 25$ clock cycles.

We can see that, while the worst-case delay is worse than the TDM case for a fully backlogged tree, the tree will typically perform better when it is not fully loaded. This is due to the inherent work-conserving nature of the tree, since the tree operates on blocking factors rather than a static slot table. The timing performance of the tree will therefore depend upon the tasks running within it, and thus there will be a distribution of execution times between the case with no contention and the case where the tree is maximally loaded. In standard TDM, however, the worst-case and the actual case are identical; a task will still need to wait its turn, even if the other turns are unfulfilled.

This worst-case is also not indicative of the actual system. Since Patmos can only have a single outstanding memory request at once, there can only be 64 outstanding memory requests at a time. Since a tree with six levels will have 126 buffers distributed across the inputs to the multiplexers, the tree cannot be maximally loaded (the maximum 64 outstanding requests can only partially fill the 126 buffers). In reality, the worst-case performance will be closer to the performance if all but the last levels of the tree are fully loaded—less pessimistic analysis incorporating this intuition remains an open issue. Finally, we note that in reality systems rarely at run-time experience their worst-case, particularly in terms of WCET [75]. Hence in terms of memory accesses, a constituent part of WCET, actual memory latencies times will be an application dependent distribution between the worst-case in Table 2 and the best (i.e., no blocking).

6. The compiler and WCET analysis

An integral part of the T-CREST design philosophy is to use static (compile-time) alternatives for commonly used performance enhancing features at runtime. This reduces the dynamic behavior and processor states outside the control (and visibility) of the code at the ISA level. In conformance with this philosophy, the compiler must generate code that specifically targets the Patmos ISA and has control over its components.

Besides the hardware architectural means to obtain tight WCET bounds, we pursue a tight integration of the compilation process and timing analysis [26]. On one hand, the compiler preserves information available during the compilation process that usually is discarded, although it would be valuable for automated and precise timing analysis. This includes preserving information about the control-flow structure, but also flow annotations provided by the user that constrain the possible flow of control, e.g., bounds on the maximum number of loop iterations (*loop bounds*). The compiler provides this information as additional input to the WCET tool. On the other hand, results from the timing analysis are fed back to the compiler, to guide optimizations that aim at reducing the guaranteed worst-case performance.

6.1. Support for the Patmos ISA

The architectural design for time predictability (see Section 3) requires dedicated support from the compiler. The absence of dynamic instruction reordering and assignment to the available functional units requires the compiler to create a feasible instruction schedule, which respects instruction latencies, bundles instructions for dual-issue, and fills delay slots of control-flow instructions.

For method cache support, the compiler splits functions that otherwise are too large to fit in the method cache as a whole [112]. For optimization purposes, the compiler also splits large functions with many divergent control-flow paths to avoid loading unused code into the cache. The Patmos ISA allows splitting functions with low overhead.

The explicitly managed stack cache requires the compiler to insert special control instructions, typically at function entries, calls, and returns. The compiler can allocate data objects of fixed size to the stack cache for which it can guarantee the persistence of their stack frame during their lifetime (e.g., register spill slots, local variables with fixed size). At the same time, the compiler manages the *shadow stack* for objects that cannot be allocated in the stack cache. Using the stack cache results in an average runtime speedup of 1.57 for the MiBench benchmarks [3]. Most stack frames allocated on the stack cache never need to be spilled to memory. Furthermore, allocating data on the stack cache reduces the cache pressure on the data cache, leading to a lower data cache miss rates, as shown in [113].

The Patmos ISA exposes the type of memory accessed in typed memory instructions. Hence, the compiler generates different instructions for accessing the main memory through the data cache, the data on the local stack cache, the scratchpad memory, or to bypass the data cache for memory accesses to unpredictable memory locations.

6.2. Single-path code generation

The fully predicated instruction set of Patmos eases single-path code with minimal control-flow complexity and stable execution-time behavior. The compiler can generate single-path code in an automated way, by using a transformation that extends if-conversion techniques. Martin: Ref for if-conversion is missing. The following sentence then starts with fixed loop iterations, which is not if-conversion. Loops are transformed to employ a fixed iteration count and predicates are maintained across function calls [93,114].

Regions of single-path code are specified by entry functions. These functions and the functions called from within are generated to exhibit a singleton execution path. The respective application code is limited to reducible control flow without indirect function calls and recursion. Structured loops are required to specify local loop bounds in the source code, which results in constant loop trip counts.

In single-path code, some control-flow paths are executed sequentially rather than alternatively based on input data. This increases the number of executed instructions and may increase the WCET. Since the method cache fetches whole functions or subfunctions in any case, the total number of instructions fetched does not necessarily increase though. Instead, the amount of code fetched but not executed is typically reduced to zero for single-

path code for most cached code blocks. However, in cases where loops contain code that is not executed in all iterations, the cache may not reach a stable cache state that contains only frequently executed paths if the whole body does not fit into the cache.

6.3. Compilation tool chain

Fig. 9 gives an overview of the compiler tool chain. The Patmos compiler extends the LLVM compiler construction framework [70]. At the beginning of the compilation process, the C frontend `clang` translates each C source code file to LLVM's language- and target-independent intermediate representation called *bitcode*. Bitcode is a control-flow graph oriented representation of the program using static single assignment form. The `llvm-link` tool links the application code with standard and support libraries at bitcode level. This linked application presents subsequent analysis and optimization passes, as well as the code generation backend, with a complete view of the whole program, thus enabling WCET-oriented optimizations.

The `opt` optimizer performs generic, target independent optimizations, such as common sub-expression elimination, constant propagation, etc. The `llc` tool constitutes the backend, translating LLVM bitcode into machine code for the Patmos ISA, and handling the target-specific features for time predictability. The backend produces a relocatable ELF binary containing symbolic address information, which `gold` processes,[6] defining the final data and memory layout, and resolving symbol relocations.

In addition to the machine code, the backend exports supplementary information for WCET analysis and optimization purposes in form of a *Program Metainfo File*. This information contains, among others, flow information (in form of loop bounds provided by symbolic analysis on bitcode level), structural information (targets of indirect branches), information on memory accesses (memory areas accessed by individual load/store instructions), as well as information to relate bitcode and machine code program representations, as detailed in Section 6.5. The `platin` toolkit enhances (e.g., by a hardware model), processes, and translates this information to the input format for annotations of AbsInt's timing analysis tool aiT [82], which uses this information in addition to the ELF binary to compute tight WCET bounds.

6.4. The WCET analyzer aiT

The aiT tool [82,83] determines safe and precise upper bounds for the worst-case execution times of non-interrupted tasks in real-time systems. Separate versions of aiT offer support for different processor architectures. Within T-CREST, we extended aiT to support the Patmos architecture, which allows us to analyze the WCET of tasks from Patmos executables.

aiT takes as input a binary executable and an AIS file, i.e., an annotation file in the proprietary AIS format that provides further information about the program. After reconstructing the control flow from the executable, aiT performs a loop analysis to automatically compute upper bounds of loop iterations, and a value analysis to determine safe approximations of the values of processor registers and memory cells for every program point and execution context. The user can extend and refine the results of these automatic analyses in the AIS file.

An architectural analysis simulates the execution behavior of the input program through an abstract hardware model. The analysis determines lower and upper bounds for the execution times of basic blocks by performing an abstract interpretation of the program execution on the particular architecture, taking into account its pipeline, caches, memory buses, and attached peripheral devices. Using the results of these analyses, the path analysis phase searches for the longest execution path and from it, derives a safe estimate for the WCET.

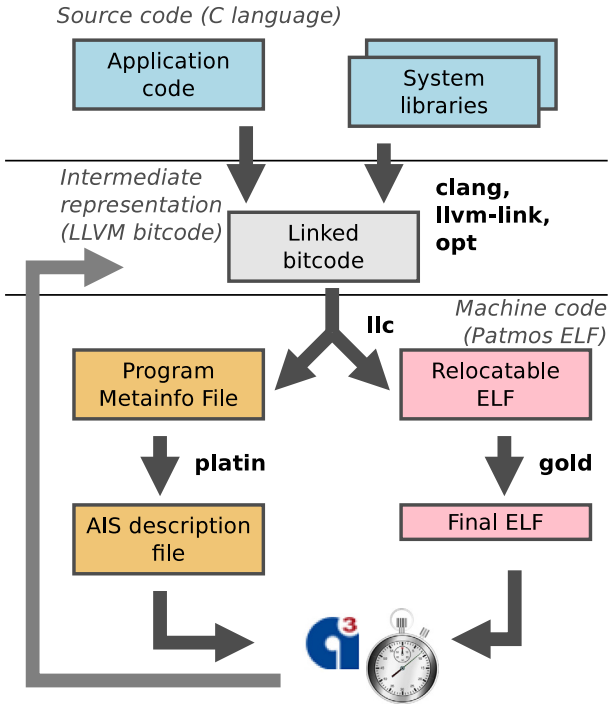


Fig. 9. Compilation tool chain overview.

6.5. Compiler and WCET analysis integration

Due to the considerable changes that backend code generation involves in LLVM, bitcode is not a suitable target for WCET analysis. On the other hand, when the compiler lowers bitcode to machine code it loses a large amount of the compiler's analysis information.

Support for WCET analysis integration requires that the compiler maintains semantic information between program representation levels, passes information about machine code to the WCET analysis tool, and has control over the performed optimizations and machine code generation [115].

6.5.1. Preservation of meta-information

Due to the complexity of modern compilers and their optimizations, transforming information from the source level to the machine-code level is not trivial to retrofit into an existing industrial-quality framework. The compilation flow described in Section 6.3 permits to use different strategies for the preservation of meta-information in different stages of compilation:

- The translation from source code to the platform-independent intermediate representation by `clang` includes translation of information available only at the source-level (e.g., annotations in form of pragmas) to bitcode meta-information. In order to separate concerns, `clang` performs no optimizations at this stage.
- The compiler performs high-level optimizations on bitcode. Some of the available optimizations perform major structural changes to the program (e.g., loop unswitching or loop unrolling). Consequently, we extended these optimizations to preserve meta-information relevant for timing analysis. In particular, techniques for maintaining loop bounds, which are

[6] gold is part of the GNU binutils, see http://sourceware.org/binutils/.

crucial for WCET analysis, require considerable additional effort for each optimization [71]. However, as these optimizations are implemented in a platform-independent way, it is likely that the investments on preserving meta-information pay off.
- In order to preserve meta-information in the compiler backend, the compiler maintains relations between basic blocks (and memory accesses) at the bitcode and the machine code level. Based on this information, we derive *control flow relation graphs* that model regular relations between execution paths at both levels. The compiler uses these control flow relation graphs to transforms flow facts from bitcode to machine code [116]. Thus it is not necessary to add dedicated support for flow-fact updates in the backend.
- The compiler and linker generate and store in the binary all symbolic names necessary to specify information for the timing analysis tool. This permits to specify all information about machine code by referring to symbolic addresses.

6.5.2. Exchange of program and timing information

Compiler passes and our integration toolkit `platin` integrate the LLVM compiler and AbsInt's WCET analysis tool aiT [82]. We adapted the compiler to export information on both bitcode and machine code, and on the relation between these representations. This compiler pass is largely platform independent. The compiler exports the program information after it performed all optimizations. This way, a particular target backend only has to provide target-specific information. Our `platin` (Portable LLVM-based Annotation and Timing Analysis Integration) tool uses this information. The main responsibility of `platin` is to consolidate and transform information about the program, its control flow and its timing behavior. `platin` collects this information from the compiler, development and analysis tools.

At the core of the information exchange strategy is the *Program Metainfo Language* (PML) file format that stores information relevant for WCET analysis and compiler optimizations. We designed PML to allow information exchange with different tools at both the bitcode and machine code level. Fundamental concepts such as control-flow graphs, loop nest trees, or linear flow constraints are thus defined in a way that is applicable to both bitcode and machine code. The relation between optimized bitcode and machine code is also stored, which allows transforming information obtained from auxiliary analysis tools that operate at the bitcode level to machine code level for use by aiT. An example for a particularly useful analysis on bitcode level is loop trip count analysis, already available in the LLVM framework. At the machine code level, the PML format is largely platform independent. To achieve platform independence, `platin` specifies machine code related concepts like jump tables in a uniform way.

For timing analysis with the aiT tool, which carries out its analysis on binary code, an AIS annotation file is exported from the PML file, which in conjunction with the executable, serves as input to the analyzer. The automatically generated annotations provide information on jump tables and indirect calls, potential targets of memory accesses as well as loop bounds and flow frequency constraints. As the annotation language may only refer to instructions at the binary level, flow constraints that refer to control-flow edges or empty basic blocks (for example) are not directly expressible in the AIS format. Therefore, we reuse the technique developed for flow-fact transformations between bitcode and machine code level, and reformulate those program entities that are not supported by the AIS format.

When the WCET analysis is complete, `platin` merges back the analysis results into the PML database. This information is available for the compiler to guide further optimizations for improving time predictability and worst-case performance. For example, the compiler might bypass the cache for those memory accesses that the timing analysis tool classifies as unpredictable. Evaluation of this WCET analysis based optimization is future work.

To obtain a profile regarding a program's statically analyzed worst-case behavior, `platin` makes use of the criticality metric [117]. It assigns to each basic block a value in the range $[0, 1]$ that signifies its importance relative to the WCET bound. E.g., all blocks on the worst-case execution path have criticality of 1, while infeasible code has criticality 0. The resulting profile information gives a complete view of the program instead of the singular worst-case path result that static WCET analysis tools usually yield. Similar to an execution profile obtained for improving average-case performance, the programmer and compiler can base their optimization decisions on the criticality of a piece of code.

7. Evaluation

To evaluate the T-CREST platform prototype, we use industrial use cases derived from real-world applications. We build these use cases upon domain-specific applications from the avionics and railway domains. Their purpose is to evaluate and validate the T-CREST platform as a whole, complementing the validation and verification activities realized, individually, over the technological elements of the platform. Therefore, in this section we discuss the evaluation from a platform point-of-view where the subject of the evaluation is the complete T-CREST platform.

All use cases were adapted to exploit the specific features provided by the platform. To provide better coverage of these features and to address the different industrial usage models, the avionics use cases explored achieving a higher degree of system integration than currently available in commercial platforms, whereas the railway use case focused on parallelization. Accordingly, the most complex avionics use case deploys three independent and unrelated applications over the platform, validating that their temporal behavior is unaffected. The most complex railway use case exploits the multi-core characteristics of the T-CREST platform to improve the performance of a Fast Fourier Transformation based application by parallelizing it.

Once the applications were ported, the first validation of the platform was to verify if they could comply with their original requirements and execute according with the expected behavior. This was the case for all the use cases from avionics and railway. Next, the behavior of the platform with regard to the WCET was assessed. The WCET for selected tasks was estimated using AbsInt's WCET analysis tool aiT.

aiT enables us to obtain WCET estimations for each individual application in a single or multi-core configuration. Multi-core WCET bounds are obtained by configuring the settings for latencies for reads and writes to memory. Apart from the memory latency, the application executable, an AIS annotations file, and the analysis entry point are the sole inputs required for the WCET analysis.

7.1. Avionics use case

The avionics use cases consist of a set of avionics applications that are typically hosted on one single computing platform as it is common practice in on-board systems integrated according to the principles of Integrated Modular Avionics (IMA). IMA is an architectural concept, originated in aeronautical systems, that enables multiple unrelated applications, with different criticalities, to share the same computing platform without interference by applying robust partitioning. The application of IMA concepts resulted in a reduction of hardware used in aircrafts by enabling resource sharing among different applications. In consequence, IMA reduced the main drivers of aircraft operational costs, in particular weight, volume, and energy consumption. Any hardware

platform applicable to avionics systems shall support the key elements of the IMA concept. More specifically, it shall support application independence.

The avionics evaluation uses three distinct applications: AOC, CAS, and IOP. The AOC (Airlines Operational Centre) is the on-board part of an Air Traffic Management (ATM) system that enables digital text communication between the aircrew and ground ATM units. Skysoft (today GMV Portugal) in cooperation with BAE Systems developed the application according to DO-178B Design Assurance Level (DAL) C. The AOC can be described as a communication router and message database. It stores reports sent from ground stations or created by aircraft subsystems, such as weather reports, trajectories and route planning information. The AOC schedules these reports for delivery or sends them to the destination on demand.

In addition, the Crew Alert System (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). The CAS improves situational awareness by allowing the aircrew to view complex information in a graphical format and also by alerting the crew to unusual or hazardous situations. CAS is an ARINC 653 prototype application whose development followed DO-178B guidance for DAL A.

The last application, the I/O Partition (IOP), is an IMA application that acts as a router mediating the access from other application partitions to data buses and avionics networks in a robust and safe manner. The IOP interfaces other partitions through ARINC 653 queuing and sampling ports. The user associates each port with a set of routing configurations that enable data traversing via this port to be forwarded to a given physical I/O interface and an address in that interface. In consequence, access to a network becomes transparent to application partitions. The application only sends and receives data from a typical queuing or sampling port. The IOP handles all routing configurations.

All applications, originally, 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, we mapped the port interfaces, used by the demonstrators, to the inter-core communication that the configurable NoC provides.

To provide a run-time environment to the avionics applications, we ported the RTEMS real-time operating system (RTOS) to the T-CREST platform. RTEMS is a free and open source RTOS, used as a baseline for dozens of space missions, that is compatible with open standards such as POSIX or iTRON [118]. RTEMS was ported in a single-core configuration. When supported by a second stage bootloader, also developed as part of the evaluation process, it enables the deployment of Asymmetric Multiprocessing (AMP) configurations over the T-CREST platform where an operating system instance is present in each one of the platform cores.

7.2. Railway use case

The T-CREST methodology is also evaluated with a use case from the railway domain. In current railway systems, especially in urban areas, analog technologies that limit performance, require constant maintenance, and infer high management costs are still widespread. It is therefore required to shift towards optimized innovative solutions in terms of hardware and software. The goal is to enhance safety and efficiency, and at the same time allow a reduction of the installation costs.

The railway use case is an adaptation of the GSM-R Integrity Detection System (GRIDES) to the T-CREST platform. GSM-R is an extension of the commercial GSM standard with additional railway specific services. Acquiring and analyzing the GSM-R radio signal, within the allocated bandwidth (for both, the uplink and downlink channels) and in proximity of one or more railway lines, enables the assessment of the link's health. The GRIDES system consists of a network of intelligent diagnostic units that measure the quality of the GSM-R radio link along railway tracks. It is necessary to monitor the radio link status continuously. Therefore, we need to know the WCET of the tasks to guarantee the continuous operation. The number of radio channels and the complexity of the radio interference detection algorithm require the use of high-performance systems. To be able to extend the area that GRIDES can monitor, cost-effective architectures are desirable. In this context the T-CREST project is well suited. The railway domain specific use case evaluates the T-CREST platform with the goal to verify its adaptability to this domain's rules, and to gain a better understanding of its possibilities. The complexity of the process requires timing/performance optimization and the research of new strategies and tools for automation. The automatic derivation of execution time bounds is a promising strategy. Therefore, the verification of the WCET-oriented T-CREST platform tools is interesting for industry.

7.3. Evaluation results of the avionics use case

The main objective of the avionics evaluation was to demonstrate that, given a configuration of the T-CREST platform, it is possible to independently obtain the WCET of any application, regardless of other software executing on the platform. This temporal independence between applications is a cornerstone in the IMA concept, being difficult to obtain in current multi-core systems. The lack of analyzable multi-core platforms hampers their adoption in the aerospace market, preventing the potential benefits in terms of higher system integration that could lead to improved and cost-efficient avionics systems.

In order to validate application independence, a demonstrator was setup where each core hosts a different application (AOC, IOP, CAS) that would, in a typical IMA system, be a standalone partition. Fig. 10 shows this situation, representing a potential deployment where several distinct avionics systems are integrated over the same multi-core platform. This distribution of applications on cores in an asymmetric fashion was used to validate that the timing of each application depends solely on its own execution and the hardware configuration.

To provide further insight over the behavior of the T-CREST platform, several test cases were setup by varying the number, configuration, and distribution of the avionics applications. For the different test cases, we estimated the WCET of selected tasks and, in some cases, compared against a measurement of the average case execution time of those same tasks. The measured execution time is obtained by reading the cycle accurate timer of Patmos at specific points in the source code.

The following set of tables shows the WCET results of the avionics use cases. Each table belongs to a single use case application and a corresponding function that was selected as the target of the WCET analysis. The results are shown per test case, and were obtained using the latency values for 75 MHz, as this is the processor frequency of our evaluation platform.

As presented in the Tables 3 and 4, we were able to obtain WCET estimations for every avionics application. Table 3 shows that the main factor influencing the WCET estimation is the number of cores available in the platform. Whereas variations in concurrently executing applications, here appearing as different test cases, have no noticeable impact on the WCET estimation. The worst-case execution time estimation for the `cas_loop` entry

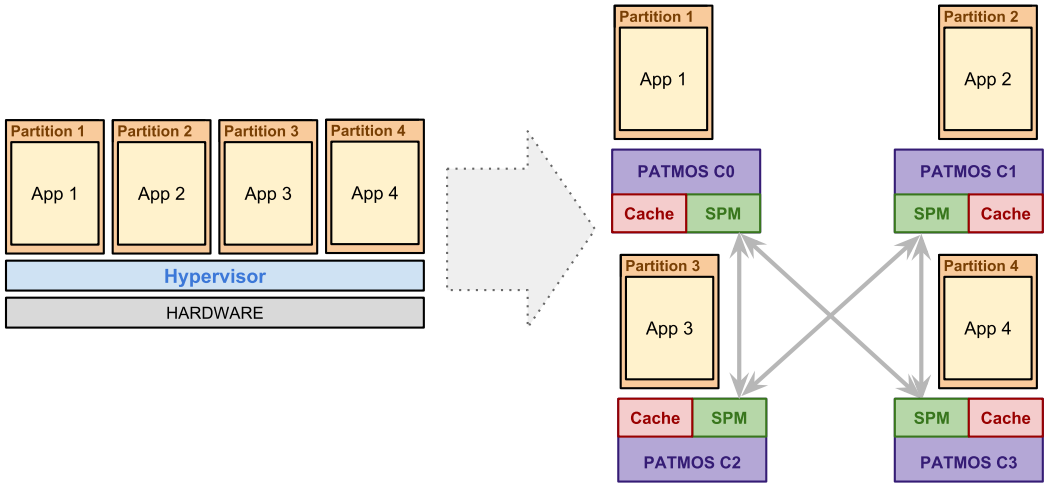


Fig. 10. Mapping of ARINC 653 partitions to cores onto the T-CREST platform.

Table 3
WCET results for selected tasks of avionics use cases.

Analysis entry	Test case	Cores	WCET estimation (in ms)
`cas_loop`	CAS + IOP	4	284
	AOC + CAS + IOP	9	619
`AGPAOCReceiveMainLoop`	AOC + IOP	9	5.41
	AOC + CAS + IOP	9	5.45
`AirplanePAOCMainTask`	AOC + IOP	9	1.92
	AOC + CAS + IOP	9	1.94
`decoderLoop`	AOC + IOP	9	210
	AOC + CAS + IOP	9	210
`AOCAlertMainLoop`	AOC + IOP	9	2.72
	AOC + CAS + IOP	9	2.74

Table 4
WCET estimations and average-case execution time measurements for IOP: pre_dispatcher and pos_router entry points.

Analysis entry	Test case	Cores	WCET estimation (in ms)	Timing measurement (in ms)
`pre_dispatcher`	CAS + IOP	4	6.24	0.952
	AOC + IOP	9	14.64	0.239
	AOC + CAS + IOP	9	23.57	1.110
`pre_router`	CAS + IOP	4	6.42	0.121
	AOC + IOP	9	15.35	0.120
	AOC + CAS + IOP	9	27.07	0.151

Table 5
Comparison of WCET results between Patmos and LEON for the IOP application.

Analysis entry	Cores	Target CPU	WCET estimation (in ms)
`pre_dispatcher`	1	Patmos	2.20
		LEON	2.32
	4	Patmos	5.75
		LEON	45.28
`pre_router`	1	Patmos	2.21
		LEON	2.15
	4	Patmos	6.06
		LEON	41.81

point displays a twofold increase when changed to a nine-core platform from a quad-core platform.

Some applications, listed on Table 3, exhibit a very small variance in the WCET estimation for different test cases while using the same number of cores. This small variation is due to different configurations being used in different test cases. Some of these configurations (e.g., number of ARINC 653 ports) have a small impact over loop bounds and, hence, over the estimated worst-case execution time. This effect is also present on the WCET estimation results from the IOP application presented in Table 4. All WCET estimations are, as expected, higher than the average case measurements. However, the rather big difference between WCET estimates and measurements has no real meaning, as average case measurements probably do not trigger the worst-case execution path. Martin: This overestimation is high. Why? I hope the reviewers will not kill us for it.

To further improve the T-CREST evaluation process we decided to setup a simple comparison between the T-CREST platform and a SPARC/LEON processor. LEON processors are very common in real-time systems, especially in the space domain. We selected the IOP application to compare the T-CREST platform against a LEON 3 processor, as it was originally a LEON application.

Alongside the compiled IOP executable for LEON 3, a manually composed AIS annotations file is used as input to AbsInt's WCET analysis tool in order to obtain the estimated WCET values for the LEON processor. These values are then compared with the values obtained using the same source code compiled for the T-CREST platform. However, this comparison is solely feasible for single core as it is not possible to determine WCET for multi-core configurations of the LEON processor (the problem is unbounded).

Nonetheless, an approximate value of the expectable WCET, for multi-core LEON, is estimated by factoring single-core WCET values with a maximum interference multiplier representative of a LEON multi-core processor. This maximum interference multiplier is extracted from the literature [119], namely an European Space Agency funded study aimed at characterizing the NGMP processor (quad-core LEON 4). In this study, it was found that inter-core interference could increase the execution time of a given code segment up to twenty times compared to its single-core value.

Table 5 presents the comparison between LEON 3 and T-CREST/Patmos. From the comparison in Table 5 we can conclude that, in single-core configurations, Patmos and LEON have similar results with one alternately exhibiting a marginal reduction (<5 %) in the WCET bound over the other depending on the specific analysis entry point being used.

Table 6
WCET of tasks from the avionics use case (in clock cycles).

Task	Stack cache size (bytes)			D-Cache only
	128	256	512	
AGP_ClientMainLoop	8747009	8747009	8747009	8817249
Airplane_PAOCMainTask	3710092	3710092	3710092	4539013
AOC_decoderLoop	2677263042	2677202454	2677202454	2233634812
AOC_feederLoop	1403864	1403864	1403864	1411845
AOC_replierLoop	6242675	6242675	6242675	6753912
Pilot_MDCUMainLoop	6082657125	6082657125	6082657125	5986097713
CAS_loop	6225437	6225437	6225437	6670233
IOP_grbc_manager	1041521886	1041521886	1041521886	851933301
dry2_1	553319	553319	553319	577039

Table 7
Worst-case execution time from aiT analysis for Bluetree and TDM in GRIDES.

Cores	Bluetree WCET (in s)	TDM WCET (in s)
1	322	322
3	225	180
15	102	56

In multi-core configurations, the difference is more significant; Patmos, as part of the T-CREST platform, can be directly targeted by static WCET analysis techniques. Such analysis is unfeasible in multi-core versions of the LEON processor, like the NGMP. Being analyzable in terms of WCET behavior offers the T-CREST platform a key advantage over the LEON multi-core processor. When comparing the WCET values obtained in multi-core T-CREST with those empirically estimated for the LEON, we can see the T-CREST platform yielding a seven times lower WCET bound. Nonetheless, the LEON values presented cannot be used to build a safety case around the software because they are rough estimates derived from empirically obtained interference patterns.

Another element assessed early on the avionics evaluation process was the stack cache, since it is a T-CREST innovation that directly impacts our applications. We compared the WCET estimations from selected tasks of the Avionics applications while varying the size of the stack cache and its presence. Table 6 presents these WCET estimations. The T-CREST system was configured with a burst length of 32 words and a cache line size of 128 bytes.

From Table 6 we can conclude that the avionics tasks analyzed make a shallow use of the stack, since the increase of the stack cache size results in a very small improvement in terms of WCET estimations. The presence of the stack cache resulted in an improvement in terms of worst-case performance for two-thirds of the analyzed tasks over the configuration with data cache only. The remaining tasks do not benefit from the stack cache due to their cache access patterns.

7.4. Evaluation results of the railway use case

Program parallelization was investigated to increase the performance of an application executing on the T-CREST platform. Three railway use cases are used to evaluate the platform.

The first use case provides the preliminary porting of the GRIDES project to the T-CREST platform with changes related to the platform to, but without multi-core adaptations. Only one Patmos core was used. In the second scenario three Patmos cores were used. A true parallel management of the Uplink and Downlink channel groups was used. The aim of the third test was to evaluate the improvements provided by the Patmos multi-core architecture. To perform this test, the GRIDES architecture has been redesigned to use the multi-core architecture and to group Uplink and Downlink channels to use 15 cores.

Table 7 shows WCET estimations for the GRIDES application executing on different numbers of cores. The table shows WCET numbers for the Bluetree memory arbiter and a TDM based memory arbiter. The table shows better performance for the application running on the multi-core architecture compared to the single core version. The achieved improvement of the tri-core version over the single-core version is 1.78 times, while the improvement achieved with the 15-core version over the single-core version is 5.67.

7.5. Access to open-source components

Many of the components developed within T-CREST are available in open source, most of them in the industry friendly BSD license. The sources are hosted at GitHub and the reader can find the sources of T-CREST at:

- https://github.com/t-crest

Further information on the project is available at:

- Official project web site http://www.t-crest.org
- Processor and compiler web site: http://patmos.compute.dtu.dk/
- For questions and discussions join the Patmos mailing list: https://groups.yahoo.com/group/patmos-processor/

8. Conclusion

The T-CREST project provides a time-predictable multi-core architecture for future hard real-time systems. Within T-CREST we provide time-predictable hardware: the Patmos processor, the Argo network-on-chip, the Bluetree memory tree, and an SDRAM memory controller. A WCET optimizing compiler built from the LLVM compiler framework supports the processor. We integrated the compiler and the WCET analysis tool aiT that supports Patmos.

We evaluated the T-CREST platform with real-time applications from the avionics and railway domains. An application from the avionic domain demonstrates that tasks executing on different cores do not interfere with respect to their WCET. A signal processing application from the railway domain shows that the WCET can be reduced for computation-intensive tasks when distributing the tasks on several cores.

Most of the technology of T-CREST is available in open source and we consider it as a platform for further research on time-predictable architectures.

Acknowledgment

This work was partially funded by the European Union's 7th Framework Programme under grant agreement No. 288008: Time-predictable Multi-Core Architecture for Embedded Systems

(T-CREST) and the EU COST Action IC1202: Timing Analysis on Code Level (TACLe).

References

[1] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C.W. Probst, S. Karlsson, 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, 2011, pp. 11–20.

[2] P. Degasperi, S. Hepp, W. Puffitsch, 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), IEEE, Reno, Nevada, USA, 2014.

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

[4] M. Schoeberl, F. Brandner, J. Spars, E. Kasapaki, 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), IEEE, Lyngby, Denmark, 2012, pp. 152–160.

[5] E. Kasapaki, J. Sparso, R.B. Sorensen, K. Goossens, Router designs for an asynchronous time-division-multiplexed network-on-chip, in: Digital System Design (DSD), 2013 Euromicro Conference on, IEEE, 2013, pp. 319–326.

[6] E. Kasapaki, J. Spars, Argo: a time-elastic time-division-multiplexed NOC using asynchronous routers, in: Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), IEEE Computer Society Press, 2014, pp. 45–52.

[7] J. Garside, N.C. Audsley, Investigating shared memory tree prefetching within multimedia NOC architectures, in: Memory Architecture and Organisation Workshop, 2013.

[8] B. Akesson, K. Goossens, M. Ringhofer, 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, ACM, New York, NY, USA, 2007, pp. 251–256.

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

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

[11] S. Dighe, S. Vangal, N. Borkar, S. Borkar, Lessons learned from the 80-core tera-scale research processor, Intel Technol. J. 13 (4) (2009) 119–130.

[12] T.G. Mattson, R.F. Van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, S. Dighe, The 48-core SCC processor: the programmer's view, in: International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2010, pp. 1–11.

[13] R. Kumar, T.G. Mattson, G. Pokam, R. Van Der Wijngaart, The case for message passing on many-core chips, in: M. Hübner, J. Becker (Eds.), Multiprocessor System-on-chip: Hardware Design and Tool Integration, Springer, 2011, pp. 115–123. Ch. 5.

[14] M. Schoeberl, F. Brandner, S. Hepp, W. Puffitsch, D. Prokesch, Patmos Reference Handbook, Technical Report, 2014.

[15] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, J. Wolf, Merasa: Multi-core execution of hard real-time applications supporting analysability, Micro IEEE 30 (5) (2010) 66–75.

[16] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Casse, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quinones, M. Panic, J. Abella, F. Cazorla, S. Uhrig, M. Rohde, A. Pyka, parMERASA – multi-core execution of parallelised hard real-time applications supporting analysability, in: 2013 Euromicro Conference on Digital System Design (DSD), 2013, pp. 363–370.

[17] M. Schoeberl, A Java processor architecture for embedded real-time systems, J. Syst. Archit. 54 (1–2) (2008) 265–286.

[18] C. Pitter, M. Schoeberl, A real-time Java chip-multiprocessor, ACM Trans. Embedded Comput. Syst. 10 (1) (2010). 9:1–34.

[19] L. Thiele, R. Wilhelm, Design for timing predictability, Real-Time Syst. 28 (2–3) (2004) 157–177.

[20] J. Gustafsson, B. Lisper, M. Schordan, C. Ferdinand, M. Jersak, G. Bernat, ALL-TIMES – a European project on integrating timing technology, in: T. Margaria, B. Steffen (Eds.), Proc. Third International Symposium on Leveraging Applications of Formal Methods (ISOLA'08), Springer, 2008, pp. 445–459.

[21] K. Goossens, A. Azevedo, K. Chandrasekar, M.D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. Beyranvand Nejad, A. Nelson, S. Sinha, Virtual execution platforms for mixed-time-criticality systems: the CompSOC architecture and design flow, ACM SIGBED Rev. 10 (3) (2013) 23–34.

[22] S. Goossens, B. Akesson, M. Koedam, A. Beyranvand Nejad, A. Nelson, K. Goossens, The CompSOC design flow for virtual execution platforms, in: Proceedings of the 10th FPGAworld Conference, ACM, New York, NY, USA, 2013, pp. 7:1–7:6.

[23] M. Schoeberl, Time-predictable cache organization, in: Proceedings of the First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009), IEEE Computer Society, Tokyo, Japan, 2009, pp. 11–16.

[24] M. Schoeberl, 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), Vol. 3292 of LNCS, Springer, Agia Napa, Cyprus, 2004, pp. 371–382.

[25] B. Huber, W. Puffitsch, M. Schoeberl, Worst-case execution time analysis driven object cache design, Concurrency Comput.: Pract. Exp. 24 (8) (2012) 753–771.

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

[27] B. Huber, S. Hepp, M. Schoeberl, Scope-based instruction cache analysis, in: Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014), 2014.

[28] S.A. Edwards, E.A. Lee, The case for the precision timed (PRET) machine, in: Proceedings of the 44th Annual Conference on Design Automation, ACM, New York, NY, USA, 2007, pp. 264–265.

[29] B. Lickly, I. Liu, S. Kim, H.D. Patel, S.A. Edwards, E.A. Lee, Predictable programming on a precision timed architecture, in: E.R. Altman (Ed.), Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008), ACM, Atlanta, GA, USA, 2008, pp. 137–146.

[30] I. Liu, J. Reineke, D. Broman, M. Zimmer, E.A. Lee, A PRET microarchitecture implementation with repeatable timing and competitive performance, in: Proceedings of IEEE International Conference on Computer Design (ICCD 2012), 2012.

[31] I. Liu, Precision Timed Machines, Ph.D. Thesis, EECS Department, University of California, Berkeley, 2012.

[32] I. Liu, J. Reineke, E.A. Lee, A PRET architecture supporting concurrent programs with composable timing properties, in: Signals, Systems and Computers, 2010 Conference Record of the Forty-Four Asilomar Conference on, 2010.

[33] M. Zimmer, D. Broman, C. Shaver, E.A. Lee, FlexPRET: A processor platform for mixed-criticality systems, in: Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS), Berlin, Germany, 2014.

[34] A. Waterman, Y. Lee, D.A. Patterson, K. Asanovic, The Risc-v Instruction Set Manual, Volume I: Base User-level isa, Technical Report, UCB/EECS-2011-62, EECS Department, University of California, Berkeley, 2011.

[35] M. Schoeberl, Time-predictable computer architecture, EURASIP J. Embedded Syst. 2009 (2009) 17 (Article ID 758480).

[36] M. Schoeberl, Is time predictability quantifiable? in: International Conference on Embedded Computer Systems (SAMOS 2012), IEEE, Samos, Greece, 2012.

[37] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, F.J. Cazorla, Assessing the suitability of the ngmp multi-core processor in the space domain, in: Embedded Software (EMSOFT), ACM, Tampere, Finland, 2012, pp. 175–184.

[38] D. Wiklund, D. Liu, SoCBUS: Switched network on chip for hard real time embedded systems, in: Proc. IEEE International Parallel and Distributed Processing Symposium, IPDPS 2003, IEEE Computer Society, 2003. p. 798a.

[39] P.T. Wolkotte, G. Smit, G. Rauwerda, L. Smit, An energy-efficient reconfigurable circuit-switched network-on-chip, in: Proc. 19th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2005, 2005, p. 155a.

[40] K. Goossens, J. Dielissen, A. Rădulescu, The thereal network on chip: concepts, architectures, and implementations, IEEE Design Test Comput. 22 (5) (2005) 414–421.

[41] A. Hansson, K. Goossens, On-chip Interconnect with Aelite/Composable and Predictable Systems, Springer, 2011.

[42] M. Millberg, E. Nilsson, R. Thid, A. Jantsch, Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip, in: Proc. Design, Automation and Test in Europe (DATE), IEEE Computer Society Press, 2004, pp. 890–895.

[43] M. Schoeberl, A time-triggered network-on-chip, in: International Conference on Field-Programmable Logic and its Applications (FPL 2007), IEEE, Amsterdam, Netherlands, 2007, pp. 377–382.

[44] C. Paukovits, H. Kopetz, Concepts of switching in the time-triggered network-on-chip, in: Proc. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2008, pp. 120–129.

[45] T. Bjerregaard, J. Spars, A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip, in: Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), IEEE Computer Society Press, 2005, pp. 34–43.

[46] M. Harrand, Y. Durand, Network on Chip with Quality of Service, US Patent 8,619,622, 2013.

[47] S. Zheng, A. Burns, L.S. Indrusiak, Schedulability analysis for real time on-chip communication with wormhole switching, Int. J. Embedded Real-Time Commun. Syst. (IJERTCS) (2010) 1–22.

[48] L.S. Indrusiak, End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration, J. Syst. Archit. 60 (7) (2014) 553–561.

[49] J.-Y. Le Boudec, Application of network calculus to guaranteed service networks, IEEE Trans. Inf. Theory 44 (3) (1998) 1087–1096.

[50] M. Bakhouya, S. Suboh, J. Gaber, T. El-Ghazawi, Analytical modeling and evaluation of on-chip interconnects using network calculus, in: Proc. ACM/IEEE International Symposium on Networks-on-Chip (NOCS), 2009, pp. 74–79.

[51] M.D. Gomony, J. Garside, B. Akesson, N. Audsley, K. Goossens, A generic, scalable and globally arbitrated memory tree for shared dram access in real-time systems, in: Proceedings 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, p. To appear.

[52] S.V. Tota, M.R. Casu, M.R. Roch, L. Rostagno, M. Zamboni, MEDEA: a hybrid shared-memory/message-passing multiprocessor NoC-based architecture, in: 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), IEEE, 2010, pp. 45–50.

[53] A. Agarwal, The Tile Processor: A 64-core Multicore for Embedded Processing Markets Demanding More Performance.

[54] A. Balkan, U. Vishkin, Mesh-of-trees and alternative interconnection networks for single-chip parallelism, IEEE Trans. Very Large Scale Integration (VLSI) Syst. 17 (10) (2009) 1419–1432.

[55] A. Rahimi, I. Loi, M.R. Kakoee, L. Benini, A Fully-synthesizable Single-cycle Interconnection Network for Shared-L1 Processor Clusters, 2011 Design, Automation & Test in Europe (2011) 1–6.

[56] M. Schoeberl, D.V. Chong, W. Puffitsch, J. Spars, A time-predictable memory network-on-chip, in: Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014), 2014.

[57] E. Ipek, O. Mutlu, J.F. Martínez, R. Caruana, Self-optimizing memory controllers: a reinforcement learning approach, in: Computer Architecture, 2008. ISCA'08. 35th International Symposium on, IEEE, 2008, pp. 39–50.

[58] Y. Kim, M. Papamichael, O. Mutlu, M. Harchol-Balter, Thread cluster memory scheduling: Exploiting differences in memory access behavior, in: Microarchitecture (MICRO), 2010 43rd Annual IEEE/ACM International Symposium on, IEEE, 2010, pp. 65–76.

[59] I. Hur, C. Lin, Memory scheduling for modern microprocessors, ACM Trans. Comput. Syst. (TOCS) 25 (4) (2007) 10.

[60] S. Bayliss, G.A. Constantinides, Methodology for designing statically scheduled application-specific SDRAM controllers using constrained local search, in: Field-Programmable Technology, 2009. FPT 2009. International Conference on, IEEE, 2009, pp. 304–307.

[61] B. Akesson, K. Goossens, Architectures and modeling of predictable memory controllers for improved system integration, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, IEEE, 2011, pp. 1–6.

[62] J. Reineke, I. Liu, H.D. Patel, S. Kim, E.A. Lee, Pret dram controller: Bank privatization for predictability and temporal isolation, in: Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, ACM, 2011, pp. 99–108.

[63] H. Shah, A. Raabe, A. Knoll, Bounding wcet of applications using sdram with priority based budget scheduling in mpsocs, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012, IEEE, 2012, pp. 665–670.

[64] Z.P. Wu, Y. Krish, R. Pellizzoni, Worst case analysis of dram latency in multi-requestor systems, in: Real-Time Systems Symposium (RTSS), 2013 IEEE 34th, IEEE, 2013, pp. 372–383.

[65] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, R.R. Rajkumar, Bounding memory interference delay in cots-based multi-core systems, in: The 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2014), 2014.

[66] M. Paolieri, E. Quiñones, F.J. Cazorla, Timing effects of ddr memory systems in hard real-time multicore architectures: issues and solutions, ACM Trans. Embedded Comput. Syst. (TECS) 12 (1s) (2013) 64.

[67] D. Wang, B. Ganesh, N. Tuaycharoen, K. Baynes, A. Jaleel, B. Jacob, DRAMsim: a memory system simulator, SIGARCH Comput. Archit. News 33 (2005) 100–107.

[68] H. Falk, P. Lokuciejewski, H. Theiling, Design of a wcet-aware c compiler, in: F. Mueller (Ed.), 6th International Workshop on Worst-Case Execution Time Analysis (WCET'06), Vol. 4 of OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2006.

[69] H. Falk, P. Lokuciejewski, A compiler framework for the reduction of worst-case execution times, Real-Time Syst. (2010) 1–50.

[70] C. Lattner, V.S. Adve, LLVM: A compilation framework for lifelong program analysis & transformation, in: International Symposium on Code Generation and Optimization (CGO'04), IEEE Computer Society, 2004, pp. 75–88.

[71] R. Kirner, P. Puschner, A. Prantl, Transforming flow information during code optimization for timing analysis, Real-Time Syst. 45 (1–2) (2010) 72–105.

[72] H.S. Negi, A. Roychoudhury, T. Mitra, Simplifying WCET analysis by code transformations, in: Workshop on Worst-Case Execution-Time Analysis (WCET), 2004.

[73] P. Puschner, The single-path approach towards WCET-analysable software, in: 2003 IEEE International Conference on Industrial Technology, Vol. 2, 2003, pp. 699–704.

[74] J. Yan, W. Zhang, A time-predictable VLIW processor and its compiler support, Real-Time Syst. 38 (1) (2008) 67–84.

[75] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, The worst-case execution time problem – overview of methods and survey of tools, Trans. Embedded Comput. Syst. 7 (3) (2008) 1–53.

[76] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: POPL '77: Proceedings of the 4th ACM Symposium on Principles of Programming Languages, ACM Press, 1977, pp. 238–252.

[77] R. Heckmann, M. Langenbach, S. Thesing, R. Wilhelm, The influence of processor architecture on the design and results of WCET tools, Proc. IEEE 91 (7) (2003) 1038–1054.

[78] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, R. Wilhelm, Predictability considerations in the design of multi-core embedded systems, in: Proceedings of Embedded Real Time Software and Systems, 2010.

[79] G. Gebhard, C. Cullmann, R. Heckmann, Software structure and WCET predictability, in: Bringing Theory to Practice: Predictability and Performance in Embedded Systems, DATE Workshop PPES 2011, Vol. 18 of OASICS, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Germany, 2011, pp. 1–10.

[80] A. Ermedahl, A Modular Tool Architecture for Worst-case Execution Time Analysis, Ph.D. Thesis, 2003.

[81] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, R. Wilhelm, Reliable and precise WCET determination for a real-life processor, in: T.A. Henzinger, C.M. Kirsch (Eds.), EMSOFT, Vol. 2211 of Lecture Notes in Computer Science, Springer, 2001, pp. 469–485.

[82] R. Heckmann, C. Ferdinand, Worst-case Execution Time Prediction by Static Program Analysis, Technical Report, AbsInt Angewandte Informatik GmbH (Online, last accessed November 2013).

[83] C. Ferdinand, R. Heckmann, Worst-case execution time – a tool provider's perspective, in: 11th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing ISORC 2008, IEEE Computer Society, 2008, pp. 340–345.

[84] S. Thesing, J. Souyris, R. Heckmann, F. Randimbivololona, M. Langenbach, R. Wilhelm, C. Ferdinand, An abstract interpretation-based timing validation of hard real-time avionics software, in: Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN 2003), IEEE Computer Society, 2003, pp. 625–632.

[85] J. Souyris, E. Le Pavec, G. Himbert, V. Jégu, G. Borios, R. Heckmann, Computing the worst case execution time of an avionics program by abstract interpretation, in: Proceedings of the 5th Intl Workshop on Worst-case Execution Time (WCET) Analysis, 2005, pp. 21–24.

[86] D. Kästner, R. Wilhelm, R. Heckmann, M. Schlickling, M. Pister, M. Jersak, K. Richter, C. Ferdinand, Timing validation of automotive software, in: 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA) 2008, Vol. 17 of Communications in Computer and Information Science (CCIS), Springer, 2008, pp. 93–107.

[87] A. Colin, I. Puaut, A modular and retargetable framework for tree-based wcet analysis, in: Real-Time Systems, 13th Euromicro Conference on, 2001, pp. 37–44.

[88] C. Ballabriga, H. Cassé, C. Rochange, P. Sainrat, OTAWA: an open toolbox for adaptive WCET analysis (regular paper), in: IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS), Waidhofen/Ybbs, Austria, 13/10/2010-15/10/2010, Springer, 2010, pp. 35–46.

[89] M. Schoeberl, W. Puffitsch, R.U. Pedersen, B. Huber, Worst-case Execution Time Analysis for a Java Processor, Software: Practice and Experience 40/6, 2010, 507–542.

[90] R. Systems, RapiTime Explained, Whitepaper, 2013. URL <http://www.rapitasystems.com/system/files/RapiTime%20Explained.pdf>

[91] B. Huber, W. Puffitsch, M. Schoeberl, 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), ACM, New York, NY, USA, 2010, pp. 26–35.

[92] P. Puschner, Experiments with WCET-oriented programming and the single-path architecture, in: Proc. 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2005.

[93] P. Puschner, R. Kirner, B. Huber, D. Prokesch, Compiling for time predictability, in: F. Ortmeier, P. Daniel (Eds.), Computer Safety, Reliability, and Security, Lecture Notes in Computer Science, Vol. 7613, Springer, Berlin/Heidelberg, 2012, pp. 382–391.

[94] C. Ferdinand, R. Wilhelm, Efficient and precise cache behavior prediction for real-time systems, Real-Time Syst. 17 (2–3) (1999) 131–181.

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

[96] K. Goossens, A. Hansson, The AEthereal network on chip after ten years: Goals, evolution, lessons, and future, in: Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC 2010), 2010, pp. 306–311.

[97] F. Brandner, M. Schoeberl, Static routing in symmetric real-time network-on-chips, in: Proceedings of the 20th International Conference on Real-Time and Network Systems (RTNS 2012), Pont a Mousson, France, 2012, pp. 61–70.

[98] R.B. Srensen, J. Spars, M. Ruvald Pedersen, J. Hjgaard, A metaheuristic scheduler for time division multiplexed networks-on-chip, in: IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS), IEEE, 2014.

[99] A. Hansson, M. Subburaman, K. Goossens, aelite: a flit-synchronous network on chip with composable and predictable services, in: Proceedings of the Conference on Design, Automation and Test in Europe (DATE 2009), Leuven, Belgium, 2009, pp. 250–255.

[100] J. Spars, E. Kasapaki, M. Schoeberl, 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, EDA Consortium, San Jose, CA, USA, 2013, pp. 1044–1047.
[101] W.J. Dally, J.W. Poulton, Digital Systems Engineering, Cambridge University Press, 1998.
[102] J. Spars, S. Furber (Eds.), Principles of Asynchronous Circuit Design – A Systems Perspective, Kluwer Academic Publishers, 2001.
[103] G. Plumbridge, J. Whitham, N. Audsley, Blueshell: a platform for rapid prototyping of multiprocessor NoCs and accelerators, in: Proceedings HEART Workshop, University of York, 2013.
[104] K. Chapman, Multiplexer Design Techniques for Datapath Performance with Minimized Routing Resources, Xilinx Application Note, 2012.
[105] J. Garside, N.C. Audsley, Prefetching across a shared memory tree within a network-on-chip architecture, in: System on Chip (SoC), 2013 International Symposium on, 2013, pp. 1–4.
[106] N.P. Jouppi, Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers, in: Proceedings of the 17th Annual International Symposium on Computer Architecture, Seattle, WA, 1990, pp. 364–373.
[107] J.W.C. Fu, J.H. Patel, B.L. Janssens, Stride directed prefetching in scalar processors, ACM SIGMICRO Newsletter 23 (1–2) (1992) 102–110.
[108] D. Joseph, D. Grunwald, Prefetching using Markov predictors, in: Proceedings of the 24th Annual International Symposium on Computer Architecture – ISCA '97, ACM Press, New York, New York, USA, 1997, pp. 252–263.
[109] K. Nesbit, J. Smith, Data cache prefetching using a global history buffer, in: 10th International Symposium on High Performance Computer Architecture (HPCA'04), IEEE, 2004, pp. 96–96.
[110] S. Goossens, T. Kouters, B. Akesson, K. Goossens, Memory-map selection for firm real-time SDRAM controllers, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, 2012, pp. 828–831.
[111] Y. Li, B. Akesson, K. Goossens, Dynamic command scheduling for real-time memory controllers, in: Proc. Euromicro Conference on Real-Time Systems (ECRTS), 2014, to appear.
[112] S. Hepp, F. Brandner, Splitting functions into single-entry regions, in: Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES '14, ACM, New York, NY, USA, 2014, pp. 17:1–17:10.
[113] S. Abbaspour, A. Jordan, F. Brandner, Lazy spilling for a time-predictable stack cache: implementation and analysis, in: H. Falk (Ed.), 14th International Workshop on Worst-Case Execution Time Analysis, OpenAccess Series in Informatics (OASIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Vol. 39, Dagstuhl, Germany, 2014, pp. 83–92.
[114] D. Prokesch, B. Huber, P. Puschner, Towards automated generation of time-predictable code, in: H. Falk (Ed.), 14th International Workshop on Worst-Case Execution Time Analysis, WCET 2014, July 8, 2014, Madrid, Spain, Vol. 39 of OASIcs, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014, pp. 103–112.
[115] G. Bernat, N. Holsti, Compiler support for WCET analysis: a wish list, in: WCET, 2003, pp. 65–69.
[116] B. Huber, D. Prokesch, P. Puschner, 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 (LCTES 2013), The Association for Computing Machinery, 2013, pp. 163–172.
[117] F. Brandner, S. Hepp, A. Jordan, Criticality: static profiling for real-time programs, Real-Time Syst. (2013) 1–34.
[118] C. Silva, Integrated Modular Avionics for Space Applications: Input/Output Module, Master's Thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, 2012.
[119] F.J. Cazorla, R. Gioiosa, M. Fernandez, E. Quiñones, Multicore OS Benchmark, Technical Report, RFQ- 3–13153/10/NL/JK, European Space Agency (ESA) and Barcelona Supercomputing Center (BSC), 2012.

Martin Schoeberl received his PhD from the Vienna University of Technology in 2005. From 2005 to 2010 he has been Assistant Professor at the Institute of Computer Engineering. He is now Associate Professor at the Technical University of Denmark. His research interest is on hard real-time systems, time-predictable computer architecture, and real-time Java. Martin Schoeberl has been involved in a number of national and international research projects: JEOPARD, CJ4ES, T-CREST, RTEMP, and the TACLe COST action. He is now technical lead of the EC funded project T-CREST. He has more then 100 publications in peer reviewed journals, conferences, and books.

Sahar Abbaspour is a PhD student at the Technical University of Denmark. She has received her Masters degree in Computer Engineering in 2011 from University of Tehran. Her research interest is on time-predictable computer architecture and currently she is working on time-predictable data caching.

Benny Akesson received his M.Sc. degree at Lund Institute of Technology, Sweden in 2005 and a Ph.D. from Eindhoven University of Technology, the Netherlands in 2010. Since then, he has been employed as a Postdoctoral Researcher at Eindhoven University of Technology, CISTER-ISEP Research Unit, and Czech Technical University in Prague. His research interests include design and analysis of multi-core real-time systems with shared resources.

Neil Audsley received a BSc (1984) and PhD (1993) from the Department of Computer Science at the University of York, UK. In 2013 he received a Personal Chair from the University of York, where he leads a substantial team researching Real-Time Embedded Systems. Specific areas of research include high performance real-time systems (including aspects of big data); real-time operating systems and their acceleration on FPGAs; real-time architectures, specifically memory hierarchies, Network-on-Chip and heterogeneous systems; scheduling, timing analysis and worst-case execution time; model-driven development.
Professor Audsley's research has been funded by a number of national (EPSRC) and european (EU) grants, including TEMPO, eMuCo, TouchHMore, MADES, JEOPARD, JUNIPER, T-CREST and DreamCloud. He has published widely, having upwards of 150 publications in peer reviewed journals, conferences and books.

Raffaele Capasso is a Project Manager at Intecs since 2002. He has a computer science diploma. With experience in the railway domain technologies, in defense domain technologies for combat managements systems and naval systems, in air traffic control domain technologies and radar data processing. He has been involved in several projects, including T-CREST (EC). Currently, he is Project Manager and Technical Leader in Intecs's for R&D projects in railway domain for Naples office.

Jamie Garside received his MEng degree at the University of York, and is currently working towards his PhD at the same university. His research interests include networks-on-chip, memory interconnect and prefetching.

Kees Goossens received his PhD in Computer Science from the University of Edinburgh in 1993 on hardware verification using embeddings of formal semantics of hardware description languages in proof systems. He worked for Philips/NXP Research from 1995 to 2010 on networks on chip for consumer electronics, where real-time performance, predictability, and costs are major constraints. He was part-time full professor at Delft university from 2007 to 2010, and is now full professor at the Eindhoven university of technology, where his research focusses on composable (virtualised), predictable (real-time), low-power embedded systems. He published 3 books, 150+ papers, and 16 patents

Sven Goossens was born in Wouw, The Netherlands in 1986. He received a MSc in embedded systems from the Eindhoven University of Technology in 2010, and is currently a PhD candidate at the same university. His research interests include mixed time-criticality systems, composability and SDRAM controllers.

Scott Hansen is Director for European Projects at The Open Group and has been the project leader for 16 previous European Commission funded projects including large RTD STREP and IP projects, as well as accompanying measures, and thematic networks. Based in the Brussels, he co-ordinates the research efforts of The Open Group in Europe amongst European members, as well as with other European standards bodies, and industry consortia, where he sits on the ICT Standards Board, a European Commission funded grouping of Standards Organisations and Industry consortia. He holds degrees in computer science, business management and industrial engineering, and has over 20 years experience working in both large multi-national organisations and smaller start-ups managing technology development, deployment, exploitation as well as the financial and administrative disciplines associated with successful introduction of new technologies.

Reinhold Heckmann studied Computer Science at Saarland University in Saarbruecken, Germany, where he received the Dr. rer. nat. degree in 1991. After being Lecturing Assistant at Saarland University and Research Fellow at Imperial College, London, he became Senior Researcher at AbsInt Angewandte Informatik GmbH in 2000. Within AbsInt, he is working on the foundations of timing analysis for hard-real time systems. This work has been pursued in the context of various European research projects funded by the FP5, FP6, FP7, Artemis, and ITEA programmes, including INTERESTED (FP7 IST-214889), PREDATOR (FP7 IST-216008), ALL-TIMES (FP7 IST-215068), T-CREST (FP7 IST-288008), and CERTAINTY (FP7 IST-288175).

Stefan Hepp received his MSc in computer engineering from Vienna University of Technology in 2011, where he is currently working toward his Ph.D. under the supervision of Professor Jens Knoop. His research interests include worst-case execution time oriented code optimizations, cache analysis techniques and time-predictable architectures.

Benedikt Huber received his MSc in computational intelligence from Vienna University of Technology in 2009. Since then, he worked as a research and teaching assistant at the same university's real-time systems group. His research focus is on WCET analysis in the context of time-predictable system design.

Alexander Jordan received his PhD in Computer Science from the Vienna University of Technology in 2014 and is currently employed as a postdoctoral research assistant at DTU in Denmark. Working as an Embedded Software engineer for several years, his research interests nowadays include code generation techniques, program analysis and optimization.

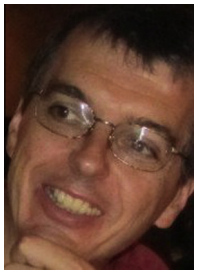
Evangelia Kasapaki has received her BSc and MSc from Computer Science Department, University of Crete, Greece in 2006 and 2008 respectively and is currently a PhD student in Technical University of Denmark. She has worked as a Electronic Design Automation software engineer from 2008 to 2011, in Nanochronous Logic, Inc, when she started her PhD. Her research interests include asynchronous design, Networks-on-Chip and SoC design, real-time systems and Electronic Design Automation.

Jens Knoop is a full professor at the Vienna University of Technology, where he leads the languages and compilers group. His research interests include program analysis, optimization, and verification, especially of safety-critical real-time systems. Jens Knoop is a member of the IEEE and the ACM.

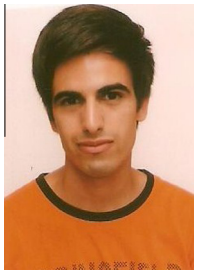
Yonghui Li got his bachelor and master degrees both from Xidian University in 2009 and 2012, respectively. Since May 2012, he is working towards a PhD at Eindhoven University of Technology. His research interests include Networks-on-Chip, memory controllers, and real-time systems.

Daniel Prokesch received his MSc in computer engineering from Vienna University of Technology in 2011. Currently, he is a PhD candidate under supervision of Professor Peter Puschner at the same university. His research interests include time-predictable system design, WCET analysis and code generation techniques.

Wolfgang Puffitsch is currently a postdoc researcher at DTU Compute in Copenhagen, Denmark, working on time-predictable computer architectures in the scope of the RTEMP project. From May 2012 to May 2013, he was a postdoc researcher at the DTIM group of ONERA in Toulouse, France, in the scope of the TOAST project. Before that, since January 2008, he worked as research and teaching assistant at the Institute of Computer Engineering at the Vienna University of Technology in Vienna, Austria, where he defended his PhD thesis on real-time garbage collection in March 2012.

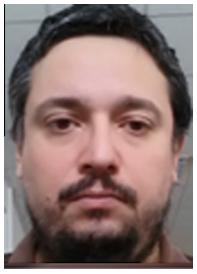


Peter Puschner is a professor in computer science at the Vienna University of Technology. His main research interest is on hard real-time systems for safety-critical applications, with a focus on the worst-case execution time (WCET) analysis and software/hardware architectures for time-predictable computing systems.
He has published more than 100 refereed conference and journal papers, received one patent, and was a guest editor of two special journal issues on WCET analysis. P. Puschner chaired the PC of ISORC 2003 and ECRTS 2004, was the general chair of ECRTS 2002, ISORC 2004, and SEUS 2010, and was the local chair of the IEEE Real-Time Systems Symposium in 2011. He is a member of the Editorial Board of the Springer Real-Time Systems journal and is in charge of the steering committees of the workshop series on worst-case execution-time analysis (WCET) and the International Workshop on Software Technologies for Future Embedded and Ubiquitous Computing Systems (SEUS).
Prof. Puschner received a Marie-Curie Category-30 fellowship and spent one year (2000) as a visiting researcher at the University of York, England. He has been involved in a number of national and international research projects, including the following projects funded by the European Commission: SETTA, NextTTA, CaberNet, Artist, Artist2, ArtistDesign, T-CREST, MultiPARTES, and the TACLe COST action. P. Puschner is a member of the IEEE Computer Society, IFIP working group 10.2 on Embedded Systems, Euromicro, the Austrian Computer Society (OCG), and the Marie-Curie Fellowship Association.



André Rocha is a software engineer at GMV since 2012, staffed to the Aerospace, Homeland Security and Defense department. He finished his Integrated Master of Science (MSc) degree in Aerospace Engineering, in the field of Avionics, at Instituto Superior Técnico in December 2011, and started working as a junior IT consultant at Everis Portugal by that time. He took part in several software development projects, targeting different platforms and customers, with a special focus on the energy sector and, in particular, the EDP group.
André enters GMV Portugal in 2012, taking on the DORATHEA project where he acquires skills in Air Traffic Management and Security Risk Assessment. In the scope of this project, he participates in the SWIM Master Class 2013. Afterwards, André joins the T-CREST project team, being responsible for not only porting the RTEMS OS to the new T-CREST processor architecture, but also adapting various avionics use case applications to the new platform.

Cláudio Silva is a technical manager and embedded software developer at GMV since 2011. With a master's degree in Aerospace engineering, he has broad experience in the aeronautics and space domain technologies, including experience in embedded software development, real time systems, and driver development. He has participated in projects focusing Integrated Modular Avionics, time and space partitioning, real time systems and on-board software development. Currently, he is Technical Leader in GMV's IMA Activities and product manager of GMV's real-time operating system AIR. Mr. Cláudio has been involved in several European Commission, European Space Agency and national projects, including T-CREST (EC), IMA for Space (ESA) and MultIMA (ESA). Moreover, he is presently a recognizable active member of the RTEMS open-source community.



Alessandro Tocchi is a software and firmware test/development engineer at Intecs since 2011. He has a masters degree in electronic engineering. With experience in the railway domain technologies, he has worked as an embedded software technician, systems integration specialist, and real time system analyst. He has been involved in several projects, including T-CREST (EC).

Jens Sparsø is a professor at the Technical University of Denmark (DTU). His research interests include architecture and design of VLSI systems, application specific processors, low power design techniques, design of asynchronous circuits, and networks-on-chip. J. Sparsø has published more than 70 refereed conference and journal papers and is coauthor of the book Principles of Asynchronous Circuit Design - A Systems Perspective (Kluwer, 2001). J. Sparsø received the Radio-Parts Award and the Reinholdt W. Jorck Award in 1992 and 2003, in recognition of his research on integrated circuits and systems, and he received the best paper award at ASYNC 2005. J. Sparsø is a member of the steering committees for the IEEE Intl. Symposium on Asynchronous Circuits and Systems (ASYNC) and the ACM/IEEE Intl. Symposium on Networks-on-Chip (NOCS).